KTH Computer Science
and Communication

Space in Proof Complexity

MARC VINYALS

Doctoral Thesis
Stockholm, Sweden 2017

TRITA CSC-A-2017:15
ISSN 1653-5723
ISRN KTH/CSC/A--17/15--SE
ISBN 978-91-7729-422-1

KTH School of Computer Science and Communication
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi fredagen den 9 juni 2017 klockan 14.00 i E2, Kungl Tekniska högskolan, Lindstedtsvägen 3, Stockholm.

Tryck: Universitetsservice US AB

Abstract

Propositional proof complexity is the study of the resources that are needed to prove formulas in propositional logic. In this thesis we are concerned with the size and space of proofs, and in particular with the latter.

There are different approaches to reasoning that are captured by corresponding proof systems, each with its own strengths and weaknesses and need for resources. The simplest and most well studied proof system is resolution, and we try to get our understanding of other proof systems closer to that of resolution. In particular we look at polynomial calculus, which captures reasoning with polynomials, and cutting planes, which captures reasoning with linear inequalities.

In resolution we can prove a space lower bound just by proving a lower bound on the widest clause in the proof, that is showing that any proof must have a large clause. We prove a similar relation between resolution width and polynomial calculus space that let us derive space lower bounds, and we use it to separate degree and space.

For cutting planes we show length-space trade-offs. This is, there are formulas that have a proof in small space and a proof in small length, but there is no proof that can optimize both measures at the same time.

We introduce a new measure of space, cumulative space, that accounts for the space used throughout a proof rather than only its maximum. This is mostly exploratory work, but we can also prove new results for the usual space measure, for instance that there are trade-off results for resolution where every short proof not only needs to use large space, but it needs to do so most of the time.

We define a new proof system that aims to capture the power of current SAT solvers based on conflict-driven clause learning (CDCL), and we show a rich landscape of length-space trade-offs comparable to those in resolution.

To prove these results we build and use tools from other areas of computational complexity. One area is pebble games, which are very simple computational models that are useful for modelling space. In addition to results with direct applications to proof complexity, we show that pebble game cost is PSPACE-hard to approximate.

Another area is communication complexity, which is the study of the amount of communication that is needed to solve a problem when its description is shared by multiple parties. We prove a simulation theorem that relates the query complexity of a function with the communication complexity of a composed function.

Sammanfattning

Satsbeviskomplexitet studerar vilka resurser som behövs för att bevisa satslogiska formler. I denna avhandling är vi intresserad med längden och minnet av bevis, och i synnerhet det senare.

Det finns olika metoder för logiska resonemang, och dessa har motsvarande bevissystem, vart och ett med egna styrkor och svagheter och behov av resurser. Det enklaste och mest välstuderade bevissystemet är resolution, och vi försöker förstå andra bevissystem lika väl. I synnerhet undersöker vi polynomkalkyl, som uttrycker resonemang med polynom, och på skärande plan, som uttrycker resonemang med linjära olikheter.

I resolution kan vi bevisa en undre gräns för minne genom att bevisa en undre gräns på den bredaste klausulen i beviset, det vill säga genom att visa att varje bevis måste ha en stor klausul. Vi visar ett liknande förhållande mellan resolutionbredd och polynomkalkylminne som låter oss härleda undre gränser för minne, och vi använder det för att skilja gradtal från minne.

För snittplan visar vi avvägningar mellan längd och minne. Det finns formler som har ett bevis i litet minne och ett bevis i liten längd, men inget bevis som optimerar båda måtten samtidigt.

Vi introducerar ett nytt mått på minne, kumulativt minne, som bokför det totala minne som används genom hela beviset, istället för endast det maximala. Det här är främst förundersökningsarbete, men vi kan också bevisa nya resultat för den vanliga minnemåttet, till exempel att det finns avvägningsresultat för resolution där varje kort bevis inte bara behöver använda stort minne, men måste göra det under den mesta delen av tiden.

Vi definierar ett nytt bevissystem som syftar till att uttrycka kraften av de nuvarande SAT-lösare, och vi visar ett rikt landskap av längdminne-avvägningar jämförbara med dem i resolution.

För att bevisa dessa resultat bygger vi och använder verktyg från andra områden av beräkningskomplexitet. Ett område är stenspel, som är mycket enkla beräkningsmodeller som är användbara för att modellera minne. Förutom resultat med direkta tillämpningar på beviskomplexitet visar vi att stenspelskostnad är PSPACE-svårt att approximera.

Ett annat område är kommunikationskomplexitet, vilket är studien av den mängd kommunikation som behövs för att lösa ett problem när dess beskrivning delas av flera parter. Vi visar en simuleringssats som relaterar beslutsträdkomplexiteten av en funktion med kommunikationskomplexiteten av en sammansatt funktion.

Acknowledgements

I would like to thank all who helped me completing this thesis. Jakob, my advisor, had a big role. Jakob, you introduced me to some fascinating research topics, we had long discussions about them, and you pushed me to write results properly. You also managed to give a positive spin to some research approaches when they most seemed doomed. Thank you for that. Thanks to Johan as well, who helped with my supervision and kept his door open for discussions.

Mladen, Susanna, Massimo, Ilario, it is always fun to sit in front of a whiteboard with you. Thanks for contributing good ideas, shooting down my stupid ones, and making research enjoyable. After spending a few years with you, I learned quite a bit when our discussions got derailed into consciousness, idioms, progressive rock, or photography.

Christoph, Jan, Jesús, Yuval, Siu Man, Jan, Joël, Li-Yang, Nicola, Navid, Arkadev, Sagnik, the same applies to you: it was nice to stare at a wall together, and I can also remember some interesting moments that were not strictly research-related.

Adam, Andreas, Benny, Björn, Cenny, Emma, Freyr, Guillermo, Gunnar, Hamed, Hojat, Jan, Jana, Joseph, Karl, Mateus, Musard, Oliver, Pedro, Raj, Sangxia, Siavash, Thatchapol, Torbjörn, Xin, thanks for making the TCS kitchen a weird place. And Lukáš, thanks for making the outdoors a weird place.

Per, thanks for reading the thesis, providing good suggestions, and helping with debugging it.

Thanks to my family and to my friends for your support and for preventing me from going completely insane.

Contents

Part I

Thesis

Chapter 1

Introduction

One day last winter I was sitting with some lifelong friends at a lifelong friend's wine bar. Since some of them are fond of riddles, I posed them the following question. "Picture seven spots, one after another, and three pebbles. You may place or remove a pebble from a spot if there is already a pebble in the spot before it. Since there is no spot before the first one, you may always place or remove a pebble from it. Can you place a pebble in the seventh spot?" (see Figure 1.1)

When they came back with the solution the following day, I had to confess that I had dedicated quite some more time to this and related questions. In fact, one could say that I spent the last five years playing the pebble game, and trying to become better at it. While this is a simplistic view, it is true that at the core of a theorem involving the space complexity of computations one can often find the pebble game.

This is not so surprising if we look at how we perform computations. Let us start with a small example and say that we want to compute $x + yz$. The most obvious way to perform the computation is to load x, y, and z into three registers, then compute yz into a new register, and finally compute $x + yz$ into another new register, for a total of five registers used. But we could easily do better by placing the final result of the

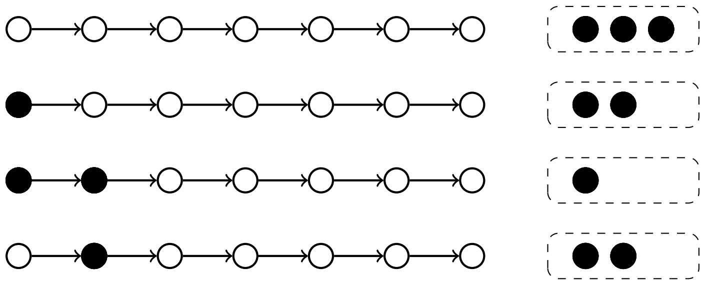


Figure 1.1: Some valid moves in the pebble riddle

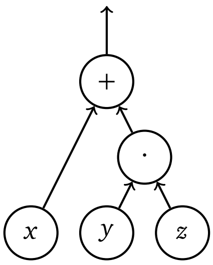


Figure 1.2: Computation $x + yz$ laid down as a graph

computation into the register that used to contain y, since we do not need to know the value of y anymore after we finished computing yz, so four registers are enough. Or we could do even better by loading only y and z, computing yz into a new register, loading x into the register that used to contain y, and placing the final result of the computation into the register that used to contain z, using only three registers. It is not hard to convince ourselves that this is the best we can do, but if we were to optimize a computation with a hundred operations we would have a harder time.

Observe that in the previous reasoning we did not care about which operations we were performing but only about whether we had the operands in memory. If we represent the computation as a graph as in Figure 1.2, then we can use the language of pebbling to say *place a pebble* instead of compute and *remove a pebble* instead of forget. In a regular computation we can always forget results, so the rules of the pebble game that models usual computations are instead “You may place a pebble on a spot if there are already pebbles in the spots before it and remove a pebble at will”.

Minimizing the amount of registers needed for a computation was among the first applications of the pebble game we described [171], and many more have been found since then, but in this thesis we are going to use it to study space in proof complexity.

Proof complexity is the study of the resources that proofs need. By resources we mean the richness of the logic system in which the proof is expressed, the length of the proof, the memory needed to verify it, or the runtime of an algorithm that can generate it, among others. It is a subfield of both mathematical logic, since we answer questions about which axioms are needed to efficiently prove a statement, and of computational complexity, where we are interested in computational resources such as time and space.

In this thesis we focus at the second type of questions, and we only look at the most simple kind of statements to prove: propositional Boolean formulas. These are expressions formed by Boolean variables joined by negations, conjunctions, and disjunctions, but without any quantifiers.

For example, let us try to prove a very particular case of the pigeonhole principle: that 3 pigeons do not fit into 2 pigeon-sized holes. Let us have a variable x_{ij} that is true if and only if pigeon number i is assigned into hole number j. Then we want to prove that either

1. some pigeon is not assigned to any hole

$$(\overline{x_{11}} \wedge \overline{x_{12}}) \vee (\overline{x_{21}} \wedge \overline{x_{22}}) \vee (\overline{x_{31}} \wedge \overline{x_{32}}) \ \text{, or}$$

2. some hole is assigned more than one pigeon

$$\begin{gathered}(x_{11} \wedge x_{21}) \vee (x_{11} \wedge x_{31}) \vee (x_{21} \wedge x_{31}) \\ (x_{12} \wedge x_{22}) \vee (x_{12} \wedge x_{32}) \vee (x_{21} \wedge x_{31}) \ .\end{gathered}$$

This is, we want to show that the previous formula is a tautology. We could do that by directly using the definition of tautology: the formula evaluates to true under each truth value assignment to variables. So, for the assignment $x_{11} \mapsto \text{FALSE}$, $x_{12} \mapsto \text{FALSE}$, $x_{21} \mapsto \text{FALSE}$, $x_{22} \mapsto \text{FALSE}$, $x_{31} \mapsto \text{FALSE}$, $x_{32} \mapsto \text{FALSE}$ we can verify that the formula evaluates to

$$(\overline{\text{FALSE}} \wedge \overline{\text{FALSE}}) \vee \cdots = (\text{TRUE} \wedge \text{TRUE}) \vee \cdots = \text{TRUE} \vee \cdots = \text{TRUE}$$

and we can actually verify that this is the case with each of the 2^6 truth value assignments.

Now, this may only seem to be a tedious process, but if we had 10 pigeons and 9 holes (so 90 variables) then even if we could evaluate one assignment per nanosecond it would take us more time than the expected life of the sun to evaluate all of the 2^{90} assignments.

Clearly we need a more efficient way of reasoning. It is more convenient to argue by contradiction, since then we have something to start working with, so we can also try to prove that the following formula, which we obtain by negating and applying De Morgan's rules, is a contradiction.

1. Every pigeon is assigned to some hole

$$(x_{11} \vee x_{12}) \wedge (x_{21} \vee x_{22}) \wedge (x_{31} \vee x_{32}) \ \text{, and}$$

2. no hole is assigned more than one pigeon

$$\begin{gathered}(\overline{x_{11}} \vee \overline{x_{21}}) \wedge (\overline{x_{11}} \vee \overline{x_{31}}) \wedge (\overline{x_{21}} \vee \overline{x_{31}}) \\ (\overline{x_{12}} \vee \overline{x_{22}}) \wedge (\overline{x_{12}} \vee \overline{x_{32}}) \wedge (\overline{x_{22}} \vee \overline{x_{32}}) \ .\end{gathered}$$

Let us, for the sake of contradiction, assume that there is some assignment that satisfies the formula. In particular, since the formula is a conjunction of subformulas, that assignment satisfies each subformula. Now we can pick the two subformulas $\overline{x_{22}} \vee \overline{x_{32}}$ and $x_{31} \vee x_{32}$ and do some case analysis. If the assignment sets $x_{32} \mapsto \text{TRUE}$, then the first subformula becomes $\overline{x_{22}} \vee \overline{\text{TRUE}} = \overline{x_{22}}$. Otherwise the assignment sets $x_{32} \mapsto \text{FALSE}$, in which case the second subformula becomes $x_{31} \vee \text{FALSE} = x_{31}$. So we deduced that the assignment satisfies either $\overline{x_{22}}$ or x_{31}, and hence it satisfies the new formula $\overline{x_{22}} \vee x_{31}$.

$$\begin{tikzcd}
 & & \bot & & & \\
 & \overline{x_{12}} \arrow[ur] & & & x_{12} \arrow[ull] & \\
\overline{x_{12}}\vee\overline{x_{22}} \arrow[ur] & & & x_{11}\vee x_{12} \arrow[ur] & & \\
 & \overline{x_{12}}\vee x_{22} \arrow[uu] & & & \overline{x_{11}} \arrow[uu] & \\
\overline{x_{12}}\vee\overline{x_{32}} \arrow[ur] & & & \overline{x_{11}}\vee\overline{x_{21}} \arrow[ur] & & \\
 & x_{22}\vee x_{32} \arrow[uu] & & & \overline{x_{11}}\vee x_{21} \arrow[uu] & \\
 & & & \overline{x_{11}}\vee\overline{x_{31}} \arrow[ur] & & \\
 & \overline{x_{21}}\vee x_{32} \arrow[uu] & & & x_{21}\vee x_{31} \arrow[uu] & \\
\overline{x_{21}}\vee\overline{x_{31}} \arrow[ur] & & x_{21}\vee x_{22} \arrow[uuul] \arrow[urr] & & & \\
 & & & & \overline{x_{22}}\vee x_{31} \arrow[uu] & \\
 & & x_{31}\vee x_{32} \arrow[uuul] \arrow[urr] & & & \overline{x_{22}}\vee\overline{x_{32}} \arrow[ul]
\end{tikzcd}$$

Figure 1.3: Proof that 3 pigeons do not fit into 2 holes

Now we can apply the same case analysis with the new formula $\overline{x_{22}} \vee x_{31}$ and the original subformula $x_{21} \vee x_{22}$ to obtain $x_{21} \vee x_{31}$, which means that the first hole is occupied by either the second or third pigeons. Some more case analysis leads to proving that the first pigeon must occupy the second hole, and that the first pigeon does not occupy the second hole, which is a contradiction (see Figure 1.3).

We can see that the proof in Figure 1.3 takes 10 steps, but how many of these do we need to keep in memory? In order to verify that a new formula is valid, we just need to know that its two predecessors are valid. But if we identify valid subformulas with pebbles, these are exactly the rules of the pebble game! Hence the space that we need is the cost of the pebble game on the proof viewed as graph, which in this case is 4.

Let us stop for a moment and reflect on what we have seen so far. We discussed how by using a different kind of reasoning we were able to come up with a shorter proof. Is this a small difference or will it keep increasing as we increase the number of pigeons and holes? Are there ways of reasoning that are more efficient than others?

We also saw that the proof by case analysis is easy to verify, but we did not discuss at all how did we find it. In contrast, the brute-force proof just required enumerating all assignments in some order. So, even if there is a better proof, will we be able to find it? For all we know, coming up with the proof might require much more work than verifying it. Could we even automate the proof finding process?[1]

Going back to discussing the scale of case-analysis proofs, it turns out that these proofs can be made exponentially smaller than listing all assignments, but they still need to be of exponential size [98]. Hence we are left again with the need for another kind of reasoning. Let us try an algebraic approach, where instead of variables taking values over $\{\text{TRUE}, \text{FALSE}\}$ they take values over $\{0, 1\}$. The meaning of a variable x_{ij} can be interpreted as the number of pigeons labelled i that are assigned into hole j. We can express our constraints more succinctly as

1. every pigeon is assigned to some hole

$$(x_{11} + x_{12} \geq 1) \wedge (x_{21} + x_{22} \geq 1) \wedge (x_{31} + x_{32} \geq 1) \text{ , and}$$

2. no hole is assigned more than one pigeon

$$(x_{11} + x_{21} + x_{31} \leq 1) \wedge (x_{12} + x_{22} + x_{32} \leq 1) \text{ .}$$

Now we just need to add up all inequalities together to obtain

$$x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} + 2 \geq x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} + 3$$

which simplifies into the contradiction $2 \geq 3$. This approach generalizes to larger numbers of pigeons and we obtain proofs of size comparable to the formula we want to prove.

Hopefully we have a better idea of what proof complexity is, but we did not discuss why we want to study these questions. The main reason is to better understand the nature of computation. The goal seems to be far away, but we still have two approaches to gain some knowledge. One is to assume some conjectures and prove results conditioned on them and the other, in which proof complexity falls, is to limit what we allow computations to do and to prove results in restricted models.

Proof complexity has practical applications in analyzing the running time of algorithms that look for satisfying assignments to propositional Boolean formulas (SAT solvers). Furthermore, during the 2016 SAT competition, where SAT solvers are tested on a variety of formulas, some parallel solvers were found to perform worse than their sequential counterparts. The reason? Parallel solvers were using too much memory [17]. Hopefully a better understanding of space in proof complexity will help us understand memory usage and build better solvers.

[1]In fact Figure 1.3 was automatically generated, but we are getting ahead of ourselves.

Chapter 2

Background

2.1 Pebbling

Since we will be using pebbling an awful lot, let us start by properly defining some pebble games. The *black-white pebble game* on a directed acyclic graph (DAG) G with fan-in 2 and a single sink z is a single player game where the player has to leave a black pebble on the sink and no pebbles elsewhere. To do that at every move they can

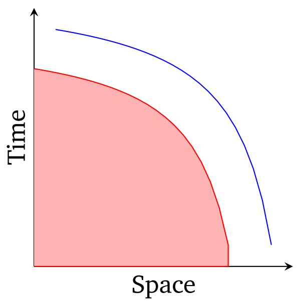
1. place a black pebble on a vertex if its immediate predecessors have pebbles,
2. remove a black pebble at any time,
3. place a white pebble at any time, or
4. remove a white pebble from a vertex if its immediate predecessors have pebbles.

In the *black* version of the game the player only uses black pebbles, so only rules 1 and 2 apply. In the *reversible* version a pebble can only be placed or removed if its immediate predecessors have pebbles, so the player can

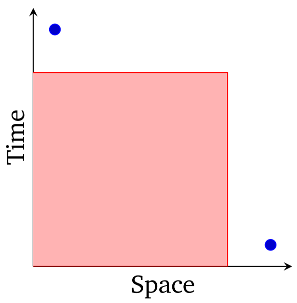
1. place a black pebble on a vertex if its immediate predecessors have pebbles, or
2. remove a black pebble on a vertex if its immediate predecessors have pebbles.

This is equivalent of asking for a black pebbling to be valid when we run it in reverse, hence the name. When we play on a path graph, the rules specialize to the riddle of Figure 1.1.

The time of a pebbling is the number of moves, and the space is the maximum number of pebbles over the graph at the same time. The time and space of a graph are defined by taking the minimum over all valid pebblings. Observe that reversible pebblings are a particular class of black pebblings, which are a particular class of black-white pebblings, so any upper bounds on these measures that we prove for reversible pebbling are valid



(a) A continuous trade-off



(b) A discrete trade-off

Figure 2.1: Generic trade-offs

for black and black-white, and any lower bounds that we prove for black-white pebbling are valid for the other two.

There is a trivial $O(n)$ upper bound on the space of a graph. More interestingly, there is also a $O(n/\log n)$ reversible pebbling space upper bound for any graph [74, 54] that generalizes an earlier upper bound for black pebbling [104], with a matching $\Omega(n/\log n)$ black-white pebbling lower bound for a family of hard graphs [130].

A pebbling in space n can always be done in time n, but if we are optimizing space we may be forced to pebble the same part of the graph again and again, leaving us with a choice between time and space efficiency. Let us give some examples of trade-offs with different ranges of parameters. On the constant space side we have that the *bit reversal* graphs of [130] can be black-white pebbled in either of time n, space 3, or any combination of space $s = O(\sqrt{n})$ and time $O(n^2/s^2)$, but any black-white pebbling must satisfy that $t = \Omega(n^2/s^2)$. This is a continuous trade-off, in the sense that we can choose an upper bound in a curve as in Figure 2.1a, where we draw upper bounds in blue and the lower bound region in red. On the other extreme we have *stacks of superconcentrators* of carefully chosen sizes [130], which can be black-white pebbled in either time n or space $O(\sqrt[8]{n})$, but any black-white pebbling in space $O(n^{1/4-\epsilon})$ requires time $\exp(n^{\Omega(1)})$. This is a discrete trade-off where once we cross a threshold pebblings become substantially different, as in Figure 2.1b.

Black pebbling space is at most quadratic on the black-white space [137], with a matching separation in [117]—albeit for graphs where the space is only logarithmic. Reversible pebbling space is at most $s^2 \log n$ [126], where s is the black pebbling space and n is the order of the graph, and the best separation we know is a factor $\log n$, which we show in Paper E.

The reversible pebble game is related to an apparently different game of Dymond and Tompa [74]. This is a two-player pebble game where we call the players Pebbler and Challenger. One of the pebbles is always challenged (initially we assume that a virtual successor of the sink is challenged). On each round of the game Pebbler adds

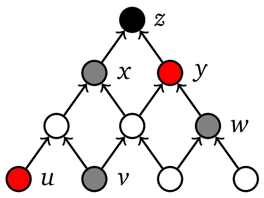


Figure 2.2: Example moves in the Dymond–Tompa game

some pebbles to the graph (but does not remove any), and Challenger may *jump* the challenge to one of the newly placed pebbles (but not older pebbles) or let the challenge *stay* in the currently challenged pebble. The game ends when, after a Challenger move, all the immediate predecessors of the challenged pebble are pebbled.

For instance, in Figure 2.2 the challenged pebble is on vertex z. Pebbler just added pebbles on u and y, and there are old pebbles over x, w, and v. Challenger is allowed to stay in z, in which case the game ends because x and y have pebbles, to jump to u, in which case the game ends because u has no predecessors, or to jump to y, in which case the game continues.

The *cost* of a Pebbler strategy is the maximum number of pebbles in the graph against any Challenger strategy, and the cost of a graph is the cost of an optimal Pebbler strategy.

The relation to one-player pebbling is that the reversible pebbling space of a graph is equal to its Dymond–Tompa cost [54]. The Dymond–Tompa cost—and hence reversible pebbling space—is also equivalent to the query complexity of the problem where each vertex in a graph is assigned a truth value and we have to find a false vertex whose predecessors are all true.

2.2 Proof Systems

Let us continue by laying down the formal, low level foundations of proof complexity; even if our later discussion is at a higher level, they can help clarify some statements.

A proof system $\mathcal{P}$ is a language consisting of tuples (F, π) of a formula and a proof. A propositional proof system is such that F is a propositional formula. We assume that formulas are given in Conjunctive Normal Form (CNF), that is $F = \bigwedge \bigvee x_i^{b_i}$, where we define $x_i^0 = \overline{x_i}$ and $x_i^1 = x_i$. It is convenient to identify TRUE with 1 and FALSE with 0, so that x^b is satisfied by the assignment $x \mapsto b$. We refer to a possibly negated variable as a literal, a disjunction of at most k literals as a k-clause, and a conjunction of k-clauses as a k-CNF.

A propositional proof system $\mathcal{P}$ is sound if it can only refute unsatisfiable formulas, i.e., $(F, \pi) \in \mathcal{P}$ implies that $F \in \text{UNSAT}$, and it is complete if every unsatisfiable formula has a proof, i.e., $F \in \text{UNSAT}$ implies that $\exists \pi : (F, \pi) \in \mathcal{P}$. From now on we assume that all proof systems are propositional, sound, and complete, and we use prove and refute interchangeably.

A proof system is polynomially verifiable if it is in P. It is polynomially bounded if every unsatisfiable formula has a proof of polynomial size, i.e., $F \in$ UNSAT implies that $\exists \pi : (F, \pi) \in \mathcal{P} \land |\pi| = \text{poly}(|F|)$. Observe that if there existed a polynomially verifiable, polynomially bounded proof system then coNP = NP, since we could take π as an NP witness for $F \in$ UNSAT.

A proof system $\mathcal{P}$ polynomially simulates a proof system $\mathcal{R}$ if $(F, \rho) \in \mathcal{R}$ implies that $\exists \pi : (F, \pi) \in \mathcal{P} \land |\pi| = \text{poly}(|\rho|)$. We say that $\mathcal{P}$ is (strictly) stronger than $\mathcal{R}$ if $\mathcal{P}$ polynomially simulates $\mathcal{R}$ but the converse is not true, and that the proof systems are incomparable if neither holds.

For us, back to a higher level, a proof system is just a set of rules to construct proofs with. All of our proofs fit in the *line configurations* framework of [2]. A proof is a sequence of configurations $\mathbb{C}_1, \ldots, \mathbb{C}_L$, and a configuration is a set of lines. The proof system determines what is a valid line. A configuration can be obtained from the previous one by either of:

Axiom download Add (the translation of) a clause in F as a new line.

Inference Add a line that follows from applying one inference rule of the proof system to (a subset of) the previous configuration.

Erasure Remove a line from the previous configuration.

This framework is enough to define the *length* of a proof, which is the number of configurations, the *size*, which is the sum of line sizes over all downloaded and inferred lines, the *(line) space*, which is the number of lines in the largest configuration, the *total space*, which is the largest sum of line sizes in a single configuration, and the *variable space*, which is the largest number of distinct variables in a single configuration.

The same measures are defined for a *formula* by taking the minimum over all proofs.

The *inference graph* of a proof is a graph where vertices are all downloaded and inferred lines, and there is an edge from vertex u to v if u was used as a premise in the inference rule used to derive v. It is not hard to see that a sequence of configurations is equivalent to a pebbling strategy for the inference graph. In fact, proofs are sometimes defined as the inference graph alone, in which case the line space is the space of an optimal pebbling strategy. The *depth* of a proof is the depth of its inference graph.

2.3 Formula Reference

In the following discussion we need to refer to some formulas that are well-known in proof complexity, so we list them together for convenience. It is not necessary to become familiar with the symbolic expressions, which at this point can be viewed as illustrations.

Pebbling Formulas The pebbling formula Peb_G of a graph G [29] has constraints claiming that the sources of the graph are true, that if all the predecessors of a vertex are true, then the vertex itself is true, and that the sink is false.

$$\bigvee_{u \in pred(v)} \overline{u} \vee v \qquad v \in V(G) \qquad \text{(Pebbling axioms)}$$

$$\overline{z} \qquad \text{(Sink axiom)}$$

This formula is the connection between pebble games and proof complexity, and it will make recurrent appearances throughout this thesis. We will most often use *substituted* pebbling formulas, in which we replace each variable by a gadget.

Pigeonhole Principle The pigeonhole principle claims that there is an injective total mapping from $[n+1]$ into $[n]$ (or that $n+1$ pigeons fit into n holes). Variable x_{ij} stands for pigeon i being mapped to hole j.

$$\bigvee_{j \in [n]} x_{ij} \qquad i \in [n+1] \qquad \text{(Pigeon axioms)}$$

$$\overline{x_{ij}} \vee \overline{x_{i'j}} \qquad i \neq i', j \in [n] \qquad \text{(Hole axioms)}$$

We can optionally require the mapping to be a function or for it to be surjective, leading to four variants of the principle.

$$\overline{x_{ij}} \vee \overline{x_{ij'}} \qquad i \in [n+1], j \neq j' \qquad \text{(Functionality axioms)}$$

$$\bigvee_{i \in [n+1]} x_{ij} \qquad j \in [n] \qquad \text{(Onto axioms)}$$

This formula is a source for many lower bounds.

Tseitin Formulas The Tseitin Formula of a graph G encodes the (negated) handshaking lemma: that the sum of vertex degrees is an even number. It is a particular form of a linear system over $\mathbb{F}_2$.

$$\bigoplus_{e \ni v^*} x_e \not\equiv \deg(v^*) \pmod 2 \qquad (2.3.1)$$

$$\bigoplus_{e \ni v} x_e \equiv \deg(v) \pmod 2 \qquad v \neq v^* \qquad (2.3.2)$$

where v^* is any vertex and each parity constraint is expanded into $2^{\deg(v)-1}$ clauses. These formulas are a source for either lower bounds or trade-offs, depending on the graph.

Ordering Principle The ordering principle claims that there is a (partial) ordering of $[n]$ that has no minimal element. Variable x_{ij} stands for i being ordered before j.

$$\begin{aligned} &\overline{x_{ij}} \vee \overline{x_{ji}} && i,j \in [n] && \text{(Antisymmetry)} \\ &\overline{x_{ij}} \vee \overline{x_{jk}} \vee x_{ik} && i,j,k \in [n] && \text{(Transitivity)} \\ &\bigvee_{j \in [n]} x_{ji} && i \in [n] && \text{(Non-minimality)} \end{aligned}$$

This formula is used to separate proof systems.

Complete Tautology The complete tautology explicitly forbids all 2^n assignments.

$$\bigvee_{i \in [n]} x_i^{b_i} \qquad b \in \{0,1\}^n \tag{2.3.3}$$

Note that the number of clauses is exponential in the number of variables. A proof for the complete tautology can be translated into a proof of any other formula with the same number of variables, hence it is useful to prove upper bounds.

With these basic definitions in hand, let us go on a tour of proof systems. We only discuss the proof systems that we refer to in Chapter 3 and we skip stronger proof systems such as k-DNF resolution, bounded depth Frege, or sums of squares for which there has been recent progress. The interested reader can find more information in the surveys [170, 145, 163].

2.4 Resolution

Resolution is the simplest, most studied, and well-understood proof system. As such, we present some of the results and techniques that hold for resolution and would also be interesting to know for other proof systems.

Lines in resolution are disjunctive clauses and there is one inference rule, the resolution rule.

Resolution

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D} \tag{2.4.1}$$

This is a sound rule as we argued in the introduction. To prove completeness, let us present a problem in query complexity. In the *falsified clause search problem* a decision tree knows an unsatisfiable formula and it can query the values of a truth assignment in order to find a clause which that assignment falsifies.

A decision tree for the falsified clause search problem of depth d and size s induces a resolution proof of length s, space $d+1$, and clauses of size at most d—the proof starts at the leaves and to each internal node we add the result of resolving the clauses of its two children. In particular resolution is complete and every formula with n variables has a proof in length at most 2^n, space $n+1$, and total space n^2 simultaneously.

We have matching lower bounds for all of these measures up to constant factors in the number of variables. The first superpolynomial lower bound on length was on the pigeonhole principle [1], later improved to exponential [98], and the first lower bound of the form $2^{\Omega(n)}$ was on Tseitin formulas [177]. The first linear lower bound on space is by [77], and the first quadratic lower bound on total space for a formula of polynomial size is by [41].

There are a few techniques to prove resolution lower bounds, but the most convenient is to go through an auxiliary complexity measure: the *width* of a proof, which is the maximum size of a clause in the proof.

All of these lower bounds can be obtained from lower bounds on width. First of all, we can translate width lower bounds into length lower bounds.

Theorem 2.4.1 ([29]). *If a k-CNF has a resolution proof in length L, then it has a proof in width* $O(k+\sqrt{n \log L})$*.*

Or, in other words, the length of a proof must be at least $2^{\Omega((w-k)^2/n)}$. Theorem 2.4.1 is tight in the sense that it cannot say anything for formulas of width less than $\sqrt{n}$, as the ordering principle has proofs of polynomial length but requires width $\Omega(\sqrt{n})$ [44].

Furthermore, just by counting we see that a formula that can be proved in width w can also be proved in length $2^w\binom{n}{\leq w} = O(n^w)$, as there are only so many different clauses of width w. This observation is also tight, as there is a family of formulas inspired on the pigeonhole principle that have proofs of width w but require length $n^{\Omega(w)}$ [15].

A result similar to Theorem 2.4.1 allows us to prove space lower bounds from width lower bounds.

Theorem 2.4.2 ([12]). *If a k-CNF has a resolution proof in space s, then it has a proof in width* $O(k+s)$*.*

The converse of Theorem 2.4.2 is not true, as there is a family of pebbling formulas that have proofs of constant width but require space $\Omega(n/\log n)$ [27].

Finally we have the counterpart to Theorem 2.4.2 for total space.

Theorem 2.4.3 ([38]). *If a k-CNF has a proof in total space s, then it has a proof in width* $O(\sqrt{k+s})$*.*

To sum up, width does not tell us everything about the complexity of a formula, but it can be very informative.

Like in pebble games, there are also length-space trade-offs in resolution. That is, it can be the case that a proof has a proof in small length and another proof in small space, but not both at the same time.

Theorem 2.4.4 ([28]). *There is a family of formulas that have*

- *a resolution proof in length* $\mathrm{O}(n)$ *and*
- *a proof in space* $\mathrm{O}(n^{1/11})$*, but*
- *every proof with space less than* $n^{2/11}$ *requires length* $\exp(n^{\Omega(1)})$.

There are even formulas where, in order to have proofs of polynomial length, space needs to be larger than linear, surpassing even the worst case upper bound on space.

Theorem 2.4.5 ([19]). *There is a family of formulas that have*

- *a resolution proof in length* $n^{\mathrm{O}(\log n)}$ *and*
- *a proof in space* n*, but*
- *every proof with space less than* $n^{\log n/18}$ *requires length* $n^{\Omega(\log n \log\log n/\log\log\log n)}$.

Even though we will not discuss them at all, it is also worth mentioning that there are also trade-offs between length and width [175, 161] and between space and width [24, 33].

2.5 Polynomial Calculus

In polynomial calculus over a field $\mathbb{F}$, lines are multilinear polynomials over the ring $\mathbb{F}[x_1,\ldots,x_n,\overline{x_1},\ldots,\overline{x_n}]$. We add axioms $\{x_i^2 = x_i\}$ that enforce any roots to be over $\{0,1\}^n$, and even though variables $\overline{x_i}$ are formally independent, we add axioms $\{x_i + \overline{x_i} = 1\}$ so that we can interpret them as the negation of x_i.

Observe that a clause $\bigvee x_i^{b_i}$ is falsified exactly by the assignment $x_i \mapsto 1 - b_i$, and a monomial $\prod x_i^{b_i}$ (where we keep using the notation $x_i^0 = \overline{x_i}$) is set to 1 exactly by the assignment $x_i \mapsto b_i$. Hence there is a bijection between the set of assignments satisfying a CNF $F = \bigwedge \bigvee x_i^{b_i}$ and the roots of the set of polynomials $\left\{\prod x_i^{b_i}\right\}$.

With this bijection in mind, we can download a clause $\bigvee x_i^{b_i}$ as $\prod x_i^{b_i}$; for instance if a CNF contains the clause $x \vee y \vee \overline{z}$, then we download the monomial $xy\overline{z}$. The inference rules are sound as long as they do not eliminate roots—equivalently, the consequences stay within the ideal generated by the premises, hence we add rules to construct an ideal.

Linear Combination

$$\frac{p \qquad q}{\alpha p + \beta q} \tag{2.5.1}$$

where $\alpha, \beta \in \mathbb{F}$.

Product

$$\frac{p}{mp} \tag{2.5.2}$$

where m is any monomial.

Quotient Axioms

$$\frac{}{x+\overline{x}-1} \qquad \frac{}{x^2-x} \tag{2.5.3}$$

where x is a variable.

These rules are enough to simulate resolution. Indeed, from the monomials px and $q\overline{x}$ we obtain the monomial pq.

$$\cfrac{\cfrac{\cfrac{px}{pqx} \qquad \cfrac{q\overline{x}}{pq\overline{x}}}{pqx+pq\overline{x}} \qquad \cfrac{\cfrac{}{x+\overline{x}-1}}{-pqx-pq\overline{x}+pq}}{pq} \tag{2.5.4}$$

We note that we gave the definition of polynomial calculus of [2], also known as polynomial calculus augmented with resolution, to distinguish it from the original version of polynomial calculus of [62]. Both proof systems are equivalent with respect to degree, but the original version needs exponential space to express a large disjunction of negative literals, making it incomparable with resolution. Since in practice one would represent a polynomial with a zero-suppressed decision diagram (ZDD) [47], which does not suffer from this blow-up, this seems to be an artificial restriction, and hence we use the definition of [2].

It is always possible to prove a formula in polynomial calculus using only constant line space (the complete tautology can be proven by adding up all monomials), hence we cannot say anything interesting about it. But such a proof would use exponentially large polynomials, which suggests measuring the *monomial space* as the maximum number of monomials in a configuration.

If line space is too optimistic a measure, then monomial space may be too pessimistic, since we could avoid storing all monomials by using an appropriate data structure. However, while there are indeed proof systems that reason directly with these data structures [14], their inference rules are too strong in comparison to polynomial calculus, so they would not capture polynomial calculus space either.

Observe that we can simulate resolution with at most a constant multiplicative overhead in length and size and at most a constant additive overhead in monomial space with respect to clause space, so for a given formula the size and space in polynomial calculus are at most as large as in resolution.

Polynomial calculus size can be much smaller than resolution size. Some examples are the *onto functional* pigeonhole principle, which has polynomial size polynomial calculus proofs [164] but requires exponentially large resolution proofs [98], and Tseitin formulas, which have polynomial size proofs over $\mathbb{F}_2$ but require exponentially large resolution proofs[177]. Nevertheless there are exponential size lower bounds, the first of which were for the pigeonhole principle [160].

There are also linear space lower bounds for (a version of) the pigeonhole principle [79]. Annoyingly, though, we do not know of any separation between resolution space and monomial space that is better than a constant factor.

About total space, we only know of a $\Omega(n^2)$ lower bound for complete tautologies [2]—where total space is referred to as variable space—but no superlinear lower bound is known for a formula of polynomial size.

The equivalent of width for polynomial calculus is the *degree* of the largest polynomial in the proof, for we also have a relation between degree and size similar to Theorem 2.4.1.

Theorem 2.5.1 ([111]). *If a k-CNF has a polynomial calculus proof in size L, then it has a proof in degree* $\mathrm{O}(k + \sqrt{n \log L})$.

Theorem 2.5.1 is tight as well, and the ordering principle provides an example here too, since it requires degree $\Omega(\sqrt{n})$ [86] and the resolution length upper bound is enough for a polynomial size upper bound. Even though it is not immediate, a formula that can be proved in degree d can also be proved in size $\mathrm{O}(n^d)$ [62], and there are formulas for which this is tight [15].

There are also length-space trade-offs in polynomial calculus; in fact the equivalent to Theorems 2.4.4 and 2.4.5 also holds.

Theorem 2.5.2 ([23]). *There is a family of formulas that have*

- *a polynomial calculus proof in size* $\mathrm{O}(n)$ *and*
- *a proof in monomial space* $\mathrm{O}(n^{1/11})$*, but*
- *every proof with monomial space less than* $n^{2/11}$ *requires size* $\exp(n^{\Omega(1)})$.

Theorem 2.5.3 ([23]). *There is a family of formulas that have*

- *a polynomial calculus proof in size* $n^{\mathrm{O}(\log n)}$ *and*
- *a proof in monomial space n, but*
- *every proof with monomial space less than* $n^{\log n/18}$ *requires size* $n^{\Omega(\log n \log\log n/\log\log\log n)}$.

If space is so similar in resolution and polynomial calculus, and we have a degree measure that plays the role of width, is there also a relation between degree and space? Recall that in resolution large width implies large space (Theorem 2.4.2) but the converse is not true, so we can ask whether there are analogues of these results. We give a partial answer in Paper A, where we prove that under certain conditions large degree does imply large space, and we show that the converse is not true either, that is large space does not imply large degree.

2.6 Cutting Planes

In cutting planes lines are linear inequalities with integer coefficients and variables ranging over $\{0, 1\}$.

For convenience we work only with inequalities in the normal form $\mathbf{c}_+^\top \mathbf{x} + \mathbf{c}_-^\top(\mathbf{1} - \mathbf{x}) \geq a$, where $\mathbf{c}_+$ and $\mathbf{c}_-$ are vectors in $\mathbb{N}^n$ with nonnegative coefficients and disjoint supports. We define $\mathbf{c} = (\mathbf{c}_+, \mathbf{c}_-) \in \mathbb{N}^{2n}$ and $\hat{\mathbf{x}} = (\mathbf{x}, \mathbf{1} - \mathbf{x})$ so that we can use the shorthand $\mathbf{c}^\top \hat{\mathbf{x}}$ to represent $\mathbf{c}_+^\top \mathbf{x} + \mathbf{c}_-^\top(\mathbf{1} - \mathbf{x})$. Hence we translate a clause $\bigvee x_i^{b_i}$ into the inequality $\sum(1-b_i)+(-1)^{1-b_i}x_i \geq 1$, or equivalently the vector $\mathbf{c}_+$ has a coefficient 1 on coordinate i whenever x_i is present in the clause and zeroes elsewhere and the vector $\mathbf{c}_-$ has a coefficient 1 on coordinate i whenever $\overline{x_i}$ is present in the clause and zeroes elsewhere. For example, the clause $x \vee y \vee \overline{z}$ gets translated into $x + y + (1 - z) \geq 1$, that is to say $(1, 1, 0)(x, y, z)^\top + (0, 0, 1)(1 - x, 1 - y, 1 - z)^\top \geq 1$, so $\mathbf{c} = (1, 1, 0, 0, 0, 1)^\top$.

A set of inequalities determines a feasibility polytope in $\mathbb{R}^n$, so adding any half-space that contains the polytope is sound. In particular, the sum of two inequalities contains their intersection, so we add that as a rule. But since we are restricting solutions to lie in the hypercube $\{0, 1\}^n$, we can shrink a half space to the nearest integer point, which is the division rule, and we can add the boundaries of the cube as new inequalities.

Addition

$$\frac{\mathbf{c}^\top \hat{\mathbf{x}} \geq a \qquad \mathbf{d}^\top \hat{\mathbf{x}} \geq b}{(\mathbf{c} + \mathbf{d})^\top \hat{\mathbf{x}} \geq a + b} \tag{2.6.1}$$

where the resulting inequality might not be in normal form if the supports of $(\mathbf{c}_+ + \mathbf{d}_+)$ and $(\mathbf{c}_- + \mathbf{d}_-)$ are not disjoint, so in order to restore the normal form we replace each instance of $x + (1 - x)$ on the LHS by -1 on the RHS.

Division

$$\frac{\mathbf{c}^\top \hat{\mathbf{x}} \geq a}{\lceil \mathbf{c}/k \rceil^\top \hat{\mathbf{x}} \geq \lceil a/k \rceil} \tag{2.6.2}$$

where k is an integer.

Boundary

$$\frac{}{x \geq 0} \qquad \frac{}{1 - x \geq 0} \tag{2.6.3}$$

where x is any variable.

The first two rules are enough to simulate resolution. Indeed, from the inequalities $\mathbf{c}^\top \hat{\mathbf{x}} + x \geq 1$ and $\mathbf{d}^\top \hat{\mathbf{x}} + (1 - x) \geq 1$ we obtain $(\mathbf{c} + \mathbf{d})^\top \mathbf{x} \geq 1 + 1 - 1$, and then we ensure that all coefficients on the LHS are at most 1 by dividing by 2.

We used a definition of cutting planes that is closer to how it is implemented in practice, and as such is slightly nonstandard. The usual normal form is $\sum_{i \in [n]} c_i x_i \geq a$, where c_i and a are arbitrary integers, but in that case the division rule as we stated it behaves slightly differently and it becomes more cumbersome to simulate resolution. In any case, both definitions are equivalent except for a linear factor in length and an additive constant in space.

Cutting planes is strictly stronger than resolution since there are cutting planes proofs of the pigeonhole principle of polynomial size, and this also implies that cutting planes is stronger than polynomial calculus, but we do not know of any simulation or separation in the opposite direction. Tseitin formulas are the natural candidate for a separation, but we do not know of any technique that would yield superpolynomial lower bounds for Tseitin formulas in cutting planes. In fact, for 20 years we only knew of one exponential length lower bound for cutting planes [156] (and the equivalent [99]), and it was only very recently we saw progress with lower bounds for random formulas [105, 83].

Line space in cutting planes is, as in polynomial calculus, a very strong measure, since the complete tautology—and hence every formula—has a proof in line space 5 [87]. These constant space proofs use exponential coefficients and quadratic total space, so we can interpret that result as either that we should focus on total space, or limit the size of coefficients.

The space upper bound is less obvious than in polynomial calculus; after all a polynomial can represent any Boolean function, but a linear threshold cannot. This suggests we might still say something meaningful of line space; and indeed there are trade-offs.

Theorem 2.6.1 ([106, 95]). *There is a family of formulas that have a proof in length* $\mathrm{O}(n)$*, but every proof with line space less than* $n^{1/2}$ *requires length* $\exp(n^{\Omega(1)})$.

We do not know any non-trivial bounds on total space: the best lower bound follows from a $\Omega(n)$ lower bound on variable space that holds for every proof system, and the best upper bound follows from the $\mathrm{O}(n^2)$ resolution upper bound.

One of the reasons resolution is very well understood—and, to a similar degree, polynomial calculus—is that we have the technology to obtain length lower bounds from width/degree respectively. To really understand cutting planes we would like to have the equivalent to Theorems 2.4.1 and 2.5.1. Defining an auxiliary measure so that lower bounds for this measure would yield lower bounds for length has been an elusive task so far, with one potential candidate proposed in [162].

The only tool we have so far is feasible interpolation [124]. This technique reduces proving cutting planes lower bounds for formulas of the form $F = A(\mathbf{p},\mathbf{q}) \wedge B(\mathbf{p},\mathbf{r})$ to proving circuit lower bounds. But since we are so terrible at the latter—the best lower bound we have for an explicit function is $(3 + 1/86)n - \mathrm{o}(n)$ [82], and that was an improvement over a 30-year-old $3n - \mathrm{o}(n)$ lower bound—the technique is only useful when the $\mathbf{p}$ variables only appear in A positively or in B negatively, in which case we only need to prove monotone circuit lower bounds. This is enough for the technique to work for resolution or cutting planes with small coefficients, but for cutting planes we need one last tweak, and that is to use real circuits, which can compute with arbitrary real values.

Theorem 2.6.2 ([156]). *If* $F = A(\mathbf{p},\mathbf{q}) \wedge B(\mathbf{p},\mathbf{r})$ *has a cutting planes proof of length* L*, then there is a real circuit of size* $\mathrm{O}(L)$ *that, on input* $\mathbf{p} = \mathbf{a}$*, can distinguish between whether* $A(\mathbf{a},\mathbf{q})$ *or* $B(\mathbf{a},\mathbf{r})$ *is unsatisfiable. If furthermore the* $\mathbf{p}$ *variables only appear in* A *positively or they only appear in* B *negatively, then the circuit is monotone.*

```
while not solved do
    unit propagate
    if conflict then
        backtrack
    else
        decide variable assignment
```

Figure 2.3: DPLL algorithm

How important is it that we should allow large coefficients in cutting planes, then? Since cutting planes can simulate resolution with coefficients at most 2, that is enough to prove any formula, but possibly at a loss of efficiency. Coefficients exponentially large in the size of the proof are always enough without loss of generality [49]. However, we do not know of any formula where having superpolynomial coefficients is of any help.

Indeed, for the pigeonhole principle formulas where cutting planes is more efficient than resolution and polynomial calculus, coefficients of constant size are enough. As for lower bounds, the first lower bounds for tree-like cutting planes [110] already apply to unrestricted coefficients, and the lower bounds for cutting planes with coefficients bounded by a polynomial of [42] were extended to unrestricted coefficients in [156].

One reason we do not understand the size of coefficients is that we do not understand the relation between Boolean and real circuits well enough either. While we know that Boolean circuits may be exponentially larger than real circuits for some function in the class of slice functions [165], we do not know of any separation for explicit functions. We remark that a separation of explicit functions within the class of slice functions would imply superpolynomial lower bounds for general, nonmonotone circuits.

We do know of a few limitations of proofs with very restricted coefficients. If we limit the size of the RHS coefficient (degree of falsity in [90]) to be linear, then there are exponential lower bounds for the pigeonhole principle [103]—here it is important that we are using the normal form we defined. There is also a very weak $\Omega(\log\log\log n)$ line space lower bound for complete tautologies if coefficients are constant [87].

2.7 Conflict Driven Clause Learning

This is all so well, but how do we find proofs? Most often by looking at the intermediate steps of a SAT solver that failed to find an assignment. The first successful algorithm for SAT was that of Davis, Putnam, Logemann, and Loveland (DPLL) [70, 69], an improvement over backtracking that avoids branching when there are forced choices (see Figure 2.3).

If we look at the execution trace of the DPLL algorithm on an unsatisfiable formula we see that it produces a decision tree for the falsified clause search problem on that formula,

```
while not solved do
    unit propagate
    if conflict then
        learn
        maybe restart backtracking
        backtrack
    else
        decide variable assignment
```

Figure 2.4: CDCL algorithm

and we argued in Section 2.4 that such a decision tree is equivalent to a resolution proof.

Unfortunately, proofs that come from decision trees all have a tree-like structure, that is, their inference graph is a tree. And it turns out that tree-like resolution proofs can be exponentially longer than DAG-like proofs, as witnessed by the pebbling formulas of [43] or the ordering principle [44].

Why is that? In a tree-like proof we can only use each clause once, and if we want to use it again then we have to rederive it. This is what happens with the DPLL algorithm: we may find a conflict, then flip the value of some variable up on the backtracking stack, and find the same conflict again. Would it not be great if we could *learn* from these conflicts and avoid repeating them?

This is precisely what the Conflict Driven Clause Learning (CDCL) [136, 18] algorithm does: it adds a new clause to its state after each conflict (see Figure 2.4). The CDCL implementation of [139] was a breakthrough in that it allowed routinely solving industrial instances with millions of variables, and to this day implementations of CDCL remain the uncontested winners of the SAT competition.[1]

The execution trace of the CDCL algorithm also produces a resolution proof (see Section D.1 for details), and these proofs are DAG-like in general. Are they as powerful as general resolution proofs? If the solver is allowed to make best choices nondeterministically, then the answer is yes, since given a resolution proof in length L, there is an execution trace of a CDCL algorithm running in time poly(L) [152, 13].

Yet, unless some parameterized complexity classes collapse, no algorithm can generate resolution proofs of a formula in time polynomial in the smallest resolution proof [4]. Since all heuristics used in practice run in time polynomial in the proof they produce, this is strong evidence that, once equipped with actual heuristics, CDCL cannot polynomially simulate resolution. So far we do not know of any formula and any reasonable set of heuristics that must produce proofs superpolynomially larger than the best resolution proof.

[1]`http://www.satcompetition.org/`

While for DPLL we only need one heuristic, to decide which variable to assign next, in CDCL we also need to choose which clauses to learn and for how long to keep them in memory. Also, since the state is now larger than the backtracking stack, it makes sense to restart the backtracking while keeping previous learned clauses, and when to do that is yet another heuristic.

2.8 Communication Complexity

Besides pebbling, we use tools from yet another area of complexity: communication complexity.

In a communication problem two players, Alice and Bob, each have an input $x \in X$ and $y \in Y$ respectively, and they compute a function $f(x, y)$ over the joint input. We can imagine the players being in different planets but having a network link between them, so they can only see their own input and any messages they exchange. In this setting communication is very expensive and the round trip times can be appalling, so the players try to minimize the length and amount of transmissions. We do not assume any computational restrictions on the players, but for our discussion we impose that their messages are deterministic.

Formally, a deterministic communication protocol is a binary tree where every internal node is labelled by either a function $a_v : X \to \{0, 1\}$ or a function $b_v : Y \to \{0, 1\}$. A pair of inputs $(x, y) \in X \times Y$ defines a path by starting at the root and taking the child determined by evaluating the label of each node with either x or y. The output of the protocol on inputs (x, y) is the label of the leaf of the path defined by (x, y). The communication cost of the protocol is the depth of the tree, and the number of rounds is the maximum number of alternations between a- and b-labelled nodes.

Any function can be computed in $n = \min\{\log|X|, \log|Y|\}$ bits of communication, simply by one player revealing their input. Hence we consider a function that requires communication linear in n as hard, and a function that can be computed with an amount of communication polylogarithmic in n as easy. Among the easy functions we find parity and the clique–independent set problem—given a common graph Alice gets a clique, Bob gets an independent set, and they have to find whether their sets intersect. On the other hand, testing for equality of two strings, disjointness of two sets, or computing the inner product of two vectors are all hard problems.

Communication complexity is an exciting research topic on its own, but also useful for its many applications in other areas of theoretical computer science, among others streaming algorithms, data structures, distributed algorithms, circuit complexity, and as we shall see even proof complexity. A good overview can be found in the classical book [128] and the upcoming [157].

We may wonder whether we can do something akin to proof complexity and pebbling and use the hardness of a simple computational model to prove hardness in communication complexity. We call a result of this kind a simulation theorem, as it allows us to

Communication	Queries	Gadget	Source
Deterministic	Deterministic	Indexing (polynomial)	[96]
Deterministic	Deterministic	Inner product (logarithmic)	[59]
Deterministic	Parity	XOR (constant)	[100]
Nondeterministic	Nondeterministic	Inner product (logarithmic)	[94]
Randomized	Randomized	Indexing (polynomial)	[97]

Table 2.1: A few simulation theorems

simulate a communication protocol in the simpler computational model, in this case query complexity.

A simulation theorem takes a communication protocol for a function composed with a gadget and builds a decision tree for the original function. There are in fact quite a few flavours of simulation theorems depending on the type of communication, gadget, and decision tree—stronger communication models need stronger query models—and we list some in Table 2.1 (see [97] for a broader list). The key ingredients for a simulation theorem are in [158], using a different language, and in [96] it was stated in full generality and its potential became clear. Among other more technical results, simulation theorems have been used to prove separations in monotone circuit complexity [158], separations in proof complexity [43], and lower bounds for the clique–independent set problem [96, 93].

We build a refined version of a simulation theorem from deterministic communication to deterministic query complexity with the indexing gadget in Paper B. Since we additionally know that the Dymond–Tompa game is equivalent to query complexity, we can translate pebbling lower bounds into communication lower bounds, which in turn we use to prove lower bounds in proof complexity.

Chapter 3

Contributions

3.1 Space in Polynomial Calculus

Summary of *Towards an Understanding of Polynomial Calculus: New Separations and Lower Bounds* by Yuval Filmus, Massimo Lauria, Mladen Mikša, Jakob Nordström, and Marc Vinyals

In Paper A we explore the relation between degree and space in polynomial calculus. First we try to find out whether an analogue of the relation between resolution width and space of Theorem 2.4.2 holds. We present a partial answer with a couple of surprises in Theorem 3.1.1, but to state it we need to define the formula F substituted with XOR or $F[\oplus]$. This is the formula that we obtain by replacing every variable x in F by the exclusive or of two new variables $x_1 \oplus x_2$. Our favourite clause $x \vee y \vee \overline{z}$ gets translated into $(x_1 \oplus x_2) \vee (y_1 \oplus y_2) \vee (\overline{z_1 \oplus z_2})$, which we expand into CNF as 32 clauses

$$\begin{aligned}
&x_1 \vee x_2 \vee y_1 \vee y_2 \vee z_1 \vee \overline{z_2} \;\wedge \\
&x_1 \vee x_2 \vee y_1 \vee y_2 \vee \overline{z_1} \vee z_2 \;\wedge \\
&x_1 \vee x_2 \vee \overline{y_1} \vee \overline{y_2} \vee \overline{z_1} \vee z_2 \;\wedge \\
&\qquad\qquad\qquad\qquad\vdots \\
&\overline{x_1} \vee \overline{x_2} \vee \overline{y_1} \vee \overline{y_2} \vee \overline{z_1} \vee \overline{z_1} \;,
\end{aligned} \tag{3.1.1}$$

and in general a k-clause gets translated into 2^{2k-1} clauses.

Theorem 3.1.1. *If a k-CNF $F[\oplus]$ has a polynomial calculus proof in monomial space s, then F has a resolution proof in width* $\mathrm{O}(k+s)$.

The first surprise is a disappointing one: we do not get a space lower bound for the same formula F for which we have a degree lower bound, but for $F[\oplus]$. The second surprise is more appealing: we do not need a degree lower bound to apply the theorem, but only a width lower bound. Since degree is never larger than width, this makes the result stronger. In fact, degree can be significantly smaller than width, and this lets us separate degree from space.

Corollary 3.1.2. *There is a family of formulas that have polynomial calculus proofs of constant degree but require space $\Omega(n)$.*

The formulas for which we obtain the separation are Tseitin formulas, which we know to be easy for polynomial calculus but hard for resolution. It is not hard to see that after applying a XOR substitution we are still left with a Tseitin formula, only for a graph that now has two edges where there used to be one. Therefore it is still an easy formula for polynomial calculus, but it is now in a form where we can apply Theorem 3.1.1 and obtain a space lower bound.

Since graphs with double edges are somewhat artificial we also prove space lower bounds for plain expander graphs, except that we need the technical condition that we can partition their edges into small cycles. Some examples of graphs that satisfy this condition are grids and random graphs of degree 4—the latter can almost surely be partitioned into cycles of length $\widetilde{O}(\sqrt{n})$ plus a negligible amount of extra edges.

We prove both space lower bounds using the extensible families framework of [39]. This framework is inspired by the characterization of resolution space of [12] in terms of the Spoiler–Duplicator game, and indeed using extensible families is enough to prove all space lower bounds known to date, so one may wonder whether extensible families characterize space in polynomial calculus. We offer circumstantial evidence that they do not by proving that the functional pigeonhole principle only has extensible families of constant size, which can only yield constant space lower bounds. While for all we know the functional pigeonhole principle can be proved in constant space, this is not the case for the usual pigeonhole principle, and both versions are equally hard with respect to size—as opposed to the onto functional pigeonhole principle, which has proofs of polynomial size in polynomial calculus.

3.2 Trade-offs in Cutting Planes

Summary of *How Limited Interaction Hinders Real Communication (and What it Means for Proof and Circuit Complexity)* by SUSANNA F. DE REZENDE, JAKOB NORDSTRÖM, AND MARC VINYALS

In Paper B we prove a round-aware simulation theorem for communication complexity and use it to prove size-space trade-offs for cutting planes and a separation in monotone circuit complexity.

Unfortunately, a communication lower bound alone does not immediately work for proving strong trade-offs where we ask not only for a small upper bound on line space but for a small upper bound on total space. That is because a total space upper bound in cutting planes implies a pebbling upper bound, which in turn implies a communication upper bound.

The solution comes from revisiting the strategy for proving trade-offs. From a proof we obtain a protocol for the communication version of the falsified clause search problem, and then prove a communication lower bound. The protocol evaluates configurations of the proof in a binary search pattern until it finds a configuration that the assignment satisfies followed by a configuration that the assignment falsifies. Since inference and erasure steps are sound, the step between these configurations was an axiom download, and that axiom is falsified. We can evaluate one configuration during the binary search in one round and communication at most $s\eta$, where s is the number of lines in the configuration and η is the cost of evaluating one line. Therefore if the proof has length L and space s then the protocol has communication at most $s\eta \log L$, and furthermore there are only $\log L$ rounds of communication. So, instead of only relating the product of measures $s \log L$ with communication as in [106, 95], we can relate space with communication per round and length with number of rounds independently.

Trade-offs between rounds and communication have been well studied. The canonical problem is *pointer jumping*, where each player holds an array with n pointers into the array of the other player, they start at Alice's first position, and they have to find the position after following k pointers. Clearly they can solve the problem in k rounds by revealing the appropriate pointer with a cost of $\log n$ bits per round, but if they only have $k-1$ rounds—or even if the wrong player starts speaking—then they require $\Omega(n/k^2)$ bits of communication [141].

We do not make use of such trade-offs directly but we use a simulation theorem to lift query complexity trade-offs.

Theorem 3.2.1. *If there is a deterministic communication protocol for a search problem* $S \circ \mathrm{IND}^n = S(\mathrm{IND}(x_1, y_1), \ldots, \mathrm{IND}(x_n, y_n))$ *with communication c and r rounds, then there is a decision tree for S with* $\mathrm{O}(c/\log n)$ *queries and r rounds.*

Here $\mathrm{IND} : [m] \times \{0,1\}^m \to \{0,1\}$ is the indexing function defined as $\mathrm{IND}(x,y) = y_x$, and $m = \mathrm{poly}(n)$.

If the original formula is a pebbling formula, then the falsified clause search problem becomes the false vertex problem, which we mentioned is equivalent to the Dymond–Tompa game. We prove that decision trees with limited adaptivity are equivalent to Dymond–Tompa strategies with limited rounds, so it only remains to prove a trade-off for the Dymond–Tompa game.

Theorem 3.2.2. *There is a family of graphs of depth d and dn vertices that have*

- *a Pebbler strategy with d rounds and $2d$ pebbles and*
- *a Pebbler strategy with $2\log d$ rounds and $(n+2)\log d$ pebbles, but*
- *any Pebbler strategy with r rounds requires* $\min\{r2^{d/r}, n\}/8$ *pebbles.*

Combining Theorems 3.2.1 and 3.2.2 we already get trade-offs for the resolution and polynomial calculus proof systems, but this is not enough for cutting planes, since evaluating one line requires comparing two integers that can be as large as $n!$ in general, and this problem requires $\Omega(n \log n)$ communication.

The solution is to use a model of communication where we can compare numbers with little communication. This could be a randomized model, but we use the *real* *communication* model of [125], which was specifically designed to study cutting planes. A real communication protocol can compare arbitrary numbers at unit cost, and while real communication is strictly more powerful than deterministic, it is easier to analyze than randomized communication. In fact, a simulation theorem for the real model was already proved—with a different language—in [43] as a follow-up to [158], and we are able to adapt their ideas to prove an analogue of Theorem 3.2.1 for real communication.

Theorem 3.2.3. *If there is a real communication protocol for $S \circ \mathrm{IND}^n$ with communication c and r rounds, then there is a decision tree for S with $\mathrm{O}(c/\log n)$ queries and r rounds.*

Combining Theorems 3.2.3 and 3.2.2 yields trade-offs for cutting planes.

Theorem 3.2.4. *There is a family of formulas that have*

- *a cutting planes proof in size $\mathrm{O}(n)$ and*
- *a proof in total space $\mathrm{O}(n^{1/40})$, but*
- *any proof with line space $n^{1/20-\epsilon}$ requires length $2^{\Omega}(n^{1/40})$.*

Another application of Theorems 3.2.1 and 3.2.2 is in monotone circuit complexity. There is a depth hierarchy for monotone-NC, i.e., there are Boolean functions that can be computed by monotone circuits of fan-in 2, polynomial size, and depth $\log^i n$, but cannot be computed by monotone circuits of fan-in 2 and depth $\log^{i-1} n$ (of any size) [158]. The functions, however, can be computed with circuits of unbounded fan-in and depth 2 of quasipolynomial size. To prove a stronger monotone-AC hierarchy we revisit the reduction from communication protocols to monotone circuits of [118] and observe that depth corresponds to number of rounds while size corresponds to communication.

Theorem 3.2.5. *For every $i \in \mathbb{N}$ there is a Boolean function over n variables that can be computed by a monotone circuit of depth $\log^i n$, fan-in $n^{4/5}$, and size $\mathrm{O}(n)$, but for which for every $q \in \mathbb{N}$ every monotone circuit of depth $q \log^{i-1} n$ requires size $2^{\Omega(n^{1/11q})}$.*

3.3 Cumulative Space

Summary of *Cumulative Space in Black-White Pebbling and Resolution* by Joël Alwen, Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals

The concept of cumulative space arises from trying to give a more complete picture of space, for instance distinguishing the space usage of the two generic computations that we plotted in Figure 3.1. One uses large space during a short peak, while another uses moderately large space throughout the whole computation.

Using a traditional space measure we would say that the peak computation uses the most space, since the maximum space is larger, and we would be right in saying so because if we are buying a memory chip, then we need it to hold the maximum memory usage at any point. But what if we are trying to run many instances of the computations

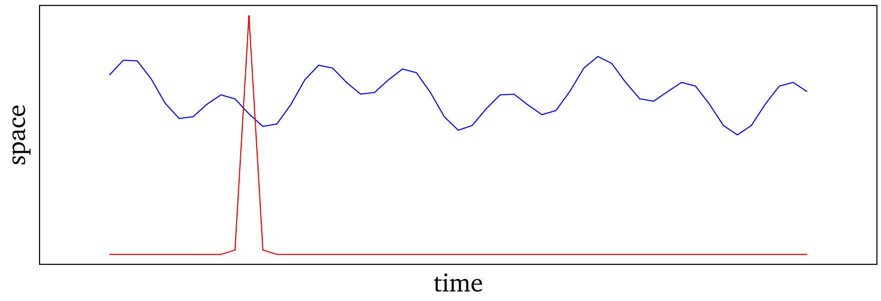


Figure 3.1: Space usage of two computations

in parallel? Then we can amortize the space usage of the peak computations by running them with a large enough delay so that their space peaks do not intersect, but we cannot do that with the barely fluctuating computations, so we would say that the latter use more space.

As its name suggests, cumulative space is the area under the time-space curve. We could have chosen some other measure such as average space, but that has the problem that we can lower it artificially by extending the computation unnecessarily. Hence we stay with cumulative space, which seems the most robust.

Far from being a fairy tale, this setup applies to cryptography, for example if we are trying to invert a hash function by evaluating it massively in parallel. This motivated the definition of cumulative space of [8] in order to obtain a more robust definition of memory-hard functions. In that case the results follow from a parallel version of the black pebble game where we lift the restriction of only changing one pebble per round. We introduce cumulative space to proof complexity in Paper C, and for that we need the black-white version of the pebble game instead. As we discuss in Section C.2, whether the game is parallel does not play such a big role for us.

We prove a few results about the cumulative space of black-white pebbling. In contrast to space, in which we can always improve on the trivial $\mathrm{O}(n)$ upper bound by a $\log n$ factor, the $\mathrm{O}(n^2)$ upper bound for cumulative space is tight.

Theorem 3.3.1. *There is a family of graphs with cumulative black-white pebbling space* $\Omega(n^2)$.

Every graph with large space also has large cumulative space, simply because if we are to place s pebbles on a graph then we need one configuration with 1 pebble, another with 2 pebbles, and so forth, for a total of $\Theta(s^2)$. The converse does not hold, as there are graphs with small space but large cumulative space.

Theorem 3.3.2. *There is a family of graphs with black-white pebbling space* $\mathrm{O}(\log n)$ *but cumulative space* $\Omega(n^2/\log n)$

We can also improve on the trade-offs of [130] for bit-reversal graphs, which are of the form $ts = \Omega(n^2/s)$, and replace ts by the cumulative space. This may not say much on first sight, but in fact it implies that the trade-offs are very robust, in that any optimal pebbling of these graphs—in the sense that it matches the $ts = \Omega(n^2/s)$ lower bound—must be using worst case space most of the time.

Theorem 3.3.3. *There is a family of graphs such that for any $s = O(\sqrt{n})$*

- *there is a black-white pebbling with space $O(s)$ and length $O(n^2/s^2)$; in particular*
  - *there is a black-white pebbling with space $O(1)$, and*
  - *there is a black-white pebbling with length $O(n)$; but*
- *every black-white pebbling in space $O(s)$ needs cumulative space $\Omega(n^2/s)$; in particular*
  - *every black-white pebbling in space $O(s)$ and length $O(n^2/s^2)$ needs $\Omega(n^2/s^2)$ configurations with space $\Omega(s)$.*

All of these results also have their counterparts in resolution, where we replace time by length, space by line space, and cumulative space by cumulative line space, and they follow directly by extending a lemma in [28] to cumulative space.

Lemma 3.3.4. *If there is a resolution proof of $Peb_G[\oplus]$ in length L, space s, and cumulative space c, then there is a black-white pebbling of G in time L, space s, and cumulative space c.*

3.4 Trade-offs in CDCL

Summary of *Trade-offs Between Time and Memory in a Tighter Model of CDCL SAT Solvers* by Jan Elffers, Jan Johannsen, Massimo Lauria, Thomas Magnard, Jakob Nordström, and Marc Vinyals

We had left our review of CDCL in Section 2.7 mentioning that CDCL can polynomially simulate resolution. Unfortunately the simulation does not tell us anything about space. For it to work we need the CDCL algorithm not to forget any clause, which is far from what an actual solver would do, hence we end up with worst case space.

In Paper D we define a proof system that faithfully models CDCL, including space, and where we can plug in heuristics to form versions of CDCL in between an ideal nondeterministic proof system and a completely specified algorithm.

Our proofs are formally execution traces that can be verified by a *partially specified* CDCL algorithm that takes a proof as input in addition to a formula. The verifier then uses the proof to make choices when a heuristic is not specified, and verifies that the proof makes the same choices as the heuristic when it is specified. For example, a verifier with an undefined decision heuristic and fixed learning heuristic will assign values to variables according to the proof and make sure that it learns the same clauses as the proof claims. Note that a verifier with a different learning heuristic may reject the proof instead. In any case, all CDCL proofs have a unique translation into a resolution proof, so we identify a proof with its translation and say that CDCL proofs are a subset of resolution proofs. The full definition is in Section D.2.

In order to study space in this proof system we use the same formulas that we use for resolution: pebbling formulas. All lower bounds follow immediately from the fact that CDCL proofs are a subset of resolution proofs, so we only need to find upper bounds. We prove that CDCL can—akin to resolution—prove a pebbling formula substituted with XOR by simulating a black pebbling on the graph. Hence all length-space trade-offs of [28] also hold for our proof system.

The kind of proofs that follow from a black pebbling crucially use restarts. Let us sketch why. We simulate a pebble placement on vertex x by deriving the clauses that represent vertex x being true, $x_1 \lor x_2$ and $\overline{x_1} \lor \overline{x_2}$, and a pebble erasure by erasing the corresponding clauses. Say that we learn the first clause $x_1 \lor x_2$: that is because setting both $x_1 \mapsto \text{FALSE}$ and $x_2 \mapsto \text{FALSE}$ leads to a conflict. To resolve the conflict we flip the value of either value, say $x_2 \mapsto \text{TRUE}$. Now vertex x is set to true, and it will not stop being so until we find a conflict higher up in the backtracking stack, so we cannot learn the second clause $\overline{x_1} \lor \overline{x_2}$. But if we restart the backtracking then the vertex becomes unassigned and we can set $x_1 \mapsto \text{TRUE}$ and $x_2 \mapsto \text{TRUE}$ and find a conflict quickly.

Hence we take the opportunity to study the importance of restarts by studying the ability to prove pebbling formulas in a model of CDCL that disallows restarts—this is, fixing a restart heuristic that never restarts. Perhaps surprisingly we are still able to prove the same kind of trade-offs, but we need to do significantly more work.

First we introduce a way to represent pebbling strategies as a binary tree so that we can operate on a pebbling recursively, and we build strategies of this form for the graphs that we want to provide proofs for. Then we modify the pebbling to make it easier to simulate, adding some padding between consecutive vertices in order to avoid undesired unit propagations. This step works for any pebbling, but at the cost of changing the graph, therefore the formulas that we prove trade-offs for differ slightly from those in [28]. The last step is to actually generate a CDCL trace from the pebbling, which is a very laborious but not conceptually hard task.

Finally, it is worth mentioning that we implemented software[1] to automate translating a black pebbling strategy to a CDCL proof without restarts. After all, and with a wink towards Donald Knuth, it can be more reassuring to build such complicated proofs rather than prove them correct. The software uses an earlier, more convoluted translation that avoids modifying the graph, and for which we did not prove correctness, so while we have proofs that we can touch they are not the same as we described. It remains for us to update the software to produce the proofs we present in this paper.

[1]Available at `https://github.com/marcvinyals/textbooksat/`

3.5 Inapproximability of Pebbling

Summary of *Hardness of Approximation in PSPACE and Separation Results for Pebble Games* by Siu Man Chan, Massimo Lauria, Jakob Nordström, and Marc Vinyals

The last paper, Paper E, takes a meta view of pebbling. Pebbling is a great tool for studying space complexity, but what is the space complexity of pebbling? More precisely, we want to decide whether a graph G can be pebbled using at most s pebbles. We can solve that within PSPACE for any pebble game (say by using a logarithmic space algorithm to determine s-t connectivity in the graph of all configurations of size at most s).

For black pebbling the problem is also PSPACE-hard [88]. A version of the black-white and reversible pebbling games that is played on graphs of unbounded fan-in is also PSPACE-hard [102, 55]. Unfortunately the techniques do not carry over, as these games have substantially different properties—to begin with, there is no $\mathrm{O}(n/\log n)$ upper bound. We can prove that reversible pebbling in graphs of fan-in 2 is PSPACE-hard, but we need more complex constructions.

Theorem 3.5.1. *Deciding the reversible pebbling space is* PSPACE*-complete.*

All proofs of hardness of pebble games are similar at a high level to the original in [88] and go through a direct and elaborate reduction from the Quantified Boolean Formula (QBF) satisfiability problem. We craft gadgets that represent the parts of a QBF, and connect them in a way such that the space of the resulting graph is s if the formula is satisfiable and $s+1$ if it is not.

It is somewhat surprising that we can reduce what is an essentially two-player game to a one-player game, but perhaps less so in the case of reversible pebbling, which we know to be equivalent to the Dymond–Tompa game. We do not use this fact in the proofs, and we leave as an open problem to find a simplified proof using the Dymond–Tompa game instead.

We go one step further and prove that pebbling space is not only hard to compute but to approximate within additive constants—again, even for graphs of fan-in 2.

Theorem 3.5.2. *Approximating the black pebbling space within any additive constant is* PSPACE*-hard.*

Theorem 3.5.3. *Approximating the reversible pebbling space within any additive constant is* PSPACE*-hard.*

These results are not as impressive as if we could prove hardness of approximation within a multiplicative constant, but not much is known about hardness of approximation in PSPACE—there are some multiplicative hardness of approximation results for optimization versions of PSPACE-hard problems [63, 107]—and nothing at all for pebbling.

Instead we prove the results with two product constructions, one for each game, that from a graph of order n and space s produce a graph of order $O(n^2)$ and space $2s + p$, where p is a constant that depends on the flavour of the game. One only needs to apply the product a constant number of times to a graph whose pebbling number is PSPACE-hard to compute exactly to obtain Theorems 3.5.2 and 3.5.3.

But most of this work would have been moot if black and reversible pebbling turned out to be the same game. We already knew that they are not quite the same, as paths have constant black space but logarithmic reversible space, but in all examples we had, black and reversible pebbling are within an additive logarithmic term. We construct some graphs, a wide version of paths that we call *roads*, where we can prove a multiplicative logarithmic separation. There is still a gap between the $s^2 \log n$ simulation of [126] and our $s \log n$ separation.

It is worth mentioning that these results have direct applications to proof complexity. Since the Dymond–Tompa cost is equivalent to decision tree depth over pebbling formulas, which is in turn equivalent to resolution depth, it follows that it is PSPACE-hard to approximate resolution depth within an additive constant. Additionally, since black pebbling space is an upper bound on variable space, the separation between black and reversible pebbling provides a separation between variable space and depth in resolution.

Chapter 4

Conclusion

In this thesis we made some progress towards understanding space in proof complexity. First we studied proof systems stronger than resolution, namely polynomial calculus and cutting planes. For polynomial calculus we proved a relation between resolution width and polynomial calculus space, and a separation between degree and space. However it still remains open to find—if it exists—a relation between degree and space that works for all formulas, maybe via a combinatorial characterization of space, and to determine whether resolution space and polynomial calculus space can be separated.

We showed strong length-space trade-offs for cutting planes, in the sense that upper bounds are for a strict measure of space, while lower bounds are for a more lenient measure. Unfortunately, in the process of applying a substitution our formulas suffer from a polynomial blow-up, which leaves us with polynomial losses in the results. Finding a better substitution for which the blow-up is only constant is an open problem not only in proof complexity but in communication complexity too.

Then we sailed into uncharted waters and introduced two new measures: one for cumulative space in resolution, and another for space in a proof system that accurately depicts the CDCL algorithm. We studied cumulative space for black-white pebbling and showed that it is possible to say things that do not immediately follow from maximal space, and then we translated these results into resolution. As natural as cumulative space is, it has not received too much attention, so it would be interesting to define it for other proof systems and models of computation.

Regarding space in CDCL we showed a rich landscape of trade-offs comparable to those in resolution. We used the opportunity to study the power of restarts in and we found out that they are not needed to prove trade-offs. There is plenty of work to do in this direction. For starters, we should verify that these results are relevant in practice and try to observe them by running experiments on SAT solver implementations—there is some preliminary work in [75]. We could also follow up on the study of restarts, perhaps aiming for a separation with different formulas. And of course we could try to extend the simulation of resolution by CDCL to work with space too.

All of these results share a common pattern of *hardness amplification*. We prove lower bounds in a simple model of computation (resolution width, query complexity, black-white pebbling space), then we modify our problem a bit (composing with XOR or with indexing), and then we show how the modified problem inherits the lower bounds in a harder model of computation (polynomial calculus space, communication complexity, resolution space). This is a very powerful paradigm and we are likely to find more applications of it.

Finally we studied the complexity of the reversible pebble game itself, as well as how hard it is to approximate the black and reversible space. We showed some connections to proof complexity but, more importantly, a good understanding of the Dymond–Tompa game helped with proving round-space trade-offs.

Part II

Included Papers

Paper A

Towards an Understanding of Polynomial Calculus: New Separations and Lower Bounds

YUVAL FILMUS, MASSIMO LAURIA, MLADEN MIKŠA, JAKOB NORDSTRÖM, AND MARC VINYALS

Full length version of the article published in Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP '13), July 2013, pp. 437–448.

Abstract

During the last decade, an active line of research in proof complexity has been into the space complexity of proofs and how space is related to other measures. By now these aspects of resolution are fairly well understood, but many open problems remain for the related but stronger polynomial calculus (PC/PCR) proof system. For instance, the space complexity of many standard "benchmark formulas" is still open, as well as the relation of space to size and degree in PC/PCR.

We prove that if a formula requires large resolution width, then making XOR substitution yields a formula requiring large PCR space, providing some circumstantial evidence that degree might be a lower bound for space. More importantly, this immediately yields formulas that are very hard for space but very easy for size, exhibiting a size-space separation similar to what is known for resolution. Using related ideas, we show that if a graph has good expansion and in addition its edge set can be partitioned into short cycles, then the Tseitin formula over this graph requires large PCR space. In particular, Tseitin formulas over random 4-regular graphs almost surely require space at least $\Omega(\sqrt{n})$.

Our proofs use techniques recently introduced in [Bonacina-Galesi '13]. Our final contribution, however, is to show that these techniques provably cannot yield non-constant space lower bounds for the functional pigeonhole principle, delineating the limitations of this framework and suggesting that we are still far from characterizing PC/PCR space.

A.1 Introduction

Proof complexity studies how hard it is to provide succinct certificates for tautological formulas in propositional logic—i.e., proofs that formulas always evaluate to true under any truth value assignment, where these proofs are verifiable in time polynomial in their size. It is widely believed that there is no proof system where such efficiently verifiable proofs can always be found of size at most polynomial in the size of the formulas they prove. Showing this would establish $\mathsf{NP} \neq \mathsf{coNP}$, and hence $\mathsf{P} \neq \mathsf{NP}$, and the study of proof complexity was initiated by Cook and Reckhow [66] as an approach towards this (still very distant) goal.

A second prominent motivation for proof complexity is the connection to applied SAT solving. By a standard transformation, any propositional logic formula F can be transformed to another formula F' in conjunctive normal form (CNF) such that F' has the same size up to constant factors and is unsatisfiable if and only if F is a tautology. Any algorithm for solving SAT defines a proof system in the sense that the execution trace of the algorithm constitutes a polynomial-time verifiable witness of unsatisfiability (such a witness is often referred to as a *refutation* rather than a *proof*, and we will use the two terms interchangeably in this paper). In the other direction, most modern SAT solvers can in fact be seen to search for proofs in systems studied in proof complexity, and upper and lower bounds for these proof systems hence give information about the potential and limitations of such SAT solvers.

In addition to running time, a major concern in SAT solving is memory consumption. In proof complexity, these two resources are modelled by *proof size/length* and *proof space*. It is thus interesting to understand these complexity measures and how they are related to each other, and such a study reveals intriguing connections that are also of intrinsic interest to proof complexity. In this context, it is natural to focus on proof systems at comparatively low levels in the proof complexity hierarchy that are, or could plausibly be, used as a basis for SAT solvers. Such proof systems include resolution and polynomial calculus. This paper takes as its starting point the former system but focuses on the latter.

A.1.1 Previous Work

The *resolution* proof system was introduced in [35], and is at the foundation of state-of-the-art SAT solvers based on so-called conflict-driven clause learning (CDCL) [18, 136].

In resolution, one derives new disjunctive clauses from the clauses of the original CNF formula until contradiction is reached. One of the early breakthroughs in proof complexity was the (sub)exponential lower bound on proof length (measured as the number of clauses in a proof) obtained by Haken [98]. Truly exponential lower bounds—i.e., bounds $\exp(\Omega(n))$ in the size n of the formula—were later established in [61, 177] and other papers.

Ben-Sasson and Wigderson [29] identified *width* as a crucial resource, where the width is the size of a largest clause in a resolution proof. They proved that strong lower bounds on width imply strong lower bounds on length, and used this to rederive essentially all known length lower bounds in terms of width.

The study of space in resolution was initiated by Esteban and Torán [77], measuring the space of a proof (informally) as the maximum number of clauses needed to be kept in memory during proof verification. Alekhnovich et al. [2] later extended the concept of space to a more general setting, including other proof systems. The (clause) space measure can be shown to be at most linear in the formula size, and matching lower bounds were proven in [2, 25, 77].

Atserias and Dalmau [12] proved that space is in fact lower-bounded by width, which allowed to rederive all hitherto known space lower bounds as corollaries of width lower bounds. A strong separation of the two measures was obtained in [27], exhibiting a formula family with constant width complexity but almost linear space complexity. Also, dramatic space-width trade-offs have been shown in [24], with formulas refutable in constant width and constant space where optimizing one of the measures causes essentially worst-case behaviour of the other.

Regarding the connections between length and space, it follows from [12] that formulas of low space complexity also have short proofs. For the subsystem of *tree-like resolution*, where each line in the proof can only be used once, [77] showed that length upper bounds also imply space upper bounds, but for general resolution [27] established that this is false in the strongest possible sense. Strong trade-offs between length and space were proven in [28, 19].

This paper focuses on the more powerful *polynomial calculus (PC)*[1] proof system introduced by Clegg et al. [62], which is not at all as well understood. In a PC proof, clauses are interpreted as multilinear polynomials (expanded out to sums of monomials), and one derives contradiction by showing that these polynomials have no common root. Intriguingly, while proof complexity-theoretic results seem to hold out the promise that SAT solvers based on PC could be orders of magnitude faster than CDCL, such algebraic solvers have so far failed to be truly competitive.

Proof size[2] in PC is measured as the total number of monomials in a proof and the

[1]Strictly speaking, to get a stronger proof system than resolution we need to look at the generalization *PCR* as defined in [2], but for simplicity we will be somewhat sloppy in this introduction in distinguishing between PC and PCR.

[2]The *length* of a proof is the number of lines, whereas *size* also considers the size of lines. In resolution the two measures are essentially equivalent. In PC size and length can be very different, however, and so size

analogue of resolution space is the number of monomials needed in memory during proof verification. Clause width in resolution translates into polynomial degree in PC. While length, space and width in resolution are fairly well understood as surveyed above, our understanding of the corresponding complexity measures in PC is much more limited.

Impagliazzo et al. [111] showed that strong degree lower bounds imply strong size lower bounds. This is a parallel to the length-width relation in [29], and in fact this latter paper can be seen as a translation of the bound in [111] from PC to resolution. This size-degree relation has been used to prove exponential lower bounds on size in a number of papers, with [3] perhaps providing the most general setting.

The first lower bounds on space were reported in [2], but only sublinear bounds and only for formulas of unbounded width. The first space lower bounds for k-CNF formulas were presented in [79], and asymptotically optimal (linear) lower bounds were finally proven by Bonacina and Galesi [39]. However, there are several formula families with high resolution space complexity for which the PC space complexity has remained unknown, e.g., Tseitin formulas (encoding that the sum of all vertex degrees in an undirected graph must be even), ordering principle formulas, and functional pigeonhole principle (FPHP) formulas.

Regarding the relation between space and degree, it is open whether degree is a lower bound for space (which would be the analogue of what holds in resolution) and also it has been unknown whether the two measures can be separated in the sense that there are formulas of low degree complexity requiring high space. However, [23] recently proved a space-degree trade-off analogous to the resolution space-width trade-off in [24] (in fact for the very same formulas). This could be interpreted as indicating that there should be a space-degree separation analogous to the space-width separation in resolution, and the authors of [39] suggest that their techniques might be a step towards understanding degree and proving that degree lower-bounds space, similar to how this was done for resolution width in [12].

As to size versus space in PC, essentially nothing has been known. It is open whether small space complexity implies small size complexity and/or the other way around. Some size-space trade-offs were recently reported in [106, 23], but these trade-offs are weaker than the corresponding results for resolution.

A.1.2 Our Results

We study the relation of size, space, and degree in PC (and the stronger system PCR) and present a number of new results as briefly described below.

1. We prove that if the resolution width of refuting a CNF formula F is w, then by substituting each variable by an exclusive or of two new variables and expanding out we get a new CNF formula $F[\oplus]$ requiring PCR space $\Omega(w)$. In one sense, this is stronger than claiming that degree is a lower bound for space, since high width

is the right measure to study.

complexity is a necessary but not sufficient condition for high degree complexity. In another sense, however, this is (much) weaker in that XOR substitution can amplify the hardness of formulas substantially. Nevertheless, to the best of our knowledge this is the first result making any connection between width/degree and space for polynomial calculus.

2. More importantly, this result yields essentially optimal separations between length and degree on the one hand and space on the other. Namely, taking expander graphs and making double copies of all edges, we show that Tseitin formulas over such graphs have proofs in size $O(n \log n)$ and degree $O(1)$ in PC but require space $\Theta(n)$ in PCR. (Furthermore, since these small-size proofs are tree-like, this shows that there is no tight correlation between size and space in tree-like PC/PCR in contrast to resolution.)

3. Using related ideas, we also prove strong PCR space lower bounds for Tseitin formulas over (simple or multi-)graphs where the edge set can be partitioned into small cycles. (The two copies of every edge in the multi-graph above form such cycles, but this works in greater generality.) In particular, for Tseitin formulas over random d-regular graphs for $d \geq 4$ we establish that an $\Omega(\sqrt{n})$ PCR space lower bound holds asymptotically almost surely.

4. On the negative side, we show that the techniques in [39] cannot prove any non-constant PCR space lower bounds for functional pigeonhole principle (FPHP) formulas. That is, although these formulas require high degree and it seems plausible that they are hard also with respect to space, the machinery developed in [39] provably cannot establish such lower bounds. Unfortunately, this seems to indicate that we are further from characterizing degree in PC/PCR than previously hoped.

A.1.3 Organization of This Paper

The rest of this paper is organized as follows. We briefly review preliminaries in Section A.2. Section A.3 presents a overview of our results and provides some proof sketches outlining the main technical ideas that go into the proofs.

In Section A.4, we prove that resolution width lower bounds plus substitutions with XOR or other suitable Boolean functions yields PCR space lower bounds. We use this in Section A.5 to separate size and degree from space in PC and PCR. In Section A.6, we show PCR space lower bounds for Tseitin formulas over graphs with edge sets decomposable into partitions of small cycles. The proof that random d-regular graphs for $d \geq 4$ (almost) decompose into cycles of length $O(\sqrt{n})$ is given in Section A.7. The fact that PCR space lower bounds cannot be obtained for the functional pigeonhole principle formulas with current techniques is proven in Section A.8, and in the same section we show that a larger class of formulas containing FPHP formulas have essentially the same space complexity for PC and PCR (so that when proving lower bounds, one can without loss of generality ignore the complementary formal variables for negative literals in PCR and focus only on PC).

We make some concluding remarks and discuss some of the (many) open questions remaining in Section A.9. For completeness, in Appendix A.10 we provide a full description of our version of the techniques in [39] and provide proofs that the same claims still hold in this slightly different setting.

A.2 Preliminaries

A *literal* over a Boolean variable x is either the variable x itself (a *positive literal*) or its negation $\neg x$ or $\overline{x}$ (a *negative literal*). It will also be convenient to use the alternative notation $x^0 = x$, $x^1 = \overline{x}$, where we identify 0 with true and 1 with false[3] (so that x^b is true if $x = b$). A *clause* $C = a_1 \vee \cdots \vee a_k$ is a disjunction of literals. We denote the empty clause by $\bot$. A clause containing at most k literals is called a *k-clause*. A *CNF formula* $F = C_1 \wedge \cdots \wedge C_m$ is a conjunction of clauses. A *k-CNF formula* is a CNF formula consisting of k-clauses. We think of clauses and CNF formulas as sets so that order is irrelevant and there is no repetitions.

Let $\mathbb{F}$ be a field and consider the polynomial ring $\mathbb{F}[x, \overline{x}, y, \overline{y}, \ldots]$ (where x and $\overline{x}$ are viewed as distinct formal variables). We employ the standard notation $[n] = \{1, \ldots, n\}$.

Definition A.2.1 (Polynomial calculus resolution (PCR)). A *PCR configuration* $\mathbb{P}$ is a set of polynomials in $\mathbb{F}[x, \overline{x}, y, \overline{y}, \ldots]$. A *PCR refutation* of a CNF formula F is a sequence of configurations $\{\mathbb{P}_0, \ldots, \mathbb{P}_\tau\}$ such that $\mathbb{P}_0 = \emptyset$, $1 \in \mathbb{P}_\tau$, and for $t \in [\tau]$ we obtain $\mathbb{P}_t$ from $\mathbb{P}_{t-1}$ by one of the following steps:

Axiom download $\mathbb{P}_t = \mathbb{P}_{t-1} \cup \{p\}$, where p is either a monomial $m = \prod_i x_i^b$ encoding a clause $C = \bigvee_i x_i^b \in F$, or a *Boolean axiom* $x^2 - x$ or *complementarity axiom* $x + \overline{x} - 1$ for any variable x (or $\overline{x}$).

Inference $\mathbb{P}_t = \mathbb{P}_{t-1} \cup \{p\}$, where p is inferred by *linear combination* $\frac{q \quad r}{\alpha q + \beta r}$ or *multiplication* $\frac{q}{xq}$ from polynomials $q, r \in \mathbb{P}_{t-1}$ for $\alpha, \beta \in \mathbb{F}$ and x a variable.

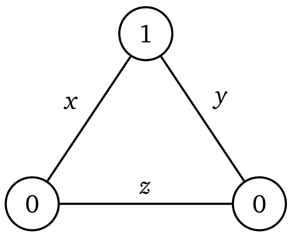
Erasure $\mathbb{P}_t = \mathbb{P}_{t-1} \setminus \{p\}$, where p is a polynomial in $\mathbb{P}_{t-1}$.

If we drop complementarity axioms and encode each negative literal $\overline{x}$ as the polynomial $(1 - x)$, the proof system is called *polynomial calculus (PC)*.

The *size* $S(\pi)$ of a PC/PCR refutation π is the number of monomials (counted with repetitions) in all downloaded or derived polynomials in π, the *(monomial) space* $Sp(\pi)$ is the maximal number of monomials (counted with repetitions)[4] in any configuration in

[3]Note that this notational convention is the opposite of what is found in many other papers, but as we will see shortly it is the natural choice in the context of polynomial calculus.

[4]We note that in [2], space was defined *without* counting repetitions of monomials. All our lower bounds hold in this more stringent setting as well.



(a) Labelled triangle graph.

$$\begin{aligned} &(x \vee y) \\ \wedge\ &(\overline{x} \vee \overline{y}) \\ \wedge\ &(x \vee \overline{z}) \\ \wedge\ &(\overline{x} \vee z) \\ \wedge\ &(y \vee \overline{z}) \\ \wedge\ &(\overline{y} \vee z) \end{aligned}$$

(b) Corresponding Tseitin formula.

Figure A.1: Example Tseitin formula.

π, and the *degree* $Deg(\pi)$ is the maximal degree of any monomial appearing in π. Taking the minimum over all PCR refutations of a formula F, we define the size $S_{\mathcal{PCR}}(F \vdash \bot)$, space $Sp_{\mathcal{PCR}}(F \vdash \bot)$, and degree $Deg_{\mathcal{PCR}}(F \vdash \bot)$ of refuting F in PCR (and analogously for PC).

We can also define *resolution* in this framework, where proof lines are always clauses (i.e., single monomials) and new clauses can be derived by the *resolution rule* inferring $C \vee D$ from $C \vee x$ and $D \vee \overline{x}$. The *length* of a resolution refutation π is the number of downloaded and derived clauses, the *space* is the maximal number of clauses in any configuration, and the *width* is the size of a largest clause appearing in π (or equivalently the degree of such a monomial). Taking the minimum over all refutations as above we get the measures $L_{\mathcal{R}}(F \vdash \bot)$, $Sp_{\mathcal{R}}(F \vdash \bot)$, and $W_{\mathcal{R}}(F \vdash \bot)$. It is not hard to show that PCR can simulate resolution efficiently with respect to all these measures.

We say that a refutation is *tree-like* if every line is used at most once as the premise of an inference rule before being erased (though it can possibly be rederived later). All measures discussed above can also be defined for restricted subsystems of resolution, PC and PCR admitting only tree-like refutations.

Let us now describe the family of CNF formulas which will be the main focus of our study.

Definition A.2.2 (Tseitin formula). Let $G = (V, E)$ be an undirected graph and $\chi : V \to \{0, 1\}$ be a function. Identify every edge $e \in E$ with a variable x_e and let $PARITY_{v,\chi}$ denote the CNF encoding of the constraint that the number of true edges x_e incident to a vertex $v \in V$ is equal to $\chi(v)$ (mod 2). Then the *Tseitin formula* over G with respect to χ is $Ts(G, \chi) = \bigwedge_{v \in V} PARITY_{v,\chi}$.

When the degree of G is bounded by d, $PARITY_{v,\chi}$ has at most 2^{d-1} clauses, all of width at most d, and hence $Ts(G, \chi)$ is a d-CNF formula with at most $2^{d-1}|V|$ clauses. Figure A.1b gives an example Tseitin formula generated from the graph in Figure A.1a. We say that a set of vertices U has *odd (even) charge* if $\chi(U) = \sum_{u \in U} \chi(u)$ is odd (even). By a simple counting argument one sees that $Ts(G, \chi)$ is unsatisfiable if $V(G)$ has odd charge. Lower bounds on the hardness of refuting such unsatisfiable formulas $Ts(G, \chi)$ can be proven in terms of the expansion of G as defined next.

Definition A.2.3 (Connectivity expansion [2]). The *connectivity expansion* of $G = (V, E)$ is the largest c such that for every $E' \subseteq E$, $|E'| \leq c$, the graph $G' = (V, E \setminus E')$ has a connected component of size strictly greater than $|V|/2$.

If F is a CNF formula and $f : \{0,1\}^d \to \{0,1\}$ is a Boolean function, then we can obtain a new CNF formula by substituting $f(x_1, \ldots, x_d)$ for every variable x and expanding out to conjunctive normal form. We write $F[f]$ to denote the resulting *substituted formula*, where we will be interested in substitutions with a particular kind of functions defined as follows.

Definition A.2.4 (Non-authoritarian function [28]). We say that a Boolean function $f(x_1, \ldots, x_d)$ is *non-authoritarian* if for every x_i and for every assignment α to x_i there exist α_0, α_1 extending α such that $f(\alpha_b) = b$ for $b \in \{0,1\}$.

By way of example, exclusive or (XOR), denoted $\oplus$, is clearly non-authoritarian, since regardless of the value of one variable, the other one can be flipped to make the function true or false, but standard non-exclusive or $\vee$ is not.

Let us finally give a brief overview of the framework developed in [39], which we use to prove our PCR space lower bounds.[5] A *partial partition* $\mathcal{Q}$ of a variable set V is a collection of disjoint sets $Q_i \subseteq V$. We use the notation $\bigcup \mathcal{Q} = \bigcup_{Q_i \in \mathcal{Q}} Q_i$. For two sets of partial assignments H and H' to disjoint domains, we denote by $H \times H'$ the set of assignments $H \times H' = \{\alpha \cup \beta \mid \alpha \in H \text{ and } \beta \in H'\}$. A set of partial assignments H to the domain Q is *flippable* on Q if for each variable $x \in Q$ and $b \in \{0,1\}$ there exists an assignment $\alpha_b \in H$ such that $\alpha_b(x) = b$. We say that H *satisfies* a formula F if all $\alpha \in H$ satisfy F.

A *$\mathcal{Q}$-structured assignment set* is a pair $(\mathcal{Q}, \mathcal{H})$ consisting of a partial partition $\mathcal{Q} = \{Q_1, \ldots, Q_t\}$ of V and a set of partial assignments $\mathcal{H} = \prod_{i=1}^{t} H_i$, where each H_i assigns to and is flippable on Q_i. We write $(\mathcal{Q}, \mathcal{H}) \preccurlyeq (\mathcal{Q}', \mathcal{H}')$ if $\mathcal{Q} \subseteq \mathcal{Q}'$ and $\mathcal{H}' \restriction_{\mathcal{Q}} = \mathcal{H}$, where $\mathcal{H}' \restriction_{\mathcal{Q}} = \prod_{Q_i \in \mathcal{Q}} H'_i$. A structured assignment set $(\mathcal{Q}, \mathcal{H})$ *respects* a CNF formula F' if for every clause $C \in F'$ either $Vars(C) \cap \bigcup \mathcal{Q} = \emptyset$ or there is a set $Q \in \mathcal{Q}$ such that $Vars(C) \subseteq Q$ and $\mathcal{H}$ satisfies C.

Expressed in this language, the key technical definition in [39] is as follows.

Definition A.2.5 (Extendible family). A non-empty family $\mathcal{F}$ of structured assignment sets $(\mathcal{Q}, \mathcal{H})$ is *r-extendible* for a CNF formula F with respect to a satisfiable $F' \subseteq F$ if every $(\mathcal{Q}, \mathcal{H}) \in \mathcal{F}$ satisfies the following conditions.

Size $|\mathcal{Q}| \leq r$.

[5]The actual definitions that we use are slightly different but essentially equivalent. We provide the full details including proofs in Section A.10 for completeness.

Respectfulness $(\mathcal{Q}, \mathcal{H})$ respects F'.

Restrictability For every $\mathcal{Q}' \subseteq \mathcal{Q}$ the restriction $(\mathcal{Q}', \mathcal{H}{\restriction}_{\mathcal{Q}'})$ is in $\mathcal{F}$.

Extendibility If $|\mathcal{Q}| < r$ then for every clause $C \in F \setminus F'$ there exists $(\mathcal{Q}', \mathcal{H}') \in \mathcal{F}$ such that 1. $(\mathcal{Q}, \mathcal{H}) \preccurlyeq (\mathcal{Q}', \mathcal{H}')$, 2. $\mathcal{H}'$ satisfies C, and 3. $|\mathcal{Q}'| \leq |\mathcal{Q}| + 1$.

When $F' = \emptyset$, we simply say that $\mathcal{F}$ is r-extendible for F.

To prove PCR space lower bounds for a formula F, it is sufficient to find an extendible family for F.

Theorem A.2.6 ([39]). *Suppose that F is a CNF formula which has an r-extendible family $\mathcal{F}$ with respect to some $F' \subseteq F$. Then $Sp_{\mathcal{PCR}}(F \vdash \bot) \geq r/4$.*

All space lower bounds presented in this paper are obtained in this manner, where in addition we always have $F' = \emptyset$.

A.3 Overview of Results and Sketches of Some Proofs

In this section, we give a more detailed overview with formal statements of our results, and also provide some proof sketches in order to convey the main technical ideas. As a general rule, the upper bounds we state are for polynomial calculus (PC) whereas the lower bounds hold for the stronger system polynomial calculus resolution (PCR). In fact, even more can be said: just as is the case in [2, 79, 39], all our lower bounds hold also for *functional calculus*, where proof lines are arbitrary Boolean functions over clauses/monomials and anything that follows semantically from the current configuration can be derived in a single step. We do not discuss this further below but instead refer to Appendix A.10 for the details.

A.3.1 Relating PCR Space and Resolution Width

The starting point of our work is the question of how space and degree are related in polynomial calculus, and in particular whether it is true that degree lower-bounds space. While this question remains wide open, we make partial progress by showing that if the resolution width of refuting a CNF formula F is large (which in particular must be the case if F requires high degree), then by making XOR substitution we obtain a formula $F[\oplus]$ that requires large PCR space. In fact, this works not only for exclusive or but for any non-authoritarian function (as defined in Definition A.2.4). The formal statement is as follows.

Theorem A.3.1. *Let F be a k-CNF formula and let f be any non-authoritarian function. Then it holds over any field that $Sp_{\mathcal{PCR}}(F[f] \vdash \bot) \geq (W_{\mathcal{R}}(F \vdash \bot) - k + 1)/4$.*

Proof sketch. In one sentence, the proof of Theorem A.3.1 is by combining the concept of extendible families in Definition A.2.5 with the combinatorial characterization of resolution width in [12]. We show that the properties of F implied by the width lower bound can be used to construct an extendible family for $F[f]$. To make this description easier to parse, let us start by describing in somewhat more detail the width characterization in [12].

Consider the following game played on F by two players *Spoiler* and *Duplicator*. Spoiler asks about assignments to variables in F and Duplicator answers true or false. Spoiler can only remember ℓ assignments simultaneously, however, and has to forget some variable when this limit is reached. If Duplicator is later asked about some forgotten variable, the new assignment need not be consistent with the previous forgotten one. Spoiler wins the game by constructing a partial assignment that falsifies some clause in F, and the game is a Duplicator win if there is a strategy to keep playing forever without Spoiler ever reaching this goal. It was proven in [12] that this game exactly captures resolution width in the sense that Duplicator has a winning strategy if and only if $\ell \leq W_{\mathcal{R}}(F \vdash \perp)$.

Let us fix $r = W_{\mathcal{R}}(F \vdash \perp) - k + 1$ and use Duplicator's winning strategy for $\ell = W_{\mathcal{R}}(F \vdash \perp)$ to build an r-extendible family for $F[\oplus]$ (the proof for general non-authoritarian functions is very similar and is given in Section A.4). Consider any assignment α reached during the game. We define a corresponding structured assignment set $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha)$ by adding a block $Q_x = \{x_1, x_2\}$ to $\mathcal{Q}_\alpha$ for every $x \in \mathrm{Dom}(\alpha)$, and let H_x contain all assignments α_x to $\{x_1, x_2\}$ such that $\alpha_x(x_1 \oplus x_2) = \alpha(x)$.

Given these structured assignment sets $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha)$, the family $\mathcal{F}$ is constructed inductively as follows. The base case is that $(\mathcal{Q}_\emptyset, \mathcal{H}_\emptyset) = (\emptyset, \emptyset)$ is in $\mathcal{F}$. To extend $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha)$ to satisfy a clause in $C[\oplus]$, we simulate a Spoiler with memory α who asks about all variables in C. Since Duplicator does not falsify C, when all variables have been queried some literal in C must be satisfied by the assignment. Fix one such variable assignment $\{x = b\}$ and add $\left(\mathcal{Q}_{\alpha \cup \{x=b\}}, \mathcal{H}_{\alpha \cup \{x=b\}}\right)$ as defined above to $\mathcal{F}$. All that remains now is to verify that this yields an extendible family as described in Definition A.2.5 and then apply Theorem A.2.6. □

A.3.2 Separation of Size and Degree from Space

An almost immediate consequence of Theorem A.3.1 is that there are formulas which have small PC refutations in constant degree but nevertheless require maximal space in PCR.

Theorem A.3.2. *For any field $\mathbb{F}$ of characteristic p there is a family of k-CNF formulas F_n (where k depends on p) of size $\mathrm{O}(n)$ for which $Sp_{\mathcal{PCR}}(F_n \vdash \perp) = \Omega(n)$ over any field but which have tree-like PC refutations $\pi_n : F_n \vdash \perp$ over $\mathbb{F}$ of size $S(\pi_n) = \mathrm{O}(n \log n)$ and degree $Deg(\pi_n) = \mathrm{O}(1)$.*

Proof sketch. Let us focus on $p = 2$, deferring the general proof to Section A.5. Consider a Tseitin formula $Ts(G, \chi)$ for any constant-degree graph G over n vertices with connectivity expansion $\Omega(n)$ and any odd-charge function χ.

From [29] we know that $W_{\mathcal{R}}(F \vdash \bot) = \Omega(n)$. It is not hard to see that XOR substitution yields another Tseitin formula $Ts(G', \chi)$ for the multi-graph G' obtained from G by adding double copies of all edges. This formula requires large PCR space (over any field) by Theorem A.3.1. The upper bound follows by observing that the CNF encodes a linear system of equations, which is easily shown inconsistent in PC by summing up all equations in a tree-like fashion. □

It follows from Theorem A.3.2 that tree-like space in PC/PCR is not upper-bounded by tree-like size, in contrast to resolution. This is the only example we are aware of where the relations between size, degree, and space in PC/PCR differ from those between length, width, and space in resolution, so let us state this as a formal corollary.

Corollary A.3.3. *It is* not true *in PC/PCR that tree-like space complexity is upper-bounded by the logarithm of tree-like size complexity.*

A.3.3 Space Complexity of Tseitin Formulas

A closer analysis of the proof of Theorem A.3.2 reveals that it partitions the edge set of G' into small edge-disjoint cycles (namely, length-2 cycles corresponding to the two copies of each original edge) and uses partial assignments that all maintain the same parities of the vertices on a given cycle. It turns out that this approach can be made to work in greater generality as stated next.

Theorem A.3.4. *Let $G = (V, E)$ be a connected graph of bounded degree d with connectivity expansion c such that the edge set E can be partitioned into cycles of length at most b. Then it holds over any field that $Sp_{\mathcal{PCR}}(Ts(G, \chi) \vdash \bot) \geq c/4b - d/8$.*

Proof sketch. We build on the resolution space lower bound in [2, 77], where the proof works by inductively constructing an assignment α_t for each derived configuration $\mathbb{C}_t$ (which corresponds to removing edges from G and updating the vertex charges accordingly) such that (a) α_t satisfies $\mathbb{C}_t$, and (b) α_t does not create any odd-charge component in G of size less than $n/2$. The inductive update can be performed as long as the space is not too large, which shows that contradiction cannot be derived in small space (since $\mathbb{C}_t$ is satisfiable).

To lift this proof to PCR, however, we must maintain not just one but an exponential number of such good assignments, and in general we do not know how to do this. Nevertheless, some more thought reveals that the only important aspect of our assignments are the resulting vertex parities. And if the edge set is partitioned into cycles, we can always shift edge charges along the cycles so that for all the exponentially many assignments, the vertex parities are all the same (meaning that on a higher level we only have to maintain one good assignment after all). The full proof is presented in Section A.6. □

Some graphs, such as rectangular grids, can be partitioned into cycles of size O(1), yielding tight bounds on space. A bit more surprisingly, random d-regular graphs for $d \geq 4$ turn out to (sort of) admit partitions into cycles of size $O(\sqrt{n})$, which yields the following theorem.

Theorem A.3.5. *Let G be a random d-regular graph on n vertices, where $d \geq 4$. Then over any field it holds almost surely that $Sp_{\mathcal{PCR}}(Ts(G,\chi) \vdash \bot) = \Omega\big(\sqrt{n}\big)$.*

Proof sketch. As long as we are interested in properties holding asymptotically almost surely, we can replace random 4-regular graphs with unions of two random Hamiltonian cycles [121]. We show that a graph distributed according to the latter model almost surely decomposes into cycles of length $O(\sqrt{n})$, along with εn additional edges (which are easily taken care of separately). Since random graphs are also excellent expanders, we can apply Theorem A.3.4. The argument extends straightforwardly to random d-regular graphs for any $d \geq 4$. The full proof, which contains a bit more by way of technical details, is given in Section A.7. □

We believe that the true space bound should actually be $\Theta(n)$, just as for resolution, but such a result seems beyond the reach of our current techniques. Also, note that to make Theorem A.3.4 go through we need graph expansion *plus* partitions into small cycles. It seems plausible that expansion alone should be enough to imply PCR space lower bounds, as for resolution, but again we are not able to prove this.

A.3.4 Limitations of the PCR Space Lower Bound Technique

The framework in [39] can also be used to rederive all PCR space lower bounds shown previously in [2, 79], and in this sense [39] sums up what we know about PCR space lower bounds. There are also intriguing similarities between [39] and the resolution width characterization in [12] (as partly hinted in the proof sketch for Theorem A.3.1), which raises the question whether extendible families could perhaps be a step towards characterizing degree and showing that degree lower-bounds space in PC/PCR.

Even more intriguingly, however, there are CNF formulas for which it seems reasonable to expect that PCR space lower bounds should hold, but where extendible families seem very hard to construct. Such formulas include ordering principle formulas, functional pigeonhole principle (FPHP) formulas, and random 3-CNF formulas. In fact, no PCR space lower bounds are known for *any* 3-CNF formula—it is consistent with current knowledge that all 3-CNF formulas could have constant space complexity in PCR (!), though this seemingly absurd possibility can be ruled out for PC [79].

We show that the problems in applying [39] to the functional version of the pigeonhole principle are inherent, in that these techniques provably cannot establish *any* nontrivial space lower bound. We refer to Section A.8 for the formal description of the formulas and the proof of the next theorem.

Theorem A.3.6. *There is no r-extendible family for $FPHP^{n+1}_{n}$ for $r > 1$.*

Since by [160] these formulas[6] require PC refutation degree $\Omega(n)$, one way of interpreting Theorem A.3.6 is that the concept of r-extendible families is very far from providing the hoped-for characterization of degree.

One step towards proving PCR space lower bounds could be to obtain a weaker PC space lower bound—as noted above in the discussion of 3-CNF formulas, this can sometimes be easier. For FPHP^{n+1}_{n}, however, and for a slightly more general class of formulas described in Section A.8, it turns out that such PC space lower bounds would immediately imply also PCR space lower bounds.

Theorem A.3.7. $\mathit{Sp}_{\mathcal{PCR}}(\mathit{FPHP}^{n+1}_{n} \vdash \bot) = \Theta(\mathit{Sp}_{\mathcal{PC}}(\mathit{FPHP}^{n+1}_{n} \vdash \bot))$.

Proof sketch. In FPHP^{n+1}_{n} we have variables $x_{i,j}$ for $i \in [n+1]$, $j \in [n]$, encoding that pigeon i goes into hole j. The clauses of the formula require that every pigeon is mapped to some hole and that this mapping is one-to-one. Because of this, the negation of $x_{i,j}$ is equivalent to $\bigvee_{j' \neq j} x_{i,j'}$ and so the literal $\overline{x}_{i,j}$ can be encoded as the monomial $\prod_{j' \neq j} x_{i,j'}$ in PC. Since this substitutes a monomial for a monomial the space does not increase. Now we can take any PCR refutation of FPHP^{n+1}_{n} and apply such substitutions line by line. The inferences remain sound (with some local auxiliary steps added) and so this process gives a PC refutation of FPHP^{n+1}_{n} in roughly the same space. $\square$

A.4 PCR Space Lower Bounds From Resolution Width

In the rest of this paper, we give formal proofs of the results described in Section A.3. We start by considering the question of relating space and degree in PCR. Although we do not know how to prove (or rule out) an analogue of the relation between space and width in resolution, we can use the combinatorial game from [12] to prove a weaker relation between PCR space and resolution width. Recall from the informal description of the game in Section A.3.1 that we have two players, Spoiler and Duplicator, and that Duplicator needs to be able to provide an answer to any of Spoiler's questions about assignments to some bounded number of variables in order to win the game. Formally, a winning strategy for Duplicator is defined as follows.

Definition A.4.1 (Duplicator's strategy [12]). A *Duplicator winning strategy* for the Boolean existential ℓ-pebble game on a CNF formula F is a non-empty family $\mathcal{D}$ of partial truth value assignments to $\mathit{Vars}(F)$ such that every $\alpha \in \mathcal{D}$ satisfies the following conditions:

1. No clause $C \in F$ is falsified by α.
2. The domain of α has size at most $|\mathrm{Dom}(\alpha)| \leq \ell$.

[6]To be precise, the degree lower bound in [160] is proven for the functional pigeonhole principle encoded as linear equations—the standard CNF version has large initial width/degree and so there is nothing to prove. However, the linear-equations encoding of FPHP has axioms of large space, and so for space lower bounds we want to study the CNF version.

3. For every subassignment $\alpha' \subseteq \alpha$ it holds that $\alpha' \in \mathcal{D}$.
4. If $|\mathrm{Dom}(\alpha)| < \ell$, then for every variable x there exists an $\alpha' \in \mathcal{D}$ that assigns a value to x and extends α (i.e., $\alpha' \supseteq \alpha$).

In [12], Atserias and Dalmau proved the following tight connection between Duplicator winning strategies and resolution refutation width.

Theorem A.4.2 ([12]). *The CNF formula F has a resolution refutation of width ℓ if and only if Duplicator has no winning strategy for the Boolean existential $(\ell + 1)$-pebble game on F.*

The Duplicator strategy in Definition A.4.1 has some similarities with the extendible family in Definition A.2.5, which can be taken to suggest that there might be a relation between resolution width and PCR space. The main difference is that extendible families consist of sets of assignments in which we must be able to flip every variable, while Duplicator's strategy is built on fixed individual assignments. However, if we substitute every variable in F with a non-authoritarian function as defined in Definition A.2.4, then it is straightforward to make the transition from fixed assignments to sets of flippable assignments.

Lemma A.4.3. *Let F be a k-CNF formula and let f be a non-authoritarian function. If Duplicator wins the Boolean existential ℓ-pebble game on F, then there exists an $(\ell - k + 1)$-extendible family for $F[f]$.*

Proof. Let $\mathcal{D}$ be a winning Duplicator strategy for F. We will use $\mathcal{D}$ to construct an $(\ell - k + 1)$-extendible family $\mathcal{F}$ for the substituted formula $F[f]$. In what follows, let us denote by $\mathit{Vars}^d(x)$ the set of variables that we get when we substitute x by $f(x_1, \ldots, x_d)$ in F for some non-authoritarian function f of arity d.

For $x \in \mathit{Vars}(F)$, define $Q_x = \mathit{Vars}^d(x)$ and let $H_{x,\alpha} = \{\beta \mid \mathrm{Dom}(\beta) = Q_x \text{ and } f(\beta) = \alpha(x)\}$ be the set of all assignments over Q_x for which f evaluates to the value that α assigns to x. For any partial assignment $\alpha \in \mathcal{D}$ we let the corresponding structured assignment set $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha)$ be the pair consisting of $\mathcal{Q}_\alpha = \{Q_x \mid x \in \mathrm{Dom}(\alpha)\}$ and $\mathcal{H}_\alpha = \prod_{x \in \mathrm{Dom}(\alpha)} H_{x,\alpha}$. We define $\mathcal{F}$ to encompass all structured assignment sets $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha)$ corresponding to partial assignments $\alpha \in \mathcal{D}$ with $|\mathrm{Dom}(\alpha)| \le \ell - k + 1$. We need to prove that $\mathcal{F}$ constructed in this way is an $(\ell - k + 1)$-extendible family with respect to $F' = \emptyset$.

By construction, for every $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha) \in \mathcal{F}$ we have that $\mathcal{Q}_\alpha$ is a partial partition and that the partial assignments $H_{x,\alpha} \in \mathcal{H}_\alpha$ assign to $Q_x \in \mathcal{Q}_\alpha$. Furthermore, $H_{x,\alpha}$ is flippable on Q_x. This is so since f is a non-authoritarian function, which means that for very variable in $x_i \in Q_x$ there exist assignments β_b, $b \in \{0, 1\}$, to Q_x such that $\beta_b(x_i) = b$ and $f(\beta_b) = \alpha(x)$. Hence, all $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha) \in \mathcal{F}$ are structured assignment sets.

The size condition $|\mathcal{Q}_\alpha| \le \ell - k + 1$ in Definition A.2.5 is clearly satisfied for all $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha) \in \mathcal{F}$, and respectfulness is vacuously true. To see that the restriction property also holds, consider any $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha) \in \mathcal{F}$ obtained from $\alpha \in \mathcal{D}$. For any subset $\mathcal{Q}' \subseteq \mathcal{Q}_\alpha$, let α' be the subassignment of α restricted to $\{x \mid Q_x \in \mathcal{Q}'\}$ and let $\mathcal{H}' = \prod_{Q_x \in \mathcal{Q}'} H_{x,\alpha} = \prod_{x \in \mathrm{Dom}(\alpha')} H_{x,\alpha'}$. Then since $\alpha' \in \mathcal{D}$ by Definition A.4.1, it follows by the construction of $\mathcal{F}$ that $(\mathcal{Q}', \mathcal{H}\restriction_{\mathcal{Q}'}) = (\mathcal{Q}', \mathcal{H}') \in \mathcal{F}$ as required.

It remains to prove that $\mathcal{F}$ has the extension property. Let $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha) \in \mathcal{F}$ be such that $|\mathcal{Q}_\alpha| < \ell - k + 1$ and let C be a clause in $F[f]$. We need to argue that $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha)$ can be extended to satisfy C. Let $A \in F$ be the clause such that $C \in A[f]$, i.e., C is one of the clauses obtained when substituting f in A. If $\alpha \in \mathcal{D}$ satisfies A, it follows by construction that $\mathcal{H}_\alpha$ satisfies all of $A[f]$ and hence, in particular, C, and we are done. Otherwise, it follows from the definition of a winning Duplicator strategy and the fact that $|\alpha| \leq \ell - k$ that α can be extended to an assignment α' that queries all of the (at most k) variables in A without falsifying the clause. Such an α' must satisfy A. Fix some variable $x^* \in \mathrm{Dom}(\alpha') \setminus \mathrm{Dom}(\alpha)$ such that α' satisfies A by assigning to x^*, and let α^* be the subassignment of α' with domain $\mathrm{Dom}(\alpha) \cup \{x^*\}$. This α' must be in $\mathcal{D}$ by Definition A.4.1, and analogously to what was argued above it must hold that $\mathcal{H}_{\alpha^*}$ satisfies $C \in A[f]$. It is clear that $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha) \preccurlyeq (\mathcal{Q}_{\alpha^*}, \mathcal{H}_{\alpha^*})$, and that $|\mathcal{Q}_{\alpha^*}| \leq |\mathcal{Q}_\alpha| + 1$. Hence, $\mathcal{F}$ satisfies extendibility, and the lemma follows. □

Combining Lemma A.4.3 with the combinatorial characterization of width in Theorem A.4.2 and the lower bound on space in terms of extendible families in Theorem A.2.6, we obtain the first theorem claimed in Section A.3.

Theorem A.3.1 (restated). *Let F be a k-CNF formula and let f be any non-authoritarian function. Then*

$$Sp_{\mathcal{PCR}}(F[f] \vdash \bot) \geq \frac{W_{\mathcal{R}}(F \vdash \bot) - k + 1}{4} \ .$$

While it can be argued that this theorem might be interpreted as an indication that degree could be a lower bound for space in PCR, a more immediate and concrete consequence is that it gives us a way to prove the existence of formulas which have very small PCR refutations, but for which any refutation must have essentially maximal space. For polynomial calculus over fields of characteristic 2, we already have all the tools needed to argue this. In particular, the space lower bound needed follows immediately from Theorem A.3.1 as described next.

Corollary A.4.4. *Let G be an expander graph of bounded degree over n vertices, let χ be an odd-charge function on $V(G)$, and let G' be the multi-graph obtained by adding two copies of each edge in G. Then*

$$Sp_{\mathcal{PCR}}(Ts(G', \chi) \vdash \bot) = \Omega(n) \ .$$

Proof. As shown in [29], refuting Tseitin formulas over expander graphs requires linear width in resolution. It is not hard to see that substituting with XOR in a Tseitin formula over G is the same as considering the formula over the multi-graph with two copies of every edge. Thus $Ts(G', \chi)$ requires monomial space $\Omega(n)$ by Theorem A.3.1, which is linear in the formula size if G is a constant-degree expander. $\square$

As briefly discussed in Section A.3.2, it is not hard to show that Tseitin formulas have small refutations in PCR (and even PC) over fields of characteristic 2, which yields Corollary A.3.3 for this characteristic. However, this upper bound does not hold for characteristics distinct from 2. Therefore, we need to work with generalized version of Tseitin formulas and prove our results for such formulas instead. We do so in the next section.

A.5 Formulas With Small Proofs May Require Large Space

In Section A.2 we defined Tseitin formulas as the CNF encoding of particular linear systems over $\mathbb{F}_2$. Here we consider a generalization over fields of any positive characteristic. Any such formula essentially defines an unsatisfiable linear system over $\mathbb{F}_p$ for some prime p. In order to efficiently encode this linear system as a CNF it is important that each equation mentions a small (for instance constant) number of variables: any equation over d variables can be encoded as a set of at most 2^d clauses with d literals each. In particular, Tseitin formulas are defined on directed graph as follows.

Definition A.5.1. Let $G = (V, E)$ be a directed graph and $\chi : V \rightarrow \{0, 1, \ldots, p-1\}$ be a function. Identify every directed edge $(u, v) \in E$ with a variable $x_{(u,v)}$ and let $Mod^p_{v,\chi}$ denote the CNF encoding of the constraint that the number of *incoming* edges $x_{(u,v)}$ incident to a vertex $v \in V$ that are set to true, minus the number of *outgoing* edges $x_{(v,w)}$ set to true is equal to $\chi(v) \pmod p$. Then the *Tseitin formula* over G with respect to χ is $Ts^p(G, \chi) = \bigwedge_{v \in V} Mod^p_{v,\chi}$.

This formula is unsatisfiable when $\sum_v \chi(v) \not\equiv 0 \pmod p$. Compare Definition A.2.2 with Definition A.5.1: for $p = 2$ the definitions coincide because in such characteristic there is no difference between the contribution of the incoming and the outgoing edges. For $p = 2$ it is natural to define the formula in terms of undirected graphs, indeed. Not surprisingly, polynomial calculus over a field of characteristic p efficiently refutes unsatisfiable Tseitin formulas defined on sums modulo p.

Lemma A.5.2. *Consider a directed graph $G = (V, E)$ with n vertices and constant degree, and a function $\chi : V \rightarrow \{0, 1, \ldots, p-1\}$ with $\sum_v \chi(v) \not\equiv 0 \pmod p$. The formula $Ts^p(G, \chi)$ has a tree-like polynomial calculus refutation of constant degree, size $\mathrm{O}(n \log n)$, and monomial space $\mathrm{O}(n)$.*

Furthermore, given any boolean function f on a constant number of variables, the result holds for the substituted formula $Ts^p(G, \chi)[f]$.

Proof. Let us first consider the case without substitution. Recall that true value is encoded as 0 and false as 1. In this encoding formula $Mod^{p}_{v,\chi}$ is equivalent to

$$\sum_{u\colon (u,v)\in E} (1-x_{uv}) - \sum_{w\colon (v,w)\in E} (1-x_{vw}) \equiv \chi(v) \pmod{p} \ . \tag{A.5.1}$$

The proof is based on the natural intuition that summing the equations (A.5.1) for all vertices in the graph results in a contradiction, since in the sum each variable appears twice: once with positive and once with negative sign. Fix an enumeration of $V = \{v_1, \dots v_n\}$, and fix the following notation for partial sums:

$$S_{a,b} := \sum_{i=a}^{b} \left[\sum_{u:(u,v_i)\in E} (1-x_{uv_i}) - \sum_{w:(v_i,w)\in E} (1-x_{v_i w}) \right] \equiv \sum_{i=a}^{b} \chi(v_i) \pmod{p} \ . \tag{A.5.2}$$

We fix $t = 2^{\lceil \log n \rceil} < 2n$ and consider $S_{i,i}$ to be the equation "$0 = 0$" for all $n < i \leq t$. We set up a tree of height $\lceil \log n \rceil$, where leaves are labeled by equations $S_{i,i}$ and internal nodes are labeled by the sum of the two children labels (i.e., a node at level k is labeled by the equation $S_{i,i+2^k-1}$ for some i).

Each equation $S_{i,i}$ is derived from the encoding of $Mod^{p}_{v_i,\chi}$. This equation mention only a constant number of variables, so by implicational completeness of polynomial calculus (see Lemma A.5.3) we have a derivation of constant space and size.

Equations in internal nodes are derived by summing the equations of the children. We derive all the equations of the tree in a bottom-up fashion. This concludes the refutation since the equation $S_{1,t}$ at the root is

$$\sum_{i=1}^{n} \left[\sum_{u:(u,v_i)\in E} (1-x_{uv_i}) - \sum_{w:(v_i,w)\in E} (1-x_{v_i w}) \right] \equiv \sum_{i=1}^{n} \chi(v_i) \pmod{p} \tag{A.5.3}$$

$$\sum_{(u,v)\in E} (1-x_{uv}) - \sum_{(v,w)\in E} (1-x_{vw}) \equiv \sum_{i=1}^{n} \chi(v_i) \pmod{p} \tag{A.5.4}$$

$$0 \equiv \sum_{i=1}^{n} \chi(v_i) \pmod{p} \tag{A.5.5}$$

Which is the end of the refutation, since $\sum_{i=1}^{n} \chi(v_i)$ is non-zero.

The size of the proof accounts $O(1)$ for the deduction of each $S_{i,i}$, and $O(n)$ for the total number of monomials at each level of the tree: at level k there are $\frac{t}{2^k}$ equations with at most $O(2^k)$ monomials. So the total size is as claimed.

Regarding the monomial space, notice that we need to keep simultaneously in memory only the equations of two adjacent levels, which have at most $O(n)$ monomials.

The degree of the refutation is $O(1)$ for the inference of each equation $S_{i,i}$. The rest of the proof has degree 1.

The case with substitution is similar: consider a substituting function f on a constant number of variables. There is a multilinear polynomial p_f which evaluates exactly as f on all $\{0,1\}$ inputs, and which mentions a constant number of monomials.

The substituted linear forms $S_{i,i}[f]$ are linear combinations of copies of p_f, so they have a constant number of variables each and their inference from $Mod^p_{v_i,\chi}[f]$ is doable in constant space, size and degree because of Lemma A.5.3.

Once the equations $S_{i,i}[f]$ are derived, the refutation goes exactly as shown for the case with no substitution. From this point on the original refutation is linear; applying the trivial substitution to these proof lines increases the space, degree and size only by constant factors. $\square$

For the sake of self-containment, we give a proof of the implicational completeness of polynomial calculus. This completes the proof of Lemma A.5.2.

Lemma A.5.3. *Consider a polynomial implication $p_1,\ldots,p_l \models p$ which is valid over $\{0,1\}$ assignments. Assume all involved polynomials collectively mention d variables and have degree $\mathrm{O}(d)$; then there is a PC proof of this implication in degree $\mathrm{O}(d)$, space $2^{\mathrm{O}(d)}$, and size $2^{\mathrm{O}(d)}$.*

Proof. Without loss of generality we assume that all polynomials are in multilinear form. This is because we can transform any polynomial of degree $\mathrm{O}(d)$ between its original and multilinear version in size and space $2^{\mathrm{O}(d)}$. So each of the polynomials has size at most 2^d and degree d. Let $\alpha = \{x_1 \mapsto v_1,\ldots,x_d \mapsto v_d\}$ be an assignment; we define C_α as $\prod_i (v_i x_i + (1-v_i)(1-x_i))$, the polynomial which evaluates to 1 exactly on the assignment α. We list some useful observations:

Observation (1) is that given the axioms $\{x_i = v_i\}_{i\in[d]}$ and any polynomial q on variables $x_1,\ldots,x_d$, it is possible to efficiently infer $q-\alpha(q) = 0$. We prove this by induction on the number of variables. If $d = 0$ then $q = \alpha(q)$. Now assume that $q-\alpha(q) = s + xt - \alpha(q)$. If we have deduced $q{\restriction}_{x=0} = s - \alpha(q)$ and we have the axiom x, we can easily infer xt and then $s + xt - \alpha(q)$. If we have deduced $q{\restriction}_{x=1}$ (which is $s + t - \alpha(q)$) and we have the axiom $x-1$, we can easily infer $(x-1)t$ and then $s + t + (x-1)t - \alpha(q) = s + xt - \alpha(q)$. This derivation requires $\mathrm{O}(d)$ steps, one per variable, and both size and space are proportional to the number of monomials in q. The degree is equal to the degree of q plus d.

Observation (2) is that for any q on variables $x_1,\ldots,x_d$, we can infer from Boolean axioms the polynomial $C_\alpha(q-\alpha(q))$, for every assignment α on such variables. The inference is in degree $\mathrm{O}(d)$, and size and space are $2^{\mathrm{O}(d)}$. It is immediate for the simple case $q = x_i$: each $C_\alpha(x_i - v_i)$ contains the factor $x_i^2 - x_i$ by construction. For any non-trivial q we apply the inference in Observation (1), with the caveat that each line is multiplied by C_α. The resulting polynomial is $C_\alpha(q-\alpha(q))$.

Observation (3) is that $\sum_{\alpha\in\{0,1\}^d} C_\alpha = 1$, and this is an easy induction over d (it also follows from the semantic of polynomials C_α).

We now see how to deduce $C_\alpha p$ for every assignment α. For α which satisfy p we derive $C_\alpha(p-0)$ using observation (2). For α which falsify p, pick any falsified p_i and deduce both $C_\alpha(p_i - \alpha(p_i))$ and $C_\alpha p_i$, using observations (2) and multiplication rule, respectively. The sum is $C_\alpha \alpha(p_i)$, and since $\alpha(p_i)$ is a non-zero field element, we can multiply by $\frac{p}{\alpha(p_i)}$ to get $C_\alpha p$.

Having deduced all $C_\alpha p$ we can use observation (3) to infer p. Notice that we did 2^d inferences (one for each α), each of them of degree $\mathrm{O}(d)$ and each of them in space $2^{\mathrm{O}(d)}$, which also gives upper bound of $2^{\mathrm{O}(d)}$ on size. □

Now we have seen that (substituted) Tseitin formulas are easy for polynomial calculus under determined conditions. Nevertheless we can use the tools from Section A.4 to show that even under such conditions, any refutation requires large space.

Theorem A.5.4 (restatement of Theorem A.3.2). *For $\mathbb{F}$ any field of characteristic p there is a family of k-CNF formulas F_n (where k depends on p) of size $\mathrm{O}(n)$ for which $Sp_{\mathcal{PCR}}(F_n \vdash \bot) = \Omega(n)$ over any field but which have tree-like PC refutations $\pi_n : F_n \vdash \bot$ over $\mathbb{F}$ of size $S(\pi_n) = \mathrm{O}(n \log n)$ and degree $Deg(\pi_n) = \mathrm{O}(1)$.*

Proof. The formula family we consider is based on Tseitin formulas over a family of Ramanujan graphs of constant degree. This is a family of simple graphs with good expansion properties; a construction is given in [138]. Consider such a graph G on m vertices: set an arbitrary orientation on the edges, and consider any $\chi : [m] \rightarrow \{0,\ldots,p-1\}$ with $\sum_i \chi(i) \neq 0 \mod p$.

In Corollary 4.5 of [3], it is claimed that if G is a d-regular graph for d at least some constant value d_p, then $Ts^p(G,\chi)$ requires refutations of degree $\Omega(m)$ in polynomial calculus over any field of characteristic different from p.

Polynomial calculus simulates resolution over any characteristic, and the degree of the simulation is exactly the width of the simulated resolution proof. This implies that resolution requires width $\Omega(m)$ to refute the formula.

Fix $k = 2d$. We apply a XOR substitution on formula $Ts^p(G,\chi)$, and we get a k-CNF formula on $n = dm$ variables. Theorem A.3.1 implies that any polynomial calculus (or PCR) refutation requires monomial space $\Omega(n)$, under any characteristic.

If the characteristic of the underlying field is p the upper bound follows from Lemma A.5.2. □

A.6 PCR Space Lower Bounds for Tseitin Formulas

In the following exposition we assume that $G = (V,E)$ is a graph with connectivity expansion c and $\chi : V \rightarrow \{0,1\}$ is a Boolean function. We call a pair (G,χ) a *charged graph*, and we say that a set of vertices U is even (odd) charged if $\sum_{v\in U} \chi(v)$ is even (odd). We denote the set of edges *incident* to a vertex v by $\mathtt{edges}v$ and extend the notation to sets of vertices. We write $\overline{\alpha}$ to denote the complementary assignment of α obtained by flipping the value of all variables in the domain $\mathrm{Dom}(\alpha)$.

Definition A.6.1. The *charged graph induced by a partial assignment* α is $((V, E \setminus \mathrm{Dom}(\alpha)), \gamma)$, where $\gamma(v) = \chi(v) + \sum_{e \ni v}(1 - \alpha(e))$.

Observation A.6.2. *The formulas* $Ts((V, E \setminus \mathrm{Dom}(\alpha)), \gamma)$ *and* $Ts(G, \chi)\restriction_\alpha$ *are equivalent. An assignment* α *satisfies the clauses* $PARITY_{v,\gamma}$ *if and only if the vertex* v *is isolated and even (as a singleton set) in the charged graph induced by* α*. In that case, we say that the assignment* α *satisfies the vertex* v*.*

Definition A.6.3 (non-splitting assignment). A charged graph is *non-splitting* if all its connected components of size at most $n/2$ are even. A partial assignment α is *non-splitting* if the charged graph induced by α is non-splitting.

Observation A.6.4. *The empty assignment is non-splitting for the charged graph* (G, χ) *if and only if* (G, χ) *is non-splitting. A connected graph is always non-splitting.*

Observation A.6.5. *Suppose* α *is a partial assignment extending a partial assignment* β *(or conversely,* $\beta = \alpha\restriction_D$ *for some* $D \subseteq \mathrm{Dom}(\alpha)$*). If* α *is non-splitting, then so is* β*. In other words, "unsubstituting" an edge cannot result in an odd component that has size less than or equal to* $n/2$ *because component sizes can only increase.*

The key idea in the resolution space lower bound is that if a proof does not mention many edges, then it is possible to maintain a satisfiable assignment to the edges the proof mentions. This satisfiable assignment shifts the charge in the graph so that a contradiction only arises in vertices that the proof does not mention and leaves enough freedom to keep adding edges to the assignment unless the proof reaches a space threshold. Thus the proof is unable to derive a contradiction unless it mentions many edges at once.

The following lemma implements the charge shifting idea.

Lemma A.6.6. *Let* α *be a non-splitting assignment. Let* e *be an edge. Let* $D = \mathrm{Dom}(\alpha) \cup \{e\}$*. If* $|D| \leq c$ *then we can extend* α *to some non-splitting assignment* β *such that* $\mathrm{Dom}(\beta) = D$*.*

Proof. Let (G', γ) be the charged graph induced by α. Let $e = (u, v)$. Let C be the connected component in G' that contains the vertices u and v. Let $\alpha_0 = \alpha \cup \{e \mapsto 0\}$ and $\alpha_1 = \alpha \cup \{e \mapsto 1\}$. Let (G'', γ_0) and (G'', γ_1) be the charged graph induced by α_0 and α_1 respectively. Observe that $\gamma_0(C) = \gamma_1(C) = \gamma(C)$, where $\gamma(C) = \left(\sum_{v \in V(C)} \gamma(v)\right) \bmod 2$ for the vertices $V(C)$ of component C.

If e is not a bridge, i.e., removing the edge e from G' does not disconnect C, then we can extend α to either α_0 or α_1. In this case there is no new component.

If e is a bridge, let C' and C'' be the components in G'' that e disconnects C into. If $\gamma(C)$ is even, either both $\gamma_0(C')$ and $\gamma_0(C'')$ are even, in which case we can extend α to α_1, or both $\gamma_0(C')$ and $\gamma_0(C'')$ are odd, in which case we can extend α to α_0 reversing both parities. In this case all new components are even.

Otherwise if $\gamma(C)$ is odd, since α is non-splitting, it holds that $|C| > n/2$. Since $|D| \leq c$, the graph G'' has a connected component larger than $n/2$. The graph G' cannot have two disjoint components both larger than $n/2$, so this large component is a subset of C; either C' or C''. Assume it is C' without loss of generality. Since C is odd, either $\gamma_0(C')$ is odd and $\gamma_0(C'')$ is even, in which case we can extend α to α_1, or $\gamma_0(C')$ is even and $\gamma_0(C'')$ is odd, in which case we can extend α to α_0 reversing both parities. In this case there is one new odd component, but it is larger than $n/2$. $\square$

Corollary A.6.7. *Let α be a non-splitting assignment. Let E be a set of edges. Let $D = \mathrm{Dom}(\alpha) \cup E$. If $|D| \leq c$ then we can extend α to some non-splitting assignment β such that $\mathrm{Dom}(\beta) = D$.*

To extend this idea to a PCR lower bound for space, and in particular to the framework of [39], we need to use assignments that are not only non-splitting but also resilient to flips of the values of some variables.

Observe that if all the edges along a cycle change their value, the graph induced by the cycle stays the same. The following definition will let us formalize this property. Recall the cartesian product notation for sets of assignments.

Definition A.6.8 (Flipped assignments). Let α be a partial assignment and let $\mathcal{Q}$ be a (total) partition of $\mathrm{Dom}(\alpha)$. The set of *flipped assignments of α with respect to $\mathcal{Q}$* is the set of assignments given by

$$\mathit{Flip}(\mathcal{Q}, \alpha) = \prod_{Q \in \mathcal{Q}} \{\alpha\restriction_Q, \overline{\alpha}\restriction_Q\} \enspace .$$

Observation A.6.9. *If α is an assignment over a cycle C, then α and $\overline{\alpha}$ induce the same charged graph. Therefore, if $\mathcal{Q}$ is a set of disjoint cycles, all the flipped assignments of some assignment α with respect to $\mathcal{Q}$ induce the same charged graph.*

Theorem A.6.10 (Strengthening of Theorem A.3.4). *Let (G, χ) be a non-splitting charged graph of maximal degree d with connectivity expansion c such that a partition M of E into edge-disjoint cycles of length at most b exists. Then*

$$Sp_{\mathcal{PCR}}(Ts(G, \chi) \vdash \bot) \geq c/4b - d/8 \enspace .$$

Note that this is a strengthening of Theorem A.3.4 since if G is connected then (G, χ) is trivially non-splitting for every χ.

Proof. By Theorem A.2.6, it is sufficient to build an r-extendible family for $r = c/b - d/2$. Let $\mathcal{F}$ be the set of all pairs $(\mathcal{Q}, \mathcal{H}^\alpha)$ satisfying:

1. $\mathcal{Q} \subseteq M$ and $|\mathcal{Q}| \leq r$.

2. $\mathcal{H}^\alpha = \mathit{Flip}(\mathcal{Q}, \alpha)$, where α is any non-splitting assignment over $\bigcup \mathcal{Q}$.

Note that $\mathcal{Q}$ is a collection of edge-disjoint cycles and every $\mathcal{H}^{\alpha}$ consists of the some non-splitting assignment α and its flips over cycles. Each $(\mathcal{Q}, \mathcal{H}^{\alpha}) \in \mathcal{F}$ has many different representations, since $\mathcal{H}^{\alpha} = \mathcal{H}^{\beta}$ whenever $\beta \in \mathit{Flip}(\alpha, \mathcal{Q})$.

Let us show that $\mathcal{F}$ is an extendible family. First, pairs $(\mathcal{Q}, \mathcal{H}^{\alpha})$ are $\mathcal{Q}$-structured by construction.

The empty assignment is non-splitting by Observation A.6.4. So the family $\mathcal{F}$ is not empty because $(\emptyset, \mathcal{H}^{\emptyset}) \in \mathcal{F}$, where $\emptyset$ is the empty assignment.

Let us show that the family is closed under restriction. Consider any $(\mathcal{Q}, \mathcal{H}) \in \mathcal{F}$ and $\mathcal{Q}' \subseteq \mathcal{Q}$. Let $\alpha \in \mathcal{H}$, and let β be the restriction of α to $\bigcup \mathcal{Q}'$. By construction α is non-splitting, and restriction preserves the property of being non-splitting as noted in Observation A.6.5, so $(\mathcal{Q}', \mathcal{H}^{\beta}) \in \mathcal{F}$. Finally $\mathcal{H}\restriction_{\mathcal{Q}'} = \mathit{Flip}(\mathcal{Q}, \alpha)\restriction_{\mathcal{Q}'} = \mathit{Flip}(\mathcal{Q}', \beta) = \mathcal{H}^{\beta}$.

Let us show that the family is closed under extension. Let $(\mathcal{Q}, \mathcal{H}) \in \mathcal{F}$ with $|\mathcal{Q}| < r$ and let $p \in \mathit{PARITY}_{v,\chi}$ for some vertex $v \in V$.

If $\mathcal{H}$ satisfies p we are done; otherwise we will extend a non-splitting assignment associated with $\mathcal{H}$.

Let $\alpha \in \mathcal{H}$ be a non-splitting assignment that does not satisfy p. Let $\mathcal{Q}_v = \{C \in M \mid v \in C\}$ be the cycles adjacent to v, and let $\mathcal{Q}_+ = \mathcal{Q}_v \setminus \mathcal{Q}$; we will see that $\mathcal{Q}_+$ is not empty, but we do not need to assume it now. Let $D = \mathrm{Dom}(\alpha) \cup \bigcup \mathcal{Q}_+$. By hypothesis $|\mathcal{Q} \cup \mathcal{Q}_+| < r + d/2$, and it follows that $|D| < c$. Thus we can apply Corollary A.6.7 on α and $\bigcup \mathcal{Q}_+$ to extend α to a non-splitting assignment β over D.

The assignment β disconnects the component $\{v\}$ and is non-splitting, so it makes the component $\{v\}$ even. By Observation A.6.2, β satisfies the vertex v. Note that β falsifies the subclause of p that mentions variables in $\bigcup \mathcal{Q}$, as α does not satisfy p. If for all $C \in \mathcal{Q}_+$ and subclauses p_C of p that mention variables in C, the assignment β either falsified or satisfied all literals in p_C, then there would be a non-splitting assignment in $\mathit{Flip}(\mathcal{Q}_+, \beta)$ that falsified all literals in p. However, this cannot happen as such an assignment would falsify the vertex v, while keeping its charge the same as the satisfying assignment β.

Thus, there is a cycle $C \in \mathcal{Q}_+$ that contains one literal of p that β satisfies and one literal that β falsifies. Let $\mathcal{Q}' = \mathcal{Q} \cup \{C\}$ and let $\mathcal{H}' = \mathcal{H}^{\beta}$. By construction $(\mathcal{Q}', \mathcal{H}') \in \mathcal{F}$, and assignments in $\mathcal{H}'$ restricted to C satisfy p, showing that $(\mathcal{Q}', \mathcal{H}')$ satisfies the extension condition. □

Theorem A.3.4 is somewhat restrictive, in that it requires us to partition *all* edges in the graph into short cycles. However, as the following corollary shows, it is enough to partition *most* of the edges.

Corollary A.6.11. *Let (G, χ) be a non-splitting charged graph of maximal degree d with connectivity expansion c such that a partition M of E into edge-disjoint cycles of length at most b and an additional number of $t < c$ edges exist. Then*

$$Sp_{\mathcal{PCR}}(Ts(G, \chi) \vdash \bot) \geq (c - t)/4b - d/8$$

Proof. Let H be the graph obtained by removing the t extra edges. Note that the connectivity expansion of H is at least $c-t$. Corollary A.6.7 on page 59 shows that there exists a non-splitting assignment α on $G \setminus H$. Observation A.6.2 on page 58 implies that for some γ, (H,γ) is a non-splitting charged graph. By a restriction argument, any PCR refutation of a non-splitting Tseitin formula on G in space S can be translated to a PCR refutation of a non-splitting Tseitin formula on H in space at most S. Theorem A.3.4 shows that $S \geq (c-t)/4b - d/8$. □

A.6.1 Application: Grid Graphs

There are families of graphs where we actually get matching upper and lower bounds for PCR space. One such family is square grids. For the following subsection let n be an even integer and denote $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$, the integers modulo n. The following defines a grid over a torus.

Definition A.6.12 (Grid graph). The *grid graph* (or discrete torus) $T(n)$ is a 4-regular graph with vertices $V = \mathbb{Z}_n \times \mathbb{Z}_n$ and edges

$$E = \left\{\big((i,j),(i+1,j)\big),\big((i,j),(i,j+1)\big) \,\middle|\, i,j \in V\right\} \ ,$$

where the sums are over $\mathbb{Z}_n$. We order the vertices of $T(n)$ lexicographically: $(i,j) < (k,l)$ if $i<k$ or $i=k$ and $j<l$. The *predecessor* of a vertex $(i,j) \neq (1,1)$, denoted *pred*(i,j), is the vertex immediately preceding (i,j) in this order.

We will explicitly refer to the edges we need to disconnect a set of vertices from a graph. This notion is known as edge boundary.

Definition A.6.13. Let $G(V,E)$ be a graph and $U \subseteq V$ be a subset of vertices. The *edge boundary* of U is the set of edges $\partial_e(U) = \{(x,y) \in E : x \in U,\ y \notin U\}$.

We can find an upper bound on PC space by mentioning all the vertices in lexicographical order.

Lemma A.6.14. *The space of refuting a Tseitin formula over the $n \times n$ grid graph for an odd charge function χ over characteristic* 2 *is $Sp_{\mathcal{PC}}(Ts(T(n),\chi) \vdash \bot) = \mathrm{O}(n)$.*

Proof sketch. Observe that for every set of vertices U it holds that $\sum_{e \in \mathsf{edges}U} e \equiv \sum_{e \in \partial_e(U)} e$ (mod 2) where $\mathsf{edges}U$ is the set of all edges incident to vertices in U, and that in PC over characteristic 2 this expression corresponds to the polynomial $\sum_{e \in \partial_e(U)} e$. Thus, we can express $\sum_{e \in \mathsf{edges}U} e \equiv \chi(U)$ in space $\partial_e(U)$. If we let $U_{ij} = \{(a,b) \in V \mid (a,b) \leq (i,j)\}$, the edge boundary of any U_{ij} is at most $2n+1$, so the monomial space of each of the polynomials $p_{ij} = \sum_{e \in \partial_e(U_{ij})} e - \chi(U_{ij})$ is at most $2n+1 = \mathrm{O}(n)$.

If we show how to derive the polynomials p_{ij} in lexicographical order in $\mathrm{O}(n)$ space, we will be done. And indeed, for any vertex (i,j) in the grid graph we can infer the polynomial $q_{ij} = \sum_{e \ni (i,j)} e - \chi((i,j))$ by downloading the 2^{d-1} axioms *PARITY*$_{(i,j),\chi}$ and adding all of them in constant space. To derive p_{ij} from $p_{pred(ij)}$ it is enough to add the polynomials $p_{pred(ij)}$ and q_{ij}. The maximum space is $Sp(p_{pred(ij)}) + Sp(p_{ij}) + \mathrm{O}(1) = \mathrm{O}(n)$. □

The connectivity expansion follows from the following isoperimetric inequality.

Theorem A.6.15 ([37]). *Let U be a subset of vertices of $T(n)$ with $|U| \le n^2/2$. Then*

$$|\partial_e(U)| \ge \min\{2n, 4|U|^{1/2}\} \ .$$

Corollary A.6.16. *The connectivity expansion of $T(n)$ is $2n-1$.*

Proof. If we erase $2n-1$ or less edges from $T(n)$, then by Theorem A.6.15 the largest region we can disconnect has size $|U| \le \big((2n-1)/4\big)^2 < n^2/2$, so $c \ge 2n-1$. If we erase the $2n$ edges $\{((i,0),(i,1)) \mid i \in \mathbb{Z}_n\} \cup \{((i,n/2),(i,n/2+1)) \mid i \in \mathbb{Z}_n\}$ we obtain two connected components of size $n^2/2$, so $c < 2n$. □

The lower bound on PCR space follows.

Corollary A.6.17. *The space of refuting a Tseitin formula over the $n \times n$ grid graph, where n is even, is $Sp_{\mathcal{PCR}}(Ts(T(n),\chi) \vdash \bot) = \Omega(n)$ (over any characteristic).*

Proof. Let us find a partition of the edges of $T(n)$. Let $C(i,j)$ be the set of edges of the cycle $\big((i,j),(i+1,j),(i+1,j+1),(i,j+1)\big)$. Then the set $M = \{C(i,j) \mid i+j \equiv 0 \pmod 2\}$ is a partition of the edges of $T(n)$ into edge-disjoint cycles of length 4. By Theorem A.6.10, $Sp_{\mathcal{PCR}}(Ts(T(n),\chi) \vdash \bot) \ge (2n-9)/16$. □

Theorem A.6.18. *The space of refuting a Tseitin formula over the $n \times n$ grid graph for an odd charge function χ over characteristic 2 is $Sp_{\mathcal{PCR}}(Ts(T(n),\chi) \vdash \bot) = \Theta(n)$.*

A.6.2 Application: Triangulations

Given a graph with good expansion, we can add a few edges to it and obtain a new graph whose Tseitin formula we can prove to be hard for PCR space. We already showed in Section A.4 how to use a XOR substitution to obtain such a multi-graph; the following subsection shows how to obtain a simple graph. The proposed method is to convert every edge into a triangle, and a greedy strategy is enough as the following lemma shows.

Lemma A.6.19. *Let G be a simple graph of order n, size m and maximal degree d. If T is an integer such that $T(n-4d-2(T+1)) \ge m$ then there exists a simple graph H of maximal degree at most $2d+2T$ which is a supergraph of G whose edges can be partitioned into disjoint triangles.*

Proof. Consider the algorithm that iteratively chooses any edge (x, y) not yet handled, chooses a vertex z not adjacent to any of the endpoints of minimal degree, and adds the two remaining edges (x,z) and (y,z) from the endpoints to the vertex.

We consider the new edges to be directed (from x and y to z) and the *indegree* and *outdegree* to refer to new edges only. The degree of a vertex is thus the sum of its initial degree, its indegree and its outdegree. Observe that at every step the outdegree of every vertex is at most its initial degree, which is at most d. When choosing the vertex z, we will choose the vertex of minimal *in*degree.

Assume that at some state S of the execution of the algorithm the maximal indegree is $2t$. We claim that the algorithm handles at least the next $n-4d-4(t+1)$ edges without the indegree exceeding $2(t+1)$.

Indeed, consider the k-th edge (x, y) the algorithm visits after state S for $k \leq n-4d-4(t+1)$. Its endpoints are connected to at most $d+2(t+1)+d$ vertices each, which we discard as candidates for z, and at most $k-1$ vertices increased their indegree to $2(t+1)$. There remain at least $n-4d-4(t+1)-k+1 \geq 1$ potential vertices of indegree at most $2t$, and the greedy algorithm chooses one of these.

The initial indegree of all vertices is 0. After handling all m edges, the maximal indegree increases at most T times, where T is such that

$$m \leq \sum_{t=0}^{T-1}(n-4d-4(t+1)) = T(n-4d-2(T+1)) \ . \qquad \text{(A.6.1)}$$

$\square$

In particular, if $d \leq n/4 - \sqrt{2m}/2 - 1/2$ such a T exists, and if $d = \mathrm{o}(n)$ the inequality (A.6.1) holds asymptotically for $T = \left\lceil \frac{d+1}{2} \right\rceil$. The lower bound on space follows by applying theorem Theorem A.2.6 to the resulting supergraph and noting that the connectivity expansion cannot decrease.

Theorem A.6.20. *Let G be a simple graph of maximal degree $d = \mathrm{o}(n)$ and connectivity expansion c. There exists a simple graph H of maximal degree at most $3d+2$ which is a supergraph of G such that the space of refuting a Tseitin formula over H is at least* $Sp_{\mathcal{PCR}}(Ts(H,\chi) \vdash \bot) \geq c/12 - (3d+2)/8$.

A.7 Cycle Partitions of Random Regular Graphs

A.7.1 Models of Random Regular Graphs

Let P_n be a sequence of probability spaces. A sequence of events E_n on P_n holds *asymptotically almost surely* if $\Pr[E_n] \longrightarrow 1$. In the sequel, we often abuse notation and say that an event is true asymptotically almost surely in a probability space, when we actually mean sequences of both. The probability space will depend on a parameter n.

Two probability spaces are *contiguous* if every event which holds asymptotically almost surely in one also holds asymptotically almost surely in the other; we will use the notation $A \approx B$ to denote that A and B are contiguous. Let $\mathcal{D}_d$ be the probability space of random d-regular graphs on n vertices, $\mathcal{H} + \mathcal{H}$ be the probability space of unions of (not necessarily disjoint) random Hamilton cycles on n vertices, and $\mathcal{H} \oplus \mathcal{H}$ be the probability space of unions of disjoint random Hamilton cycles on n vertices; $\mathcal{H} \oplus \mathcal{H}$ is obtained by conditioning $\mathcal{H} + \mathcal{H}$ upon the event that the two random Hamilton cycles are disjoint. Note that $\mathcal{H} + \mathcal{H}$ is a probability space on multi-graphs. Kim and Wormald [121] proved the following theorem (see also Wormald's survey [185] and [112, §9.3–9.6]).

Theorem A.7.1. *We have $\mathcal{D}_4 \approx \mathcal{H} \oplus \mathcal{H}$.*

We will need one more fact from [121], whose proof we only sketch.

Lemma A.7.2. *If $G \sim \mathcal{H} + \mathcal{H}$ then $\Pr[G \text{ is simple}] \longrightarrow e^{-2}$.*

Proof sketch. Fix the first Hamilton cycle H_1. Let e_i be the (random) ith edge of the second Hamilton cycle H_2. It is easy to see that $\Pr[e_i \in H_1] = 2/(n-1)$, hence $\mathrm{E}[|H_1 \cap H_2|] \longrightarrow 2$. Moreover, one can show using Brun's sieve (for example [6, Theorem 8.3.1]) that the distribution of $|H_1 \cap H_2|$ is asymptotically Poisson; the required calculations are sketched in [121, §2(iii)]. Hence $\Pr[|H_1 \cap H_2| = 0] \longrightarrow e^{-2}$. □

Putting both facts together, we get the following result which will serve as our vantage point over random 4-regular graphs.

Lemma A.7.3. *Suppose E is an event which holds asymptotically almost surely in $\mathcal{H} + \mathcal{H}$. Then E also holds asymptotically almost surely for random 4-regular graphs.*

Proof. Lemma A.7.2 shows that E holds asymptotically almost surely in $\mathcal{H} \oplus \mathcal{H}$, and so in $\mathcal{D}_4$ by Theorem A.7.1. □

Corollary A.7.4. *A random 4-regular graph is connected asymptotically almost surely.*

A.7.2 Some Properties of Random Regular Graphs

For a graph $G = (V, E)$ and a subset U of the vertices, recall that $N(U)$ is the set of edges connecting U and $V \setminus U$. We say that the graph G is a *δ-expander* if for every set U of at most $|V|/2$ vertices, $|N(U)| \geq \delta |U|$. Note that our definition involves edge expansion. Bollobás [36] proved the following fundamental result.

Theorem A.7.5. *There is a constant c_1 such that asymptotically almost surely, a random 4-regular graph is a c_1-expander.*

In fact, we can choose any $c_1 < 2(1-\eta) \approx 0.4401$, where η is the unique positive solution of $(1-\eta)^{1-\eta}(1+\eta)^{1+\eta} = 2$. In particular, asymptotically almost surely a random 4-regular graph is a 0.44-expander.

The following lemma gives a lower bound on the connectivity expansion of a random 4-regular graph, defined in Definition A.2.3.

Lemma A.7.6. *There is a constant c_2 such that asymptotically almost surely, the connectivity expansion of a random 4-regular graph on n vertices is at least $c_2 n$.*

Proof. Let G be a random 4-regular graph. Theorem A.7.5 shows that asymptotically almost surely, G is a c_1-expander. Suppose G has connectivity expansion s. There is a set W of s edges and an edge e such that $G \setminus W$ has a component of size larger than $n/2$, but $G \setminus (W \cup \{e\})$ has no component of size larger than $n/2$. Since e breaks the giant component into two components, $G \setminus (W \cup \{e\})$ must have a component U of size larger than $n/4$. Expansion shows that $|N(U)| \geq c_1|U| > (c_1/4)n$, and so $s = |W| \geq (c_1/4)n$. This shows that we can choose $c_2 = c_1/4$. □

A.7.3 Simple Lower Bound

In this section we prove that refuting a non-splitting Tseitin formula on a random 4-regular graph on n vertices requires space $\Omega\big(\sqrt{n/\log n}\big)$, asymptotically almost surely over the choice of the graph.

The idea is to prove that asymptotically almost surely, a random 4-regular graph on n vertices can be partitioned into cycles of length $O\big(\sqrt{n \log n}\big)$. In order to prove that, it will be useful to consider a model related to $\mathcal{H} + \mathcal{H}$.

Let $[n] = \{1, \dots, n\}$, and let S_n be the set of all permutations on $[n]$. Every permutation $\pi \in S_n$ determines a Hamilton cycle

$$H(\pi) = (\pi(1), \pi(2)), (\pi(2), \pi(3)), \dots, (\pi(n-1), \pi(n)), (\pi(n), \pi(1)) \ . \qquad \text{(A.7.1)}$$

(The cycle is undirected.) Let ι denote the identity permutation. We will consider the probability space $\mathcal{H}(\iota) + \mathcal{H}(\pi)$ formed by taking the union of $H(\iota)$ and $H(\pi)$, where π is chosen uniformly at random from S_n.

The idea of the proof is to divide $[n]$ into $\sqrt{n/\log n}$ blocks of length $\sqrt{n \log n}$. We will show that asymptotically almost surely, each block I_k contains a point t_k such that $s_k = \pi(t_k) \in I_k$. For any two adjacent blocks I_k, I_{k+1}, we can form a cycle of length $O\big(\sqrt{n \log n}\big)$ by pasting together the path from s_k to s_{k+1} in $H(\iota)$ and the path from $\pi(t_k)$ to $\pi(t_{k+1})$ in $H(\pi)$. As a result, the graph decomposes into $\sqrt{n/\log n}$ cycles of length $O\big(\sqrt{n \log n}\big)$.

Let m be a parameter depending on n; in this section, we choose $m = \sqrt{n \log n}$, while in the next section, we choose $m = C\sqrt{n}$. For simplicity, we assume that m and n/m are both integers. We partition $[n]$ into n/m blocks $I_1, \dots, I_{n/m}$ of size m: $I_k = \{(k-1)m + 1, \dots, (k-1)m + m\}$. Let B_k be the event that $\pi(I_k) \cap I_k = \emptyset$. We think of B_k as a bad event, and our goal in this section is to show that asymptotically almost surely, none of the B_k happen. In order to show this, we estimate the probability that B_k happens.

Lemma A.7.7. *For $k \in [n/m]$, $\Pr[B_k] \leq e^{-m^2/n}$.*

Proof. Using $1-x \le e^{-x}$, we calculate

$$\Pr[B_k] = \prod_{i=0}^{m-1}\left(1-\frac{m}{n-i}\right) \le \left(1-\frac{m}{n}\right)^m \le e^{-m^2/n} \ . \tag{A.7.2}$$

□

If $\overline{B_k}$ holds, we define t_k to be the first point in I_k such that $\pi(t_k) \in I_k$, and let $s_k = \pi(t_k)$.

Lemma A.7.8. *Suppose $\overline{B_k}$ and $\overline{B_{k+1}}$ both hold (indices taken modulo n/m). Define a cycle C_k by taking two paths P_k^ι, P_k^π from $s_k = \pi(t_k)$ to $s_{k+1} = \pi(t_{k+1})$, one from each of the two Hamilton cycles:*

$$P_k^\iota = (s_k, s_k+1), (s_k+1, s_k+2), \ldots, (s_{k+1}-1, s_{k+1}) \ ,$$
$$P_k^\pi = (\pi(t_k), \pi(t_k+1)), (\pi(t_k+1), \pi(t_k+2)), \ldots, (\pi(t_{k+1}-1), \pi(t_{k+1})) \ .$$

The length of C_k is at most $4m$.

Proof. Assume for simplicity that $k \neq n/m$. Then $s_k, t_k \ge (k-1)m+1$ and $s_{k+1}, t_{k+1} \le km+m$. The length of C_k is $(s_{k+1}-s_k)+(t_{k+1}-t_k) \le 4m-2$. □

If none of the bad events happen, then the cycles $C_1, \ldots, C_{n/m}$ cover all of the graph. Choosing m accordingly, we can ensure that this happens asymptotically almost surely.

Lemma A.7.9. *Let $m = \sqrt{n \log n}$. Asymptotically almost surely, a graph chosen according to $\mathcal{H}(\iota) + \mathcal{H}(\pi)$ decomposes into n/m cycles of size at most $4m$.*

Proof. According to Lemma A.7.7, for each $k \in [n/m]$, $\Pr[B_k] \le e^{-\log n} = 1/n$. A union bound shows that asymptotically almost surely, none of the B_k happen. Lemma A.7.8 shows that the graph decomposes into n/m cycles of size at most $4m$. □

The lemma easily implies the lower bound.

Theorem A.7.10. *Asymptotically almost surely, the space required to refute in PCR any Tseitin formula on a random 4-regular graph on n vertices is $\Omega\big(\sqrt{n/\log n}\big)$.*

Proof. For reasons of symmetry, Lemma A.7.9 implies that asymptotically almost surely, a graph chosen according to $\mathcal{H} + \mathcal{H}$ decomposes into cycles of size at most $4\sqrt{n \log n}$. Lemma A.7.6 shows that asymptotically almost surely, the connectivity expansion of the graph is at least $\Omega(n)$. Corollary A.7.4 shows that asymptotically almost surely, the graph is connected, and so the Tseitin formula is non-splitting. Hence Theorem A.3.4 gives a lower bound of $\Omega(\sqrt{n/\log n})$. □

A.7.4 Improved Lower Bound

In this section we improve the results of Section A.7.3 by showing that refuting a non-splitting Tseitin formula on a random 4-regular graph on n vertices requires space $\Omega(\sqrt{n})$, asymptotically almost surely over the choice of the graph.

We use the general method of Section A.7.3, with a different choice of m, namely $m = C\sqrt{n}$ for some constant C to be determined later. Thinking of B_k as an indicator variable, let $B = \sum_{k=1}^{n/m} B_k$. Lemma A.7.7 shows that $\mathrm{E}[B] \leq e^{-C^2}(n/m)$. We will show that asymptotically almost surely, $B \leq 2e^{-C^2}(n/m)$. This implies that the cycles C_k together cover most of the graph, and therefore Corollary A.6.11 applies. The difficult part of the proof is showing that B is concentrated around its mean.

Let $p = \Pr[B_k]$ (all the probabilities are the same). We need the following strengthening of Lemma A.7.7.

Lemma A.7.11. *Let $p = \Pr[B_k]$, where B_k is the event that $I_k \cap \pi(I_k) = \emptyset$. As $n \longrightarrow \infty$, we have that $p \longrightarrow e^{-C^2}$.*

In order to show that B is concentrated around its mean, we show that for $k \neq l$, the events B_k and B_l are asymptotically negatively correlated.

Lemma A.7.12. *For every $k \neq l \in [n/m]$, $\Pr[B_k \wedge B_l] \leq p^2 + \mathrm{o}(1)$.*

We prove both lemmas below, but first, let us see how they imply the desired result. The idea is that since any two bad events are asymptotically negatively correlated, the variance of B is small, and so Chebyshev's inequality shows that B is concentrated around its mean.

Lemma A.7.13. *Asymptotically almost surely, $B \leq 2e^{-C^2}(n/m)$.*

Proof. We have $\mathrm{E}[B] = (n/m)p$ and

$$\begin{aligned}\mathrm{Var}(B) &= \mathrm{E}[B^2] - (\mathrm{E}[B])^2 \\ &= (n/m)p + (n/m)(n/m-1)(p^2+\mathrm{o}(1)) - (n/m)^2p^2 \\ &= (n/m)p(1-p) + \mathrm{o}\big((n/m)^2\big)\ ,\end{aligned}$$

using Lemma A.7.12. Chebyshev's inequality shows that

$$\Pr[|B - \mathrm{E}[B]| > \mathrm{E}[B]] \leq \frac{\mathrm{Var}(B)}{\mathrm{E}[B]^2} \leq \frac{(n/m)p + \mathrm{o}\big((n/m)^2\big)}{(n/m)^2p^2} = \mathrm{o}(1)\ , \tag{A.7.3}$$

since $p = \Omega(1)$ by Lemma A.7.11. Therefore asymptotically almost surely, $B \leq 2\,\mathrm{E}[B] = 2(n/m)p \leq 2e^{-C^2}(n/m)$, using Lemma A.7.7. □

The preceding lemma shows that the fraction of bad indices (indices k such that B_k holds) is small. Say that a block I_k is *good* if $\overline{B_k}$ and $\overline{B_{k+1}}$ both hold, and say that it is *supergood* if both I_{k-1} and I_k are good. Lemma A.7.8 associates a cycle C_k with each good block I_k. If I_k is supergood, then the cycles C_{k-1} and C_k together cover the entire stretch of I_k, as the following lemma shows.

Lemma A.7.14. *Suppose that block I_k is supergood. Then the union of the cycles C_{k-1}, C_k given by Lemma A.7.8 contains the path of length m from $\min I_k$ to $\min I_{k+1}$ in $H(\iota)$, as well as the path of length m from $\pi(\min I_k)$ to $\pi(\min I_{k+1})$ in $H(\pi)$.*

Proof. The cycle C_{k-1} contains the path from $s_{k-1} < \min I_k$ to s_k in $H(\iota)$. The cycle C_k contains the path from s_k to $s_{k+1} \geq \min I_{k+1}$ in $H(\iota)$. Both paths together cover the path from $\min I_k$ to $\min I_{k+1}$ in $H(\iota)$. The argument for $H(\pi)$ is identical. □

We can now prove an analogue of Lemma A.7.9.

Lemma A.7.15. *Let $m = C\sqrt{n}$. Asymptotically almost surely, a graph chosen according to $\mathcal{H}(\iota) + \mathcal{H}(\pi)$ decomposes into cycles of size at most $4m$ and t additional edges, where $t \leq 8e^{-C^2}n$.*

Proof. Lemma A.7.13 shows that asymptotically almost surely, all but $4e^{-C^2}(n/m)$ of the n/m blocks $I_1, \ldots, I_{n/m}$ are supergood, as each bad block prevents two blocks from being supergood. Let $\mathcal{C}$ be the (disjoint) union of all cycles C_k constructed using Lemma A.7.8 for all supergood blocks I_k. The lemma shows that each cycle has size at most $4m$. Lemma A.7.14 shows that $\mathcal{C}$ contains all but at most $8e^{-C^2}n$ edges of the graph. □

Replacing Theorem A.3.4 with its corollary, Lemma A.7.15 easily implies the lower bound.

Theorem A.7.16. *Asymptotically almost surely, the space required to refute in PCR any Tseitin formula on a random 4-regular graph on n vertices is $\Omega(\sqrt{n})$.*

Proof. For reasons of symmetry, Lemma A.7.15 implies that asymptotically almost surely, a graph chosen according to $\mathcal{H} + \mathcal{H}$ decomposes into cycles of size at most $4C\sqrt{n}$ and t additional edges, where $t \leq 8e^{-C^2}n$. For an appropriate choice of C we have $t \leq (c_2/2)n$. Lemma A.7.6 shows that asymptotically almost surely, the connectivity expansion of the graph is at least $c_2 n$. Corollary A.7.4 shows that asymptotically almost surely, the graph is connected, and so the Tseitin formula is non-splitting. Hence Corollary A.6.11 gives a lower bound of $\Omega(\sqrt{n})$. □

Technical Lemmas

We now turn to the proofs of Lemma A.7.11 and Lemma A.7.12. We start with the former.

Proof of Lemma A.7.11. It is easy to check that for $0 \leq x \leq 1/2$, $1-x \geq e^{-x-x^2}$. Therefore for large enough n,

$$p = \prod_{i=0}^{m-1}\left(1-\frac{m}{n-i}\right) \geq \left(1-\frac{m}{n-m}\right)^m \geq \exp\left[-\frac{m^2}{n-m}-\frac{m^3}{(n-m)^2}\right] . \tag{A.7.4}$$

For large enough n, $m \leq n/2$, and so $m^2/(n-m) = m^2/n + m^3/(n(n-m)) \leq m^2/n + 2m^3/n^2$. Similarly, $m^3/(n-m)^2 \leq 4m^3/n^2$. Therefore, using $e^{-x} \geq 1-x$,

$$p \geq \exp\left[-\frac{m^2}{n}-6\frac{m^3}{n^2}\right] = \exp\left[-C^2-\frac{6C^3}{\sqrt{n}}\right] \geq e^{-C^2}\left(1-\frac{6C^3}{\sqrt{n}}\right) . \tag{A.7.5}$$

Hence $\liminf p \geq e^{-C^2}$. Lemma A.7.7 shows that also $\limsup p \leq e^{-C^2}$. □

The proof of Lemma A.7.12 is more involved. Recall that the lemma claims that the events B_k and B_l are asymptotically negatively correlated. In fact, they are asymptotically uncorrelated. Recall that $\Pr[B_k]$ is roughly equal to e^{-C^2}. Given the value of π on I_k, the probability $\Pr[B_l]$ depends on $|\pi(I_k) \cap I_l|$. Typically, this intersection will be very small, and so $\Pr[B_l]$ is also roughly equal to e^{-C^2}.

We will show that $|\pi(I_k) \cap I_l|$ is typically small using an extension of the well-known Chernoff bound due to Kabanets and Impagliazzo [108, Theorem 1.1], attributed there to Panconesi and Srinivasan [148].

Theorem A.7.17. *Let $X_1, \ldots, X_r$ be Boolean random variables such that for any set $S \subseteq [r]$, $\Pr[\bigwedge_{i \in S} X_i] \leq \delta^{|S|}$. Then for $\gamma \geq \delta$,*

$$\Pr\left[\sum_{i=1}^{r} X_i \geq \gamma r\right] \leq e^{-2r(\gamma-\delta)^2} .$$

The following lemma applies this bound to our situation (in an abstracted version).

Lemma A.7.18. *Let a, b, c be integers such that $a \geq b, c$, and let T be a random subset of $[a]$ of size b. For all $\rho \geq 1$,*

$$\Pr[|T \cap [c]| \geq \rho(bc/a)] \leq e^{-2c(\rho-1)^2(b/a)^2} .$$

Proof. For $i \in [c]$, let X_i be the event that $i \in T$. For $S \subseteq [c]$ such that $|S| \leq b$,

$$\Pr_T[S \subseteq T] = \frac{\binom{a-|S|}{b-|S|}}{\binom{a}{b}} = \prod_{k=0}^{|S|-1}\frac{b-k}{a-k} \leq \left(\frac{b}{a}\right)^{|S|} . \tag{A.7.6}$$

Therefore we can apply Theorem A.7.17 with $r = c$, $\delta = b/a$ and $\gamma = \rho(b/a)$. □

We can now prove Lemma A.7.12.

Proof of Lemma A.7.12. We will show that $\Pr[B_l \mid B_k] \le p + o(1)$. This implies that $\Pr[B_k \wedge B_l] = \Pr[B_k]\Pr[B_l \mid B_k] \le p(p + o(1)) = p^2 + o(1)$.

Assuming the event B_k happens, $\pi(I_k)$ is a random subset of $[n] \setminus I_k$ of size m. Plugging $a = n - m$ and $b = c = m$ in Lemma A.7.18, we deduce that for all $\rho \ge 1$ there is a n_0 such that for all $n \ge n_0$

$$\Pr[|\pi(I_k) \cap I_l| \ge \rho C^2 \mid B_k] \le e^{-2(\rho-1)^2 m(m/(n-m))^2} \tag{A.7.7}$$

$$\le e^{-2(\rho-1)^2 m^3/n^2} = e^{-2C^3(\rho-1)^2/\sqrt{n}} \ . \tag{A.7.8}$$

Hence with probability $1 - o(1)$ given B_k, $D =: |\pi(I_k) \cap I_l| \le \sqrt{m \log m}$. Now

$$\Pr[B_l \mid D = d] = \prod_{i=0}^{m-1} \left(1 - \frac{m-d}{n-m-i}\right) \le \left(1 - \frac{m-d}{n}\right)^m \le e^{-m(m-d)/n} \ . \tag{A.7.9}$$

For $0 \le x \le 1$, one can check that $e^x \le 1 + 2x$. Hence

$$\Pr[B_l \mid D \le \sqrt{m \log m}] \le e^{-m(m-\sqrt{m \log m})/n} \tag{A.7.10}$$

$$= e^{-C^2 + m\sqrt{m \log m}/n} \le e^{-C^2}\left(1 + \frac{2m\sqrt{m \log m}}{n}\right) \ . \tag{A.7.11}$$

Using Lemma A.7.11, we deduce that $\Pr[B_l \mid D \le \sqrt{m \log m}] \le e^{-C^2} + o(1) = p + o(1)$. We conclude that $\Pr[B_l \mid B_k] = p + o(1)$ and so $\Pr[B_k \wedge B_l] = p^2 + o(1)$. □

A.7.5 Regular Graphs of Degree Larger Than Four

Wormald [185, Corollary 4.17] showed that when $d > 4$, a random d-regular graph can be obtained (up to contiguity) by taking the disjoint union of a random 4-regular graph and a random $(d-4)$-regular graph, a result summarized in the following theorem (see also [112, Corollary 9.44]).

Theorem A.7.19. *For $d > 4$ we have $\mathcal{D}_d \approx \mathcal{D}_4 \oplus \mathcal{D}_{d-4}$. Furthermore, the probability that a uniformly random 4-regular graph and a uniformly random $(d-4)$-regular graph do not intersect tends to a positive constant.*

A Tseitin formula on a random d-regular graph generated according to $\mathcal{D}_4 \oplus \mathcal{D}_{d-4}$ is harder to refute than a Tseitin formula on a random 4-regular graph, and so we can generalize Theorem A.7.16 to random d-regular graphs for arbitrary $d \ge 4$.

Theorem A.7.20 (restatement of Theorem A.3.5). *Let $d \ge 4$. Asymptotically almost surely, the space required to refute in PCR any Tseitin formula on a random d-regular graph on n vertices is $\Omega(\sqrt{n})$.*

Proof. If $d = 4$ then Theorem A.7.16 already applies, so assume $d > 4$. Let G_1 be a random 4-regular graph, and let G_2 be a random $(d-4)$-regular graph. The graph $G = G_1 + G_2$ is distributed according to $\mathcal{D}_4 + \mathcal{D}_{d-4}$. We show below that asymptotically almost surely, the space required to refute in PCR any Tseitin formula on G is $\Omega\big(\sqrt{n}\big)$. Since G_1 and G_2 are disjoint with constant probability according to Theorem A.7.19, the theorem follows.

Let α be an arbitrary assignment to the edges of G_2. Observation A.6.2 on page 58 shows that for every function f, $\mathit{Ts}(G,\chi)\restriction_\alpha = \mathit{Ts}(G_1,\gamma)$ for some other function γ. By a restriction argument, any PCR refutation of $\mathit{Ts}(G,\chi)$ in space S can be translated to a PCR refutation of $\mathit{Ts}(G_1,\gamma)$ in space at most S. Theorem A.7.16 on page 68 shows that asymptotically almost surely, we must have $S = \Omega\big(\sqrt{n}\big)$. □

A.8 Current Techniques and the Functional Pigeonhole Principle

We now discuss the intrinsic limitations of the techniques employed so far. In Section A.8.1 we show that Bonacina-Galesi framework does not allow to prove PCR space lower bounds for an interesting formula like functional pigeonhole principle. In Section A.8.2 we show that restricting to PC does not make the problem easier.

A.8.1 FPHP Formulas Do Not Have Extendible Families

One of the limits of the Bonacina-Galesi framework is that we cannot apply it to formulas for which fixing a small set of variables causes a lot of unit clause propagation. Indeed, most of the lower bound strategies in this paper aim to control this phenomenon (see for example Lemma A.4.3). For the functional pigeonhole principle these strategies do not work, as we now prove.

Definition A.8.1. The *functional pigeonhole principle* on m pigeons and n holes is the formula defined on variables x_{ij} for $i \in [m]$ and $j \in [n]$, made of the following clauses:

$$\bigvee_{j\in[n]} x_{ij} \qquad \text{for all } i \in [m]; \qquad \text{(pigeon axioms)}$$

$$\neg x_{ij} \vee \neg x_{i'j} \qquad \text{for any } i \neq i' \in [m] \text{ and } j \in [n]; \qquad \text{(hole axioms)}$$

$$\neg x_{ij} \vee \neg x_{ij'} \qquad \text{for any } i \in [m] \text{ and } j \neq j' \in [n]. \qquad \text{(functional axioms)}$$

It is already known that this formula requires large space in resolution [29, 12]. It is natural to suspect that this formula is hard in terms of monomial space as well. However, the Bonacina-Galesi framework is not strong enough to prove it.

Theorem A.8.2 (restatement of Theorem A.3.6). *There is no r-extendible family for FPHP^m_n for $r > 1$.*

Proof. Assume that there is an r-extendible family $\mathcal{F}$ for the formula FPHP^m_n which respects some satisfiable $F' \subseteq \mathit{FPHP}^m_n$, for $r > 1$.

Let C be any clause in $FPHP^m_n \setminus F'$; such clause exists because $FPHP^m_n$ is a contradiction. The extension property of $\mathcal{F}$ implies that there is a pair $(\{Q_1\}, H_1) \in \mathcal{F}$, where H_1 satisfies C.

Recall that 0 encodes true, and 1 encodes false. Pick a variable x_{ij} in Q_1. In H_1 there is at least one partial assignment for which $x_{ij} = 0$, and for any such assignment it holds that $x_{i'j} = 1$ and $x_{ij'} = 1$ for all $i' \neq i$ and $j' \neq j$, otherwise an initial clause would be false.

Indeed, fix v to be any of these variables (either $x_{i'j}$ or $x_{ij'}$); the clause $\neg x_{ij} \vee \neg v$ is an axiom. If $v \notin Q_1$ then this clause is not in F' because of the respectfulness of $\mathcal{F}$, and furthermore there is at least one assignment in H_1 which does not satisfy it (i.e., any assignment with $x_{ij} = 0$). The extension property of $\mathcal{F}$ guarantees that there is $(\{Q_1, Q_2\}, H_1 \times H_2) \in \mathcal{F}$ with $v \in Q_2$, such that $H_1 \times H_2$ satisfies $\neg x_{ij} \vee \neg v$. But this contradicts the fact that $H_1 \times H_2$ contains the assignment $\{x_{ij} = 1, v = 1\}$, which falsifies $\neg x_{ij} \vee \neg v$.

It follows that $\{x_{i'j}, x_{ij'} \mid i' \neq i \text{ and } j' \neq j\} \subseteq Q_1$, and that H_1 satisfies all axioms involving either pigeon i or hole j. We have just shown that assuming some $x_{ij} \in Q_1$, we get $\{x_{i'j}, x_{ij'} \mid i' \in [m], j' \in [n]\} \subseteq Q_1$. This choice was arbitrary, so it follows that for any $i \in [m], j \in [n]$, the variable x_{ij} is in Q_1. In other words, Q_1 contains all the variables. Since $FPHP^m_n \setminus F'$ is contradictory, every assignment in H_1 falsifies some clause, and so the extension property fails for any such clause. We conclude that $FPHP^m_n$ has no 2-extendible family. □

A.8.2 Formulas with Equal PC and PCR Space Complexities

Although finding an r-extendible family for the functional pigeonhole principle (and hence proving a space lower bound) is not feasible, we might try and prove a weaker PC space lower bound. However, as we have pointed out in Section A.3.4, in the case of functional pigeonhole principle this makes no difference. In this section, we prove formally this result for a broader class of formulas that is captured by the following definition.

Definition A.8.3. We say that a CNF formula F is *totally weight constrained* if for every variable x appearing in F there exists a clause $C_x \in F$ with the following properties:

1. All literals in C_x are positive;
2. x is one of the variables appearing in C_x;
3. For every two distinct variables y, z appearing in C_x, clause $\overline{y} \vee \overline{z}$ is in F.

For each variable x we refer to C_x as the *x-neighborhood clause*.

In such formulas each negative literal can be replaced with a clause/monomial consisting of only positive literals that has the same semantic meaning. Thus, we can turn a PCR refutation into a PC refutation without any substantial loss of space. In order for us to be able to show that such a refutation is a valid PC refutation we need to show that there are PC derivations of these monomials that use small space.

Theorem A.8.4. *For a totally weight constrained CNF formula F, where each clause has a costant number of negative literals, it holds that $Sp_{PC}(F \vdash \perp) = \Theta(Sp_{PCR}(F \vdash \perp))$.*

Proof. We can easily see that PCR simulates PC with only a constant loss in space. The only problem in the simulation could arise when downloading an axiom that has negative literals. Nevertheless, it is not hard to prove that PCR can expand every axiom to its PC form while respecting the stated space bound.

In the other direction, we prove that PC can simulate a PCR refutation of F. Let π be a PCR refutation of F in space at most s. As F is a totally weight constrained formula, for every variable x we can fix its x-neighborhood clause C_x. Let us denote by $N(x)$ the set of variables from C_x excluding x. We transform the PCR refutation π into a PC refutation by replacing each negative literal $\overline{x}$ with the monomial $\prod_{y \in N(x)} y$. Obviously this transformation preserves space and we need to show that the transformed configurations form a backbone of a valid PC refutation.

If the PCR refutation deletes a polynomial, we delete the appropriate transformed polynomial from the configuration in the PC refutation. Similarly, in the case of linear combination steps we just deduce the linear combination of the transformed polynomials. Hence, these two types of steps can be done without any loss in space. In the case of multiplication with a literal, if the literal is positive we multiply the appropriate transformed polynomial with the same literal. Otherwise, the literal is negative and we multiply the polynomial with all the variables in $N(x)$, where $\overline{x}$ is the literal, while making sure to delete the intermediate polynomials when they are no longer needed. In this way we derive the transformed polynomial in at most $O(s)$ space.

The axiom download steps are the only ones that remain. In the case of Boolean axiom download, if we downloaded an axiom for a positive literal, we just download the appropriate axiom in the PC refutation. Otherwise, the Boolean axiom corresponds to some negative literal $\overline{x}$ and we need to derive the polynomial $\prod_{y \in N(x)} y^2 - \prod_{y \in N(x)} y$. This is done by downloading the Boolean axioms for each $y \in N(x)$ and combining them to get the transformed polynomial. Let $B^2 - B$ be one of the intermediate polynomials in the derivation of the transformed Boolean axiom, where B is a monomial formed by multiplying the variables in some subset of $N(x)$. Then, for some variable y not mentioned in B, we derive $(By)^2 - By$ by downloading $y^2 - y$ and taking the linear combination of $y(B^2 - B)$ and $B^2(y^2 - y)$. This PC derivation uses $O(1)$ more monomials than the PCR axiom download.

When the PCR proof downloads the complementarity axiom $1 - x - \overline{x}$, the corresponding PC proof needs to derive the polynomial $1 - x - \prod_{y \in N(x)} y$. Let $N(x) = \{y_1, \ldots, y_l\}$. We derive the transformed polynomial by successively deriving polynomials

$$T(i) = \prod_{k=i+1}^{l} y_k - x \prod_{k=i+1}^{l} y_k - \prod_{k} y_k \ , \tag{A.8.1}$$

for $i = 1,\ldots,l$. Note that $T(l)$ is our transformed polynomial. The first $T(1)$ in the PC proof can be derived by downloading the axiom $(1-x)(1-y_1)$ and multiplying it with variables $y_2,\ldots,y_l$ in order to get $T(1) + x\prod_k y_k$. Subtracting from it the x-neighborhood clause $C_x = x\prod_k y_k$ we get $T(1)$.

We proceed to derive $T(i+1)$ from $T(i)$ for all i. Similarly as before, we start by downloading the axiom $(1-x)(1-y_{i+1})$ and multiplying it with variables $y_{i+2},\ldots,y_l$ in order to get $T(i+1)-T(i)$. Adding this polynomial to $T(i)$ we derive the $(i+1)$st polynomial $T(i+1)$ in our derivation of the transformed complementarity axiom. This PC derivation uses O(1) more monomials than the PCR proof and all axioms of the form $(1-x)(1-y_i)$ exist because F is totally weight constrained.

In the case of axiom download step for a clause axiom, we again have two cases. If all literals of the axiom are positive we download the corresponding axiom in the PC proof. Otherwise, we can write the axiom as $\overline{x_1}\cdots\overline{x_s}\cdot x_{s+1}\cdots x_l$, where s is the number of its negative literals. Let us denote by $A(i)$ the polynomial

$$A(i) = \prod_{y_1\in N(x_1)} y_1 \cdots \prod_{y_i\in N(x_i)} y_i(1-x_{i+1})\cdots(1-x_s)x_{s+1}\cdots x_l \ , \tag{A.8.2}$$

where i ranges over $0,\ldots,s$. Note that $A(0)$ is the original PC axiom, while $A(s)$ is the transformed axiom that we want to derive. Also, let us denote by $R(i)$ the polynomial

$$R(i) = \prod_{y_1\in N(x_1)} y_1 \cdots \prod_{y_{i-1}\in N(x_{i-1})} y_{i-1}\cdot(1-x_{i+1})\cdots(1-x_s)x_{s+1}\cdots x_l \ , \tag{A.8.3}$$

for i ranging from 1 to s, that is $A(i) = R(i)\prod_{y_i\in N(x_i)} y_i = R(i+1)(1-x_{i+1})$.

We first derive $A(1)$ by deriving the transformed complementarity axiom $1-x_1-\prod_{y_1\in N(x_1)} y_1$ for the variable x_1 and multiplying it with $R(1)$ in order to get $A(0)-A(1)$. Now we can get $A(1)$ by subtracting the derived polynomial from the PC axiom $A(0)$.

We proceed to derive $A(s)$ by deriving $A(i+1)$ from $A(i)$ for all i from 1 to $s-1$. This is again done by first deriving the appropriate complementarity axiom $1-x_{i+1}-\prod_{y_{i+1}\in N(x_{i+1})} y_{i+1}$ and multiplying it by $R(i+1)$ in order to get $A(i)-A(i+1)$. Subtracting the derived polynomial from previously derived $A(i)$, we get the $(i+1)$st polynomial in our derivation. These steps use $O(2^s)$ monomials, which is constant by the theorem hypothesis, and the PC derivation of the transformed axiom uses at most O(1) monomials more than the PCR axiom download step.

Hence, the theorem follows. Also, although we have ignored the constants involved in the simulation, these constants can be computed explicitly and are small. The only possible exception is the additive constant $O(2^{s^*})$, where s^* is the largest number of negative literals in a clause of F. □

An obvious example of the totally weight constrained formula is the functional pigeonhole principle.

Corollary A.8.5 (Restatement of Theorem A.3.7). *It holds that*

$$Sp_{PCR}(FPHP^m_n \vdash \bot) = \Theta(Sp_{PC}(FPHP^m_n \vdash \bot)) \ .$$

Proof. It is easy to see that $FPHP^m_n$ formula is totally weight constrained, as every variable appears in some pigeon axiom that is constrained by the functional axioms. Also, $FPHP^m_n$ has at most 2 negative literals in each clause and hence we have that $Sp_{PCR}(FPHP^m_n \vdash \bot) = \Theta(Sp_{PC}(FPHP^m_n \vdash \bot))$. □

Actually, we can say even more about the space complexity of the functional pigeonhole principle formulas. In [79], the authors prove that the PCR space complexity of $FPHP^m_n$ is equal (up to constant factors) to the PCR space complexity of the extended formula $\widetilde{FPHP}^m_n$, where $\widetilde{FPHP}^m_n$ is the canonical equivalent 3-CNF version[7] of the formula $FPHP^m_n$. Hence, we have that the PC space complexity lower bound for $FPHP^m_n$ would actually lower bound the PCR space complexity of $\widetilde{FPHP}^m_n$ and give us the first PCR space lower bound for some family of 3-CNF formulas.

This holds in greater generality for totally weight constrained formulas that also fulfill the following technical condition: F is a *weight-constrained* CNF formula if for each clause $a_1 \vee a_2 \vee \ldots \vee a_m$ of F with more than three literals, the formula also contains clauses $\neg a_i \vee \neg a_j$ for all $1 \le i < j \le m$. We stress the fact that the conditions of being weight-constrained and totally weight constrained are incomparable.

Corollary A.8.6. *For a simultaneously weight-constrained and a totally weight constrained formula F, where each clause has a costant number of negative literals, it holds that*

$$Sp_{PCR}(\widetilde{F} \vdash \bot) = \Theta(Sp_{PCR}(F \vdash \bot)) = \Theta(Sp_{PC}(F \vdash \bot)) \ .$$

A.9 Concluding Remarks

In this paper, following up on recent work in [23, 39, 79, 106], we report further progress on understanding space complexity in polynomial calculus and how the space measure is related to size and degree. Specifically, we separate size and degree from space, and provide some circumstantial evidence for the conjecture that degree might be a lower bound on space in PC/PCR. We also prove space lower bounds for a large class of Tseitin formulas, a well-studied formula family for which nothing was previously known regarding PCR space.

We believe that our lower bounds for Tseitin formulas over random graphs are *not* optimal, however. And for the functional pigeonhole principle, we show that the technical tools developed in [39] cannot prove any non-constant PCR space lower bounds. Although we have not been able to prove this, we believe that similar impossibility results should hold also for ordering principle formulas and for the canonical 3-CNF version of the pigeonhole principle. Since all of these formulas require large degree in PCR and large space in resolution, it is natural to suspect that they should be hard for PCR

[7]We substitute every clause $a_1 \vee a_2 \vee \ldots \vee a_k$, which has more than three literals, with the formula $(a_1 \vee y_1) \wedge (\neg y_1 \vee a_2 \vee y_2) \wedge \ldots \wedge (\neg y_{i-1} \vee a_i \vee y_i) \wedge \ldots \wedge (\neg y_{k-1} \vee a_k)$ where for each substituted clause all variables y_i are new. The substituted formula is a 3-CNF and it is satisfiable if and only if the original one is. It is also easy to deduce the original clause from the substituting formula.

space as well. The fact that arguments along the lines of [39] do not seem to be able to establish this suggests that we are still far from a combinatorial characterization of degree analogous to the characterization of resolution width in [12].

It thus remains a major open problem to understand the relation between degree and space in PC/PCR, and in particular whether degree is a lower bound on space or not (or whether it even holds that resolution width provides a lower bound on PCR space).

Also, our separations of size and degree on the one hand and space on the other depend on the characteristic of the underlying field, in that the characteristic must be chosen first and the formula family exhibiting the separation works only for this specific characteristic. It would be satisfying to find formulas that provide such separations regardless of characteristic. Natural candidates are (various flavours of) ordering principle formulas or onto function pigeon principle formulas, or, for potentially even stronger separations, pebbling formulas.

Finally, an intriguing question is how (monomial) space in PC/PCR is related to (clause) space in resolution. There are separations known for size versus length and degree versus width, and it would seem reasonable to expect that PCR should be strictly stronger than resolution also with respect to space, but this is completely open.[8] The flipside of this question is to what extent space lower bound techniques for resolution carry over to PC/PCR. Since so far we do not know of any counter-examples, it is natural to ask, for instance, whether *semiwide* CNF formulas as defined in [2] have high space complexity not only in resolution but also in PCR.

Acknowledgements

The authors wish to thank Ilario Bonacina and Nicola Galesi for numerous and very useful discussions.

The research of the first author has received funding from the European Union's Seventh Framework Programme (FP7/2007–2013) under grant agreement no. 238381. Part of the work of the first author was performed while visiting KTH Royal Institute of Technology. The other authors were funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 279611. The fourth author was also supported by Swedish Research Council grants 621-2010-4797 and 621-2012-5645.

A.10 PCR Space Lower Bounds from Extendible Families

For the sake of self-containment, in this appendix we give an exposition of the Bonacina-Galesi framework [39] for proving space lower bounds in Polynomial Calculus. We show how the existence of a r-extendible family for a large value of r implies such bounds.

[8]For completeness, we mention that there is a very weak (constant-factor) separation in [2], but it crucially depends on a somewhat artificial definition of space where monomials are *not* counted with repetitions.

This framework can actually prove space lower bounds for a proof system that it stronger than PC or PCR.

Definition A.10.1 (Functional Calculus (FC)). A *functional calculus* configuration is a set of arbitrary Boolean functions over Boolean variables. There is a single derivation rule, *semantic implication*, where g can be inferred from $f_1, \dots, f_n$ if every assignment that satisfies $f_1 \wedge \cdots \wedge f_n$ also satisfies g.

Verifying a proof in FC is coNP-complete, and so FC is not a proof system in the sense of Cook and Reckhow [66] unless $\mathsf{coNP} = \mathsf{NP}$.

There are many different circuit representations of the same Boolean function, so we need to choose a minimal representation in order to define clause space.

Definition A.10.2. Let $\mathbb{P}$ be a FC configuration. A set of monomials $U = \{m_1, \dots, m_s\}$ *defines* $\mathbb{P}$ if for every function $f \in \mathbb{P}$ there is a function g such that $g(m_1, \dots, m_s) \equiv f(x_1, \dots, x_n)$. The *monomial space* of $\mathbb{P}$ is the minimum size of a defining set of monomials.

We can interpret polynomials in PCR as Boolean functions if we project them to the Boolean ring $\mathbb{F}[x, \overline{x}, y, \overline{y}, \dots]/\operatorname{Span}\big(x^2 - x, 1 - x - \overline{x}, \overline{x}^2 - \overline{x}, y^2 - y, \dots\big)$. Furthermore, the set of monomials in a PCR configuration counted without repetitions is a defining set of monomials for a FC configuration. Therefore we can view every proof in PCR as a proof in FC that uses at most the same space. In particular, $Sp_{\mathcal{FC}}(F \vdash \bot) \le Sp_{\mathcal{PCR}}(F \vdash \bot)$.

We now prove Theorem A.2.6, following Bonacina and Galesi [39]. The general plan of the proof is to consider a FC derivation of a formula F in small space, and show that every configuration arising in the derivation is satisfiable. Since a refutation ends with an unsatisfiable configuration, the derivation is not a refutation.

In order to show that every configuration arising in the derivation is satisfiable, we maintain a satisfiability witness, in the form of a structured set of assignments together with a CNF formula. The following definition captures the sense in which a satisfiability witness guarantees that a board configuration is satisfiable. Fix a set of variables V and consider partitions and total assignments with respect to this set. Recall that a total assignment assigns a value to *each* variable in V.

Definition A.10.3. Let $(\mathcal{Q}, \mathcal{H})$ be a structured set of assignments, G be a CNF formula, and $\mathbb{P}$ be a set of Boolean functions. We write $G \models_{(\mathcal{Q},\mathcal{H})} \mathbb{P}$ if every total assignment that extends some partial assignment in $\mathcal{H}$ and satisfies G also satisfies $\mathbb{P}$.

In the proof, $\mathbb{P}$ is the contents of the board at a given point in the FC refutation, and $(\mathcal{Q}, \mathcal{H}), G$ together form a satisfiability witness. The CNF G is composed of two parts: a satisfiable subset $F' \subset F$, which could be empty, and a 2-CNF M with a very specific form given by the following definition.

Definition A.10.4. Let M be a 2-CNF formula over the variables V. We say that M is a *transversal* of a partial partition $\mathcal{Q}$ defined on V if M mentions exactly one variable from each block $Q_i \in \mathcal{Q}$. (In particular, $|\mathcal{Q}|$ must be even and the number of clauses in M is $|\mathcal{Q}|/2$.)

A transversal CNF formula is always satisfiable, and so for $F' = \emptyset$, any board configuration $\mathbb{P}$ that has a satisfiability witness of this form must in fact be satisfiable. To handle an arbitrary F', we add the requirement that $(\mathcal{Q}, \mathcal{H})$ respect F'. Finally, we can formally define the concept of satisfiability witness.

Definition A.10.5. Let $\mathbb{P}$ be a set of Boolean functions. A tuple $(F'; \mathcal{Q}, \mathcal{H}, M)$ is a *satisfiability witness* for $\mathbb{P}$ if:

1. F' is a satisfiable CNF formula.
2. $(\mathcal{Q}, \mathcal{H})$ is a structured assignment set which respects F'.
3. M is a 2-CNF formula which is a transversal of $\mathcal{Q}$.
4. $F' \wedge M \models_{(\mathcal{Q},\mathcal{H})} \mathbb{P}$.

The *size* of a satisfiability witness $(F'; \mathcal{Q}, \mathcal{H}, M)$ is $|M|$.

We single F' out since its value is fixed while $\mathcal{Q}, \mathcal{H}, M$ are dynamic and change throughout the FC refutation.

A FC refutation is composed of three kinds of steps: axiom download, inference and erasure. It turns out that the first two steps are relatively easy to handle, as long as we maintain the invariant that the size of the satisfiability witness is $O(Sp(\mathbb{P}))$. This invariant allows us to expand the witness in order to accommodate new axioms as long as the monomial space is small enough, using the extension property of extendible families.

Erasure is more difficult, since the monomial space of the configuration could shrink, and in order to maintain the invariant, we need to shrink the witness as well. This is accomplished by the following crucial lemma, which shows that if a configuration has any satisfiability witness, then we can find another satisfiability witness for the configuration whose size is bounded in terms of the monomial space of the configuration.

Because of the technical issue of multiple representations we also need to use the locality lemma in axiom download steps, but we could omit it in a proof of a space lower bound for PCR. It is however a key piece in erasure steps.

Lemma A.10.6 (Locality lemma). *Suppose $(F'; \mathcal{Q}, \mathcal{H}, M)$ is a satisfiability witness for some set of Boolean functions $\mathbb{P}$. There is another satisfiability witness $(F'; \mathcal{Q}', \mathcal{H}', M')$ for $\mathbb{P}$ such that $\mathcal{Q}' \subseteq \mathcal{Q}$, $\mathcal{H}' = \mathcal{H}\restriction_{\mathcal{Q}'}$ and $|M'| \leq 2Sp(\mathbb{P})$.*

Proof. In this proof $\mathcal{Q}[x]$ denotes the (unique) class in $\mathcal{Q}$ that contains variable x.

The starting point of the proof is understanding the relation between monomials in a defining set of monomials U of $\mathbb{P}$ and clauses in M which underlies the property $F' \wedge M \models_{(\mathcal{Q},\mathcal{H})} \mathbb{P}$. A clause $C \in M$ affects a monomial $m \in U$ whenever the two mention variables belonging to the same partition in $\mathcal{Q}$. If a clause C does not affect a monomial m, then the clause C puts no constraints on the value of m.

Formally, we construct a bipartite graph between a minimal defining set of monomials U and the set of clauses in M (which we identify with M itself). We draw an edge between $m \in U$ and $C \in M$ whenever for some $Q \in \mathcal{Q}$, both m and C mention some variable in Q.

We break U into two parts: one part which is collectively affected by a small number of clauses, and another part in which we can associate with each monomial two clauses affecting it. To this end, let U_1 be an inclusion-maximal set under the constraint $|N(U_1)| \leq 2|U_1|$, and let $U_2 = U \setminus U_1$. We partition M accordingly into $M_1 = N(U_1)$ and $M_2 = M \setminus M_1$. As a slight modification of Hall's marriage theorem shows, the maximality of U_1 implies that we can associate with each monomial in U_2 two *unique* clauses in M_2 (that is, each clause in M_2 is associated with at most one monomial). In other words, there is a double matching from U_2 to M_2. (For more details on this step, see [2, 79, 39].)

We construct the new 2-CNF M' out of two parts: $M' = M_1 \cup M_2'$. The first part M_1, taken verbatim from M, takes care of U_1. The other part M_2', which we construct from the double matching, takes care of U_2.

The 2-CNF M_2' consists of one clause C_m for every monomial $m \in U_2$. In order to define C_m, let $x^a \vee y^b$ and $z^c \vee w^d$ be the two clauses in M_2 that are matched to m in the double matching. Assume without loss of generality that $m = r^e s^f m'$, where $r \in \mathcal{Q}[x]$ and $s \in \mathcal{Q}[z]$. The clause C_m is defined as $C_m = r^e \vee s^f$.

By construction, $|M'| \leq 2|U_1| + |U_2| \leq 2|U| = 2Sp(\mathbb{P})$. Having defined M', we complete the definition of the new satisfiability witness as follows. First, let $\mathcal{Q}' = \{\mathcal{Q}[x] \mid x \in \mathit{Vars}(M')\}$; this guarantees that M' is a transversal of $\mathcal{Q}'$. Observe that $\mathcal{Q}' \subseteq Q$. Second, let $\mathcal{H}' = \mathcal{H}\restriction_{\mathcal{Q}'}$. It is easy to check that $(F'; \mathcal{Q}', \mathcal{H}', M'))$ satisfies the first three properties of a satisfiability witness. It remains to prove that $F' \wedge M' \models_{(\mathcal{Q}',\mathcal{H}')} \mathbb{P}$.

In order to show that $F' \wedge M' \models_{(\mathcal{Q}',\mathcal{H}')} \mathbb{P}$, we consider an arbitrary total assignment α extending some partial assignment in $\mathcal{H}'$ and satisfying $F' \wedge M'$. We will modify α to another total assignment β that extends some partial assignment in $\mathcal{H}$ and satisfies $F' \wedge M$, and furthermore has the property that $\beta(m) = \alpha(m)$ for every $m \in U$. By assumption, $F' \wedge M \models_{(\mathcal{Q},\mathcal{H})} \mathbb{P}$, and so $\beta(\mathbb{P}) = 0$. Since $\beta(m) = \alpha(m)$ for every $m \in U$, we conclude that $\alpha(\mathbb{P}) = 0$ as well.

We proceed to define β. For each clause $x^a \vee y^b$ in M_2, we will define β on $\mathcal{Q}[x], \mathcal{Q}[y]$ using partial assignments from $\mathcal{H}$, distinguishing two cases: the clause is matched to some monomial in U_2, or it is *unmatched*. The values of all the other variables are taken directly from α.

Suppose $m \in U_2$ is matched to the clauses $x^a \vee y^b$ and $z^c \vee w^d$ and $C_m = r^e \vee s^f$, where $\mathcal{Q}[x] = \mathcal{Q}[r]$ and $\mathcal{Q}[z] = \mathcal{Q}[s]$. (In other words, we are in exactly the same situation described above while constructing M'.) Define β on $\mathcal{Q}[x], \mathcal{Q}[y], \mathcal{Q}[z], \mathcal{Q}[w]$ using partial assignments from $\mathcal{H}$ satisfying r^e, y^b, s^f, w^d. As a result, β satisfies the clauses $x^a \vee y^b$ and $z^c \vee w^d$ and the monomial m.

For each unmatched clause $x^a \lor y^b$ in M_2, we define β on $\mathcal{Q}[x]$ and $\mathcal{Q}[y]$ using partial assignments from $\mathcal{H}$ satisfying x^a and y^b. As a result, β satisfies the clause $x^a \lor y^b$. Finally, complete the definition of β by defining $\beta(x) = \alpha(x)$ for any hitherto undefined variable x. From the construction it is clear that β extends some partial assignment in $\mathcal{H}$.

In order to complete the proof, we need to show that β satisfies $F' \land M$, and that β agrees with α on all the monomials in U. We start by showing that β satisfies $F' \land M$. By construction, β satisfies the clauses in M_2. Since β agrees with α on variables mentioned in M_1, β satisfies M_1. Finally, let $C \in F'$. Since $(\mathcal{Q}, \mathcal{H})$ respects F', either the variables in C are disjoint from $\bigcup \mathcal{Q}$, or the variables in C all belong to some $Q_i \in \mathcal{Q}$, and all assignments in the respective $H_i \in \mathcal{H}$ satisfy C. In the former case, β agrees with α on variables mentioned in C, and so β satisfies C. In the latter case, β satisfies C since β extends some partial assignment in $\mathcal{H}$.

It remains to show that $\beta(m) = \alpha(m)$ for all monomials $m \in U$. In short, this is true for monomials in U_1 since α and β agree on all the relevant variables, and for monomials in U_2 since in both assignments they are reduced to zero. We proceed to show this formally.

Suppose first that $m \in U_1$. We claim that $\alpha(v) = \beta(v)$ for all variables v mentioned in m. Indeed, if $\alpha(v) \neq \beta(v)$ then $v \in \mathcal{Q}[x]$ for some clause $C = x^a \lor y^b$ in M_2. Yet this implies that m is connected to C, contradicting the definition of M_2. We conclude that α and β agree on all variables mentioned in m, and so $\alpha(m) = \beta(m)$ in this case.

Suppose next that $m \in U_2$. We claim that $\alpha(m) = \beta(m) = 0$. Let $C_m = r^e \lor s^f$, and recall that m is of the form $m = r^e s^f m'$. Thus $\alpha(m) = 0$ since α satisfies C_m, and $\beta(m) = 0$ since it satisfies r^e and s^f by construction. $\square$

Theorem A.10.7 (restatement of Theorem A.2.6 [39]). *Let F be a CNF formula with an r-extendible family $\mathcal{F}$ with respect to some $F' \subseteq F$. Then $Sp_{\mathcal{F}C}(F \vdash \bot) \geq r/4$.*

Proof. Let $\mathcal{F}$ be an r-extendible family with respect to some satisfiable $F' \subseteq F$. Let π be a derivation from F in space $Sp(\pi) < r/4$. We will show that $1 \notin \pi$ or, even stronger, that every configuration $\mathbb{P}_t$ appearing in π is satisfiable.

We will maintain a satisfiability witness $(F'; \mathcal{Q}_t, \mathcal{H}_t, M_t)$ for every configuration $\mathbb{P}_t$. Our satisfiability witnesses will satisfy two conditions: $(\mathcal{Q}_t, \mathcal{H}_t) \in \mathcal{F}$, and the *size bound* $|M_t| \leq 2Sp(\mathbb{P}_t)$. The existence of a satisfiability witness implies that $\mathbb{P}_t$ is satisfiable. Indeed, let $\alpha \in \mathcal{H}_t$ be some partial assignment that satisfies all the literals in M_t. Since $(\mathcal{Q}_t, \mathcal{H}_t)$ respects F', each clause in F' is either already satisfied by α or is completely disjoint from the domain of α. As F' is satisfiable, we can extend α to a total assignment β which satisfies F'. Hence, from $F' \land M_t \models_{(\mathcal{Q}_t, \mathcal{H}_t)} \mathbb{P}_t$ we have that β satisfies $\mathbb{P}_t$, and so $\mathbb{P}_t$ is satisfiable.

We construct the satisfiability witnesses by induction. For $t = 0$, the satisfiability witness is $(F'; \emptyset, \emptyset, \emptyset)$. For the induction step, suppose we are given a satisfiability witness $(F'; \mathcal{Q}, \mathcal{H}, M)$ for $\mathbb{P}_t$. We will construct a satisfiability witness $(F'; \mathcal{Q}', \mathcal{H}', M')$ for $\mathbb{P}_{t+1}$.

To simplify the notation, let $\mathbb{P} = \mathbb{P}_t$ and $\mathbb{P}' = \mathbb{P}_{t+1}$. We distinguish three cases, which correspond to the three possible steps in the proof.

Axiom download. Let C be the downloaded clause, which we also view as a monomial. If $C \in F'$ or every extension α of a partial assignment in $\mathcal{H}$ satisfies C, then in particular $F' \wedge M \models_{(\mathcal{Q},\mathcal{H})} \mathbb{P} \cup \{C\} = \mathbb{P}'$, and $M' = M$, $\mathcal{Q}' = \mathcal{Q}$, $\mathcal{H}' = \mathcal{H}$ form a satisfiability witness.

Otherwise, by hypothesis $Sp(\mathbb{P}') < r/4$ and so $Sp(\mathbb{P}) < r/4 - 1$. Indeed, if U is a defining set of monomials of $\mathbb{P}$, then $U \cup \{C\}$ is a defining set of monomials of $\mathbb{P}'$. By the induction hypothesis, $|\mathcal{Q}| < r - 1$. By the extension property of extendible family, there exists a structured set of assignments $(\tilde{\mathcal{Q}}, \tilde{\mathcal{H}}) \in \mathcal{F}$ such that $|\tilde{\mathcal{Q}}| < r$, $(\mathcal{Q},\mathcal{H}) \preccurlyeq (\tilde{\mathcal{Q}}, \tilde{\mathcal{H}})$ and $\tilde{\mathcal{H}} \models C$. By assumption $\mathcal{H} \not\models C$ and so $\mathcal{Q} \neq \tilde{\mathcal{Q}}$. Let $\tilde{\mathcal{Q}} = \mathcal{Q} \cup \{Q\}$.

The assignments corresponding to Q in $\tilde{\mathcal{H}}$ will ensure that the clause C is satisfied. Since we are going to add a new clause to M', we need to come up with two new parts in $\mathcal{Q}'$, and so we repeat the process. Let D be any axiom in $F \setminus F'$ such that $\tilde{\mathcal{H}} \not\models D$; if no such axiom exists then F is satisfiable and the theorem follows vacuously. Repeat the argument above and obtain a new disjoint set Q' and a structured set of assignments $(\mathcal{Q}', \mathcal{H}') \in \mathcal{F}$.

Choose arbitrary variables $x \in Q$ and $y \in Q'$, and let $M' = M \cup \{x \vee y\}$. By construction, $(F'; \mathcal{Q}', \mathcal{H}', M')$ is a satisfiability witness for $\mathbb{P}'$.

In both cases, by Lemma A.10.6 there is another satisfiability witness $(F'; \mathcal{Q}'', \mathcal{H}'', M'')$ for $\mathbb{P}'$ satisfying the size bound and with $\mathcal{Q}'' \subseteq \mathcal{Q}'$, $\mathcal{H}'' = \mathcal{H}'\restriction_{\mathcal{Q}''}$. By the restriction property of the extendible family, we have $(\mathcal{Q}'', \mathcal{H}'') \in \mathcal{F}$.

Inference. It is enough to pick $M' = M$, $\mathcal{Q}' = \mathcal{Q}$, $\mathcal{H}' = \mathcal{H}$. The first three properties in the definition of satisfiability witness continue to hold, while the last property follows from the soundness of FC. Finally, the size bound trivially holds since $|\mathbb{P}'| \geq |\mathbb{P}|$.

Erasure. Since FC is sound, $(F'; \mathcal{Q}, \mathcal{H}, M)$ is a satisfiability witness for $\mathbb{P}'$ as well. Hence Lemma A.10.6 furnishes us with a satisfiability witness $(F'; \mathcal{Q}', \mathcal{H}', M')$ for $\mathbb{P}'$ satisfying the size bound and with $\mathcal{Q}' \subseteq \mathcal{Q}$, $\mathcal{H}' = \mathcal{H}\restriction_{\mathcal{Q}'}$. By the restriction property of extendible, $(\mathcal{Q}', \mathcal{H}') \in \mathcal{F}$.

□

Paper B

How Limited Interaction Hinders Real Communication (and What it Means for Proof and Circuit Complexity)

Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals

Full length version of the article published in Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS '16), October 2016, pp. 295–304.

Abstract

We obtain the first true size-space trade-offs for the cutting planes proof system, where the upper bounds hold for size and total space for derivations with constant-size coefficients, and the lower bounds apply to length and formula space (i.e., number of inequalities in memory) even for derivations with exponentially large coefficients. These are also the first trade-offs to hold uniformly for resolution, polynomial calculus and cutting planes, thus capturing the main methods of reasoning used in current state-of-the-art SAT solvers.

We prove our results by a reduction to communication lower bounds in a round-efficient version of the real communication model of [Krajíček '98], drawing on and extending techniques in [Raz and McKenzie '99] and [Göös et al. '15]. The communication lower bounds are in turn established by a reduction to trade-offs between cost and number of rounds in the game of [Dymond and Tompa '85] played on directed acyclic graphs.

As a by-product of the techniques developed to show these proof complexity trade-off results, we also obtain an exponential separation between $\mathsf{monotone\text{-}AC}^{i-1}$ and $\mathsf{monotone\text{-}AC}^{i}$, improving exponentially over the superpolynomial separation in [Raz and McKenzie '99]. That is, we give an explicit Boolean function that can be computed by monotone Boolean circuits of depth $\log^i n$ and polynomial size, but for which circuits of depth $O(\log^{i-1} n)$ require exponential size.

B.1 Introduction

Ever since the discovery of NP-completeness by Cook and Levin in [64, 131], the problem of how hard it is to decide satisfiability of formulas in propositional logic has played a leading role in theoretical computer science. Although the conventional wisdom is that SAT should be a very hard problem—to the extent that the *Exponential Time Hypothesis* [109] concerning its worst-case complexity is a standard assumption used in many other hardness results—essentially no non-trivial lower bounds on the time complexity of the SAT problem are known.

A less ambitious goal is to ask for lower bounds if not only the running time but also the memory usage of the algorithm is restricted. Yet it took until [84] to rule out a linear-time, logarithmic-space algorithm for SAT. Later research has shown that refuting unsatisfiable formulas on random-access machines cannot be done non-deterministically in simultaneous time $n^{4^{1/3}}$ and space $n^{o(1)}$ [72] and SAT cannot be decided deterministically in simultaneous time $n^{1.8}$ and space $n^{o(1)}$ [184]. On Turing machines, no non-deterministic algorithm solving SAT in time T and space s can achieve $T \cdot s = n^2/\log^3 n$ [166]. (See [181] for a good survey of the area with more details on this kind of results.)

For a problem that is believed to require exponential time, the results listed above might not seem very impressive. Yet they should not necessarily be viewed only as an illustration of the weaknesses of current techniques for proving lower bounds. It is important to realize that the adversary is formidable—applied research in the last 15–20 years has led to the development of amazingly efficient algorithms, so-called *SAT solvers*, that solve many real-world instances with millions of variables, and do so in linear time. Today, practitioners often think of SAT as an *easy problem to reduce to*, rather than a hard problem to reduce from (we refer the reader to [34] for more on this fascinating topic).

Virtually the only tool currently available for a rigorous analysis of the performance of such SAT solvers is *proof complexity* [66], where one studies the methods of reasoning used by the corresponding algorithms. The transcript of the computations made can be viewed as a formal proof applying the relevant method of reasoning, and proof complexity analyses the resources needed when all computational choices are made optimally (i.e., non-deterministically). Even though this is quite a challenging adversarial setting, proof complexity has nevertheless managed to give tight exponential lower bounds on the worst-case running time for many approaches for SAT used in practice by lower-bounding proof size.

The focus of this paper is on time-space trade-offs in computational models describing current state-of-the-art SAT solvers. This research is partly driven by SAT solver running time and memory usage—in practice, space consumption can be almost as much of a bottleneck as running time—but is also motivated by the fundamental importance of time and space complexity in computational complexity.

B.1.1 Previous Work on Proof Complexity Trade-offs

In *resolution* [35], which is arguably the most well-studied proof system in proof complexity, the input is an unsatisfiable formula in conjunctive normal form (CNF) and new disjunctive clauses are derived from this formula until an explicit contradiction is reached (in the form of the empty clause without literals). Resolution is also the method of reasoning underlying the currently most successful SAT solving paradigm based on so-called *conflict-driven clause learning (CDCL)* [18, 136, 139]. The question of time-space trade-offs for resolution was first raised by Ben-Sasson in 2002 (journal version in [24]), who also obtained such trade-offs for the restricted subsystem of tree-like resolution. Size-space trade-offs for general, unrestricted resolution were later shown in [143, 28, 19, 23].

In contrast to the trade-off results for random-access and Turing machines reviewed above, in these more limited models of computation one can obtain exponential lower bounds on proof size (corresponding to running time) for proofs in sublinear but polynomial space [143, 28], and results in [19, 23] even exhibit trade-offs where size has to be superpolynomial and space has to be superlinear simultaneously. Another difference is that these results are *true trade-offs* in the sense that it is actually possible to refute the formulas both in small size and small space, only not simultaneously. A third nice feature of the trade-offs are that the upper bounds are on proof size and total space, whereas the (sometimes tightly matching) lower bounds are on *length* and *formula space*, meaning that one only charges one time unit for each derivation step regardless of its complexity, and only one space unit per "formula" (for resolution: per clause) regardless of how large it is. Thus, the upper bounds are algorithmically achievable, while the lower bounds hold in a significantly stronger model.

A stronger proof system than resolution is *polynomial calculus* [62, 2], where the clauses of a formula are translated to multilinear polynomials and calculations inside the ideal generated by these polynomials (basically corresponding to a Gröbner basis computation) establishes unsatisfiability. Among other things, polynomial calculus captures CDCL solvers extended with reasoning about systems of linear equations $\bmod 2$. The first size-space trade-offs for polynomial calculus—which were not true trade-offs in the sense discussed above, however—were obtained in [106], and these results were further improved in [23] to true trade-offs essentially matching the results cited above for resolution except for a small loss in parameters.

Another proof system that is also stronger than resolution and that has been the focus of much research is *cutting planes* [68], which formalizes the integer linear programming algorithm in [91, 60] and underlies so-called *pseudo-Boolean* SAT solvers. In cutting planes the clauses of a CNF formula are translated to linear inequalities, which are then manipulated to derive a contradiction. Thus, the question of Boolean satisfiability is reduced to the geometry of polytopes over the real numbers. Cutting planes is much more poorly understood than resolution and polynomial calculus, however, and size-space trade-offs have proven elusive. The results in [106] apply not only to resolution and polynomial calculus but also to cutting planes, and were improved further in [95] to hold for even stronger proof systems, but unfortunately are not true trade-offs in the sense discussed above.

The problem is that what is shown in [106, 95] is only that proofs in small space for certain formulas have to be very large, but it is not established that these formulas can be refuted space-efficiently. In fact, for resolution it can be shown using techniques from [27] that such small-space proofs provably do not exist, and for polynomial calculus there is circumstantial evidence for a similar claim. As discussed in Section B.3, this turns out to be an inherent limitation of the technique used.

In a recent surprising paper [87], it was shown that cutting planes can refute any formula in *constant* space if we only count the number of lines or formulas. Plugging this result into [106, 95] yields a trade-off of sorts, since "small-space" proofs will always exist, but the catch is that such proofs will have exponentially large coefficients. This means that these trade-offs do not seem very "algorithmically relevant" in the sense that such proofs could hardly be found in practice, and saying that a proof with exponential-size coefficients has "constant space" somehow does not feel quite right.

B.1.2 Our Proof Complexity Contributions

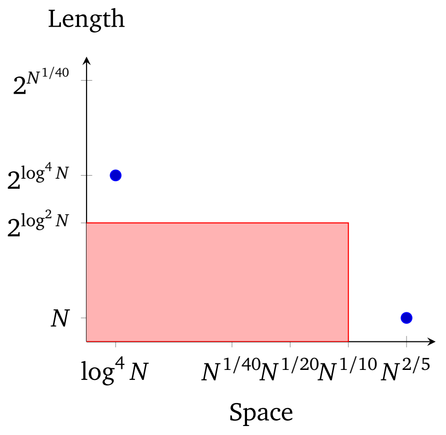
In this paper we report the first true, algorithmically realizable trade-offs for cutting planes, where the upper bounds hold for proof size and total space and the lower bounds apply to proof length and formula space (i.e., number of inequalities). The trade-offs also hold for resolution and polynomial calculus, making them the first trade-offs that hold for essentially all methods of reasoning used in the most successful SAT solvers to date.[1]

Below, we state two examples of the kind of trade-offs we obtain (referring the reader to Section B.2 for the missing formal definitions). In the rest of this section we will focus on cutting planes, since this proof system is the main target of this work. However, all the lower bounds stated also hold for polynomial calculus (and for the strictly weaker proof system resolution), and since all our upper bounds are actually proven in resolution they transfer to both polynomial calculus and cutting planes.

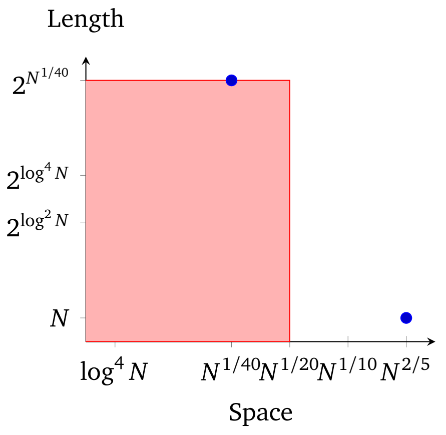
The first result is a "robust trade-off" that holds all the way from polylogarithmic to polynomial space as stated next.

Theorem B.1.1 (Informal). *There exists an explicitly constructible family of* 6*-CNF formulas* $\{F_N\}_{N=1}^{\infty}$ *of size* $\Theta(N)$ *such that:*

[1]We remark that this ignores the issue of formula *preprocessing techniques*, which are heavily used in most state-of-the-art SAT solvers, and some of which potentially require the full extended Frege proof system for a complete formal description (but can also sometimes cause a provable exponential *loss* in reasoning power). Since in practice SAT solvers fail to solve many of the combinatorial benchmark formulas that are hard for resolution, polynomial calculus, and cutting planes but easy for (even non-extended) Frege, however, and since in addition it is usually not hard to come up with formulas that foil any concrete preprocessing techniques actually used, this seems like a reasonable simplification.



(a) Robust trade-off



(b) Exponential trade-off

Figure B.1: Pictorial illustrations of trade-offs in Theorems B.1.1 and B.1.2

1. *F_N can be refuted by cutting planes with constant-size coefficients in size $\mathrm{O}(N)$ and total space $\mathrm{O}\big(N^{2/5}\big)$.*

2. *F_N can be refuted by cutting planes with constant-size coefficients in total space $\mathrm{O}(\log^4 N)$ and size $2^{\mathrm{O}(\log^4 N)}$.*

3. *Any cutting planes refutation of F_N, even with coefficients of unbounded size, in formula space less than $N^{1/10-\epsilon}$ requires length greater than $2^{\Omega(\log^2 N)}$.*

The second trade-off holds over a smaller space range, but causes an exponential and not just superpolynomial blow-up in proof size.

Theorem B.1.2 (Informal). *There exists an explicitly constructible family of 6-CNF formulas $\{F_N\}_{N=1}^{\infty}$ of size $\Theta(N)$ such that:*

1. *F_N can be refuted by cutting planes with constant-size coefficients in size $\mathrm{O}(N)$ and total space $\mathrm{O}\big(N^{2/5}\big)$.*

2. *F_N can be refuted by cutting planes with constant-size coefficients in total space $\mathrm{O}\big(N^{1/40}\big)$ and size $2^{\mathrm{O}(N^{1/40})}$.*

3. *Any cutting planes refutation of F_N, even with coefficients of unbounded size, in formula space less than $N^{1/20-\epsilon}$ requires length greater than $2^{\Omega(N^{1/40})}$.*

See Figure B.1 for an illustration of these results, where blue dots denote provable upper bounds on time-space parameters of cutting planes refutations and the shaded red areas show ranges of parameters that are impossible to achieve.

B.1.3 Previous Work in Monotone Circuit Complexity

Since this paper also makes contributions to monotone circuit complexity, we next review some relevant background in this area. After superpolynomial lower bounds on the size of monotone circuits computing explicit functions were obtained in [159, 11] (see also [46]), the first step towards the natural next goal of establishing a depth hierarchy for monotone circuits was taken in [118], proving that connectivity, which is in $\mathsf{monotone\text{-}NC}^2$, requires depth $\Omega(\log^2 n)$ for monotone circuits with fan-in 2. This implies a separation between $\mathsf{monotone\text{-}NC}^1$ and $\mathsf{monotone\text{-}NC}^2$. The same approach was used in [158] to prove a separation between $\mathsf{monotone\text{-}NC}^{i-1}$ and $\mathsf{monotone\text{-}NC}^{i}$ for every i. This result can be rephrased as saying that there is a family of Boolean functions $\{f^i\}$ such that f^i can be computed by monotone circuits of depth $\log^i n$, fan-in 2, and polynomial size but cannot be computed by any monotone circuit of depth $\mathrm{o}(\log^i n)$ and fan-in 2. This result was later extended in [114] to circuits of semi-unbounded fan-in—i.e., with AND-gates of fan-in 2 and OR-gates of unbounded fan-in.

Going into more details, the function in [158] that witnesses the separation between $\mathsf{monotone\text{-}NC}^{i-1}$ and $\mathsf{monotone\text{-}NC}^{i}$ can be computed by a monotone circuit of depth $\log^{i-1} n$, polynomial fan-in, and polynomial size, and therefore the separation is between $\mathsf{monotone\text{-}NC}^{i-1}$ and $\mathsf{monotone\text{-}AC}^{i-1}$. This immediately implies a separation between $\mathsf{monotone\text{-}AC}^{i-1}$ and $\mathsf{monotone\text{-}AC}^{i}$ as well, since $\mathsf{monotone\text{-}AC}^{i-1}$ is contained in $\mathsf{monotone\text{-}NC}^{i}$. However, this separation only guarantees a superpolynomial circuit size lower bound. Furthermore, the function f^i only depends on $\log^{40i} n$ variables and so it can be computed by a monotone DNF of size $2^{\log^{40i} n}$, i.e., there is a quasipolynomial upper bound.

We remark that it is clearly not possible to prove a superpolynomial separation between $\mathsf{monotone\text{-}NC}^{i-1}$ and $\mathsf{monotone\text{-}NC}^{i}$ in view of the simple fact that circuits in these classes have fan-in 2, and hence it only makes sense to talk about superpolynomial versus exponential separations in the $\mathsf{monotone\text{-}AC}$ hierarchy. It should be noted that exponential separations between monotone circuits of bounded depth were previously known, but only for depth less than logarithmic. It was shown in [122] that the complete tree of depth k, arity $n^{1/k}$, and size $\Theta(n)$, with alternating levels of AND and OR, requires size $2^{\Omega(n^{1/k}/k)}$ to compute with circuits of depth $k-1$. This result was later reproven in [141] using the communication complexity of the pointer jumping function (see also [157]).

B.1.4 Our Monotone Circuit Complexity Contributions

In this paper, we establish an exponential separation in the $\mathsf{monotone\text{-}AC}$ hierarchy. More precisely, for each $i \in \mathbb{N}$ we exhibit a Boolean function f^i that can be computed by monotone circuits of depth $\log^i n$ but such that every monotone circuit of depth at most $\mathrm{O}(\log^{i-1} n)$ requires size $2^{n^{\Omega(1)}}$ (where the hidden constant in the lower bound depends inversely on that in the upper bound).

Theorem B.1.3. *For every $i \in \mathbb{N}$ there is a Boolean function over n variables that can be computed by a monotone circuit of depth $\log^i n$, fan-in $n^{4/5}$, and size $\mathrm{O}(n)$, but for which every monotone circuit of depth $q \log^{i-1} n$ requires size $2^{\Omega(n^{1/(10+4\epsilon)q})}$.*

B.1.5 Discussion of Techniques

Let us now briefly discuss the techniques we use to establish the above results, focusing for concreteness on Theorems B.1.1 and B.1.2. These theorems are proven by a careful chain of reductions as follows.

1. Our first step is to use the connection made explicit in [106], and also used in [95], that short and space-efficient proofs for a CNF formula F can be converted to efficient communication protocols for the *falsified clause search problem* for F. Going beyond [106, 95], however, we make the simple but absolutely crucial additional observation that protocols obtained in this way are also round-efficient. Furthermore, in contrast to [106, 95] we do not study randomized communication, but instead focus on the *real communication model* introduced by Krajíček [125] with the purpose of getting a tighter correspondence with cutting planes.

2. We next generalize the communication-to-decision-tree simulation theorem for *composed search problem* in the celebrated paper by Göös et al. [96] to the real communication model, and then extend it further to be able to handle rounds using the *parallel decision trees* introduced by Valiant [178]. This part is inspired by [43], where the simulation theorem in the precursor [158] of [96] was proven for real communication but without taking round efficiency into account.

3. To leverage this machinery we need a base CNF search problem, and just as in [28, 95, 106, 23] (and many other papers) the *pebbling formulas Peb_G* from [29] turn out to be handy here, provided that they are defined over appropriately chosen directed acyclic graphs G. These formulas are then *lifted* (corresponding to composition of search problems) as described in [20], though the parameters of the lifting are different (and unfortunately significantly worse than in [106]).

4. The following step is the relatively straightforward observation that efficient parallel decision trees for formulas Peb_G yield good strategies in the pebble game of Dymond and Tompa [74] played on the underlying graph G. At the same time, this is a somewhat unexpected twist, since in previous papers such as [27, 28, 23] size and space lower bounds for pebbling formulas always followed from the *black-white pebble game* [67] on G, but we cannot make use of that latter game here.

5. Since we have to use the Dymond–Tompa game rather than the black-white pebble game, as a consequence we also have to use different graphs than in [28, 106, 23]—in particular, modifying the construction of graphs with good black-white pebbling

trade-offs in [130]—and as a concluding step we prove Dymond–Tompa trade-offs for these graphs.

Putting all these pieces together, we obtain a general theorem saying that graphs with Dymond–Tompa trade-offs yield explicit 6-CNF formulas with size-space trade-offs for cutting planes (and polynomial calculus and resolution). Theorem B.1.3 follows by a similar chain of reductions.

B.1.6 Paper Outline

The rest of this paper is organized as follows. In Section B.2 we give a more detailed overview of the steps in the proofs of our main theorems, introducing formal definitions of the concepts discussed above as need arises. In Section B.3 we translate proofs into communication protocols. The heart of the paper is then in Section B.4, where we establish that communication protocols for lifted search problems can be simulated by decision trees for the original search problems. In Section B.5 we show how decision trees for our search problem for pebbling formulas can be converted to Dymond–Tompa game strategies for the corresponding graphs, and in Section B.6 we show Dymond–Tompa trade-offs. After having established the upper bounds needed for our proof complexity trade-offs in Section B.7, we put all the pieces together in Section B.8. Section B.9 then discusses how we can use the same tools to obtain circuit complexity separations. Finally, we make some concluding remarks in Section B.10.

B.2 Preliminaries and Proof Overview

In this section, we describe which components are needed for our results stated in Section B.1 and how they fit together. Our goal is to give an accessible high-level outline of the proofs, but still make clear what are the main technical points in the arguments and also indicate some of the challenges that have to be overcome.

Let us start by reviewing the concepts we need from proof complexity. Throughout this paper all logarithms are to base 2 unless otherwise specified, and we write $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

B.2.1 Proof Complexity Basics and Cutting Planes

For x a Boolean variable, a *literal over* x is either the variable x itself or its negation, denoted $\overline{x}$. It will also be convenient to use the notation $x^1 = x$ and $x^0 = \overline{x}$. A *clause* $C = a_1 \vee \cdots \vee a_k$ is a disjunction of literals and a *CNF formula* $F = C_1 \wedge \cdots \wedge C_m$ is a conjunction of clauses. We will think of clauses and CNF formulas as sets, so that the ordering is inconsequential and there are no repetitions. A *k-CNF formula* is a CNF formula consisting of clauses containing at most k literals.

We write α, β to denote truth value assignments, i.e., functions to $\{0, 1\}$, where we identify 0 with false and 1 with true (thus, x^b is the literal satisfied by setting $x = b$).

We have the usual semantics that a clause is true under α, or *satisfied* by α, if at least one literal in it is true, and a CNF formula is satisfied if all clauses in it are satisfied. We write $\perp$ to denote the empty clause without literals, which is false under all truth value assignments.

Following [2, 77], we view a proof of unsatisifiability of a CNF formula F, or *refutation* of F, as a non-deterministic computation, with a special read-only input tape from which the clauses of the formula F being refuted (which we refer to as *axioms*) can be downloaded and a working memory where all derivation steps are made. In a *cutting planes (CP)* derivation, memory configurations are sets of linear inequalities $\sum_j a_j x_j \geq c$ with $a_j, c \in \mathbb{Z}$. We translate clauses C to linear inequalities $L(C)$ by identifying the clause $\bigvee_j x_j^{b_j}$ with the inequality $\sum_j (-1)^{1-b_j} x_j \geq 1 - \sum_j (1 - b_j)$. A CP refutation of F is a sequence of configurations $(\mathbb{L}_0, \ldots, \mathbb{L}_\tau)$ such that $\mathbb{L}_0 = \emptyset$, the inequality $0 \geq 1$ occurs in $\mathbb{L}_\tau$, and for $t \in [\tau]$ we obtain $\mathbb{L}_t$ from $\mathbb{L}_{t-1}$ by one of the following rules:

Axiom download $\mathbb{L}_t = \mathbb{L}_{t-1} \cup \{L\}$ for L being either the encoding $L(C)$ of an *axiom clause* $C \in F$ or a *variable axiom* $x_j \geq 0$ or $-x_j \geq -1$ for any variable x_j.

Inference $\mathbb{L}_t = \mathbb{L}_{t-1} \cup \{L\}$ for L inferred by *addition* $\dfrac{\sum_j a_j x_j \geq c \qquad \sum_j b_j x_j \geq d}{\sum_j (a_j + b_j) x_j \geq c + d}$,

multiplication $\dfrac{\sum_j a_j x_j \geq c}{\sum_j k a_j x_j \geq kc}$, or *division* $\dfrac{\sum_j k a_j x_j \geq c}{\sum_j a_j x_j \geq \lceil c/k \rceil}$ for $k \in \mathbb{N}^+$.

Erasure $\mathbb{L}_t = \mathbb{L}_{t-1} \setminus \{L\}$ for some $L \in \mathbb{L}_{t-1}$.

The *length* of a CP refutation is the number of linear inequalities L appearing in download and inference steps, counted with repetitions. We obtain the *size* of a refutation by also summing the sizes of the coefficients and constant terms in the inequalities, i.e., each inequality $\sum_j a_j x_j \geq c$ contributes $\log|c| + \sum_j \log|a_j|$. The *formula space* of a configuration $\mathbb{L} = \{\sum_j a_{i,j} x_{i,j} \geq c_i \mid i \in [s]\}$ is the number of inequalities s in it, and the *total space* of $\mathbb{L}$ is $\sum_{i \in [s]} \bigl(\log|c_i| + \sum_j \log|a_{i,j}|\bigr)$. We obtain the formula space or total space of a refutation by taking the maximum over all configurations in it. Finally, the length, size, formula space, and total space of refuting a formula F is obtained by taking the minimum over all CP refutations of the formula with respect to the corresponding complexity measure.

B.2.2 Composed Search Problems and Lifted CNF Formulas

Informally speaking, the idea behind *lifting*, or *composition*, is to take a relation over some domain and extend it to tuples from the same domain by combining it with a *selector* function that determines on which coordinates from the tuples the relation should be evaluated.

Let S be any relation on the Cartesian product $A \times Q$. We will think of S as a *search problem* with input domain A and output range Q, where on any input $a \in A$ the task is to find some $q \in Q$ such that $(a,q) \in S$ (assuming that S is such that there always exists at least one solution). Throughout this paper, we will have $A = \{0,1\}^m$ for some $m \in \mathbb{N}^+$, so for simplicity we fix A to be such a domain from now on.

For any $\ell \in \mathbb{N}^+$, we define the *lift of length ℓ of S* to be a new search problem $\mathit{Lift}_\ell(S) \subseteq \left([\ell]^m \times \{0,1\}^{m\cdot\ell}\right) \times Q$ with input domain $[\ell]^m \times \{0,1\}^{m\cdot\ell}$ and output range Q such that for any $x \in [\ell]^m$, any bit-vector $\{y_{i,j}\}_{i\in[m],j\in[\ell]}$, and any $q \in Q$, it holds that

$$(x,y,q) \in \mathit{Lift}_\ell(S) \quad \text{if and only if} \quad \left((y_{1,x_1}, y_{2,x_2}, \ldots, y_{m,x_m}), q\right) \in S \ . \tag{B.2.1}$$

In what follows, we will refer to the coordinates of the x-vector as *selector variables* and those of the y-vector as *main variables*, and we will sometimes use the notation

$$\mathsf{select}(x_i, y_i) = y_{i,x_i} \tag{B.2.2}$$

to denote the bit in y_i selected by x_i. We extend this notation to vectors to write $\mathsf{select}(x,y) = y_x = (y_{1,x_1}, \ldots, y_{m,x_m})$.

As in [106, 95], we obtain our results by studying lifted search problems defined in terms of CNF formulas and proving communication lower bounds for such problems. Syntactically speaking, however, these objects are not themselves CNF formulas, which is what we use to feed to our proof system. Therefore, we need an additional step which translates the lifted search problems back to CNF as follows.

Definition B.2.1 (Lifted formula [20]). Given $\ell \in \mathbb{N}^+$ and a CNF formula F over variables $u_1, \ldots, u_n$, the *lift of length ℓ of F*, denoted $\mathit{Lift}_\ell(F)$, is the formula over variables $\{x_{i,j}\}_{i\in[n],j\in[\ell]}$ (*selector variables*) and $\{y_{i,j}\}_{i\in[n],j\in[\ell]}$ (*main variables*) containing the following clauses:

- For every $i \in [n]$, an *auxiliary clause*

$$x_{i,1} \lor x_{i,2} \lor \cdots \lor x_{i,\ell} \ . \tag{B.2.3a}$$

- For every clause $C \in F$, where $C = u_{i_1} \lor \cdots \lor u_{i_s} \lor \overline{u}_{i_{s+1}} \lor \cdots \lor \overline{u}_{i_t}$ for some variable indices $i_1, \ldots, i_t \in [n]$, and for every tuple $(j_1, \ldots, j_t) \in [\ell]^t$, a *main clause*

$$\overline{x}_{i_1,j_1} \lor y_{i_1,j_1} \lor \cdots \lor \overline{x}_{i_s,j_s} \lor y_{i_s,j_s} \lor \overline{x}_{i_{s+1},j_{s+1}} \lor \overline{y}_{i_{s+1},j_{s+1}} \lor \cdots \lor \overline{x}_{i_t,j_t} \lor \overline{y}_{i_t,j_t} \ . \tag{B.2.3b}$$

Intuitively, we can think of the selector variables as encoding the vector $x \in [\ell]^m$ in the lifted search problem (B.2.1). Since $\overline{x}_{i,j} \lor y_{i,j}$ is equivalent to the implication $x_{i,j} \rightarrow y_{i,j}$, we can rewrite (B.2.3b) as

$$(x_{i_1,j_1} \rightarrow y_{i_1,j_1}) \lor \cdots \lor (x_{i_t,j_t} \rightarrow \overline{y}_{i_t,j_t}) \ , \tag{B.2.4}$$

from which we can see that for every clause C the auxiliary clauses encode that there is at least one choice of selector variables $x_{i,j}$ which are all true, and for this choice of selector variables the $y_{i,j}$-variables in the lifted main clause will play the role of the u_i-variables, giving us back the original clause C. It is easily verified that F is unsatisfiable if and only if $H = \mathit{Lift}_\ell(F)$ is unsatisfiable, and that if F is a k-CNF formula with m clauses, then H is a $\max(2k, \ell)$-CNF formula with at most $m\ell^k + n$ clauses. A small technical issue for us compared to [106, 95] is that $\ell \gg k$ will not be constant, but we can convert the wide clauses in (B.2.3a) to constant width using extension variables, and so we will just ignore this issue in our proof overview.

B.2.3 Pebbling Contradictions

An important role in many proof complexity trade-off results is played by so-called *pebbling contradictions*. For our purposes it suffices to say that they are defined in terms of directed acyclic graphs (DAGs) G, where for simplicity we assume that all vertices have indegree 0 or 2. We refer to vertices with indegree 0 as *sources* and assume that there is a unique *sink* vertex with outdegree 0. What the pebbling contradiction over G says is that the sources are true and that truth propagates from predecessors to successors, but that the sink is false. The formal definition follows.

Definition B.2.2 (Pebbling contradiction [29]). Let G be a DAG with sources S and a unique sink z, and with all non-sources having indegree 2. Then the *pebbling contradiction* over G, denoted Peb_G, is the conjunction of the following clauses over variables $\{v \mid v \in V(G)\}$:

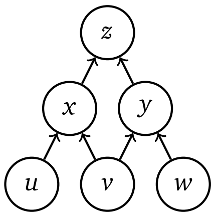
- for every source vertex $s \in S$, a unit clause s (*source axioms*),
- For all non-sources w with immediate predecessors u, v, a clause $\overline{u} \vee \overline{v} \vee w$ (*pebbling axioms*),
- for the sink z, the unit clause $\overline{z}$ (*sink axiom*).

If G has n vertices, the formula Peb_G is an unsatisfiable 3-CNF formula with $n+1$ clauses over n variables. For an example of a pebbling contradiction, see the CNF formula in Figure B.2b defined in terms of the graph in Figure B.2a.

To make the connection back to Definition B.2.1, in Figure B.3 we present the lift of length 2 of the CNF formula in Figure B.2b, with the auxiliary clauses at the top of the left column followed by the main clauses one by one, listed for all tuples of selector indices (with the only difference that since the variables in this formula are u, v, w, x, y, z rather than $u_1, \ldots, u_n$, we denote the main variables by $y_{u,j_1}, y_{v,j_2}, y_{w,j_3}$, et cetera, rather than $y_{i_1,j_1}, y_{i_2,j_2}, \ldots$). We will refer to the main clauses in Figure B.2b as source axioms, pebbling axioms, and sink axioms, respectively, when they have been obtained by lifting of the correspondingly named axioms in the pebbling contradiction.

B.2.4 Real Communication and Falsified Clause Search Problems

For our communication complexity results we study a two-player communication model, referring to the players as *Alice* and *Bob* following tradition. We first briefly discuss the



(a) Pyramid graph Π_2 of height 2.

$$\begin{aligned} & u \\ \wedge\ & v \\ \wedge\ & w \\ \wedge\ & (\overline{u} \vee \overline{v} \vee x) \\ \wedge\ & (\overline{v} \vee \overline{w} \vee y) \\ \wedge\ & (\overline{x} \vee \overline{y} \vee z) \\ \wedge\ & \overline{z} \end{aligned}$$

(b) Pebbling contradiction Peb_{Π_2}.

Figure B.2: Example pebbling contradiction for the pyramid of height 2.

basic deterministic model, and then explain how we need to extend it, directing the reader to [128] for any omitted standard communication complexity facts.

In the communication problem of computing a function $f : X \times Y \to Q$, Alice is given an input $x \in X$, Bob is given an input $y \in Y$, and they are required to find $f(x, y)$ while minimizing the communication between them. A communication protocol is a binary tree where Alice and Bob start at the root, every node specifies who is going to speak, the value of the spoken bit is only a function of the node v and the input x if Alice speaks or y if Bob does, and leaves are labelled by correct values $f(x, y)$. Similarly, for any relation $S \subseteq X \times Y \times Q$, the communication problem for S is one in which Alice is given $x \in X$, Bob is given $y \in Y$, and they are required to communicate to find some q such that $(x, y, q) \in S$. The cost of a protocol is the maximum number of bits communicated on any input, and the number of rounds is the maximum number of alternations between Alice and Bob speaking.

In order to obtain trade-offs for cutting planes, we need to study the more general *real communication* model in [125], where Alice and Bob interact via a referee, and also introduce the concept of rounds in this model. It is convenient to describe the protocol as a (non-binary) tree, where at node v in the protocol tree Alice and Bob send k_v real numbers $\phi_{v,1}(x), \ldots, \phi_{v,k_v}(x)$ and $\psi_{v,1}(y), \ldots, \psi_{v,k_v}(y)$, respectively, to the referee. The referee announces the results of the comparisons $\phi_{v,i}(x) \leq \psi_{v,i}(y)$ for $i \in [k_v]$ as a k_v-bit binary string, after which the players move to the corresponding next node in the protocol tree. The number of rounds r of a protocol is the depth of the tree and the cost c is the total number of comparisons made by the referee for any input. It is easy to see that this model can simulate standard deterministic communication (for instance, if Alice wants to send a message, she sends the complement of that message to the referee and Bob sends a list of the same length with all entries $1/2$) and is in fact strictly stronger (since equality can be solved with just two bits of communication).

The communication problem that we are interested in is the *(falsified) clause search problem*. This is the problem of, given an unsatisfiable CNF formula F and a truth value

$$
\begin{aligned}
&(x_{u,1} \lor x_{u,2}) \\
\land\ &(x_{v,1} \lor x_{v,2}) \\
\land\ &(x_{w,1} \lor x_{w,2}) \\
\land\ &(x_{x,1} \lor x_{x,2}) \\
\land\ &(x_{y,1} \lor x_{y,2}) \\
\land\ &(x_{z,1} \lor x_{z,2}) \\
\land\ &(\overline{x}_{u,1} \lor y_{u,1}) \\
\land\ &(\overline{x}_{u,2} \lor y_{u,2}) \\
\land\ &(\overline{x}_{v,1} \lor y_{v,1}) \\
\land\ &(\overline{x}_{v,2} \lor y_{v,2}) \\
\land\ &(\overline{x}_{w,1} \lor y_{w,1}) \\
\land\ &(\overline{x}_{w,2} \lor y_{w,2}) \\
\land\ &(\overline{x}_{u,1} \lor \overline{y}_{u,1} \lor \overline{x}_{v,1} \lor \overline{y}_{v,1} \lor \overline{x}_{x,1} \lor y_{x,1}) \\
\land\ &(\overline{x}_{u,1} \lor \overline{y}_{u,1} \lor \overline{x}_{v,1} \lor \overline{y}_{v,1} \lor \overline{x}_{x,2} \lor y_{x,2}) \\
\land\ &(\overline{x}_{u,1} \lor \overline{y}_{u,1} \lor \overline{x}_{v,2} \lor \overline{y}_{v,2} \lor \overline{x}_{x,1} \lor y_{x,1}) \\
\land\ &(\overline{x}_{u,1} \lor \overline{y}_{u,1} \lor \overline{x}_{v,2} \lor \overline{y}_{v,2} \lor \overline{x}_{x,2} \lor y_{x,2}) \\
\land\ &(\overline{x}_{u,2} \lor \overline{y}_{u,2} \lor \overline{x}_{v,1} \lor \overline{y}_{v,1} \lor \overline{x}_{x,1} \lor y_{x,1}) \\
\land\ &(\overline{x}_{u,2} \lor \overline{y}_{u,2} \lor \overline{x}_{v,1} \lor \overline{y}_{v,1} \lor \overline{x}_{x,2} \lor y_{x,2}) \\
\land\ &(\overline{x}_{u,2} \lor \overline{y}_{u,2} \lor \overline{x}_{v,2} \lor \overline{y}_{v,2} \lor \overline{x}_{x,1} \lor y_{x,1}) \\
\land\ &(\overline{x}_{u,2} \lor \overline{y}_{u,2} \lor \overline{x}_{v,2} \lor \overline{y}_{v,2} \lor \overline{x}_{x,2} \lor y_{x,2}) \\
\land\ &(\overline{x}_{v,1} \lor \overline{y}_{v,1} \lor \overline{x}_{w,1} \lor \overline{y}_{w,1} \lor \overline{x}_{y,1} \lor y_{y,1}) \\
\land\ &(\overline{x}_{v,1} \lor \overline{y}_{v,1} \lor \overline{x}_{w,1} \lor \overline{y}_{w,1} \lor \overline{x}_{y,2} \lor y_{y,2}) \\
\land\ &(\overline{x}_{v,1} \lor \overline{y}_{v,1} \lor \overline{x}_{w,2} \lor \overline{y}_{w,2} \lor \overline{x}_{y,1} \lor y_{y,1}) \\
\land\ &(\overline{x}_{v,1} \lor \overline{y}_{v,1} \lor \overline{x}_{w,2} \lor \overline{y}_{w,2} \lor \overline{x}_{y,2} \lor y_{y,2}) \\
\land\ &(\overline{x}_{v,2} \lor \overline{y}_{v,2} \lor \overline{x}_{w,1} \lor \overline{y}_{w,1} \lor \overline{x}_{y,1} \lor y_{y,1}) \\
\land\ &(\overline{x}_{v,2} \lor \overline{y}_{v,2} \lor \overline{x}_{w,1} \lor \overline{y}_{w,1} \lor \overline{x}_{y,2} \lor y_{y,2}) \\
\land\ &(\overline{x}_{v,2} \lor \overline{y}_{v,2} \lor \overline{x}_{w,2} \lor \overline{y}_{w,2} \lor \overline{x}_{y,1} \lor y_{y,1}) \\
\land\ &(\overline{x}_{v,2} \lor \overline{y}_{v,2} \lor \overline{x}_{w,2} \lor \overline{y}_{w,2} \lor \overline{x}_{y,2} \lor y_{y,2}) \\
\land\ &(\overline{x}_{x,1} \lor \overline{y}_{x,1} \lor \overline{x}_{y,1} \lor \overline{y}_{y,1} \lor \overline{x}_{z,1} \lor y_{z,1}) \\
\land\ &(\overline{x}_{x,1} \lor \overline{y}_{x,1} \lor \overline{x}_{y,1} \lor \overline{y}_{y,1} \lor \overline{x}_{z,2} \lor y_{z,2}) \\
\land\ &(\overline{x}_{x,1} \lor \overline{y}_{x,1} \lor \overline{x}_{y,2} \lor \overline{y}_{y,2} \lor \overline{x}_{z,1} \lor y_{z,1}) \\
\land\ &(\overline{x}_{x,1} \lor \overline{y}_{x,1} \lor \overline{x}_{y,2} \lor \overline{y}_{y,2} \lor \overline{x}_{z,2} \lor y_{z,2}) \\
\land\ &(\overline{x}_{x,2} \lor \overline{y}_{x,2} \lor \overline{x}_{y,1} \lor \overline{y}_{y,1} \lor \overline{x}_{z,1} \lor y_{z,1}) \\
\land\ &(\overline{x}_{x,2} \lor \overline{y}_{x,2} \lor \overline{x}_{y,1} \lor \overline{y}_{y,1} \lor \overline{x}_{z,2} \lor y_{z,2}) \\
\land\ &(\overline{x}_{x,2} \lor \overline{y}_{x,2} \lor \overline{x}_{y,2} \lor \overline{y}_{y,2} \lor \overline{x}_{z,1} \lor y_{z,1}) \\
\land\ &(\overline{x}_{x,2} \lor \overline{y}_{x,2} \lor \overline{x}_{y,2} \lor \overline{y}_{y,2} \lor \overline{x}_{z,2} \lor y_{z,2}) \\
\land\ &(\overline{x}_{z,1} \lor \overline{y}_{z,1}) \\
\land\ &(\overline{x}_{z,2} \lor \overline{y}_{z,2})
\end{aligned}
$$

Figure B.3: Lifted formula $\mathit{Lift}_2\big(\mathit{Peb}_{\Pi_2}\big)$ of length 2 obtained from the pebbling contradiction over Π_2.

assignment α, finding a clause $C \in F$ falsified by α. We denote this problem by *Search*(F). In fact, from a communication complexity point of view we will be interested in lifts of this search problem *Lift*$($*Search*$(F))$, while for our proof complexity trade-offs the perspective is slightly different in that we need to study the CNF formula *Lift*(F) from Definition B.2.1 and relate the hardness of this formula to the communication complexity of the falsified clause search problem *Search*$($*Lift*$(F))$. Happily, this distinction does not really matter to us, since a good communication protocol for *Search*$($*Lift*$(F))$ can also be used to solve *Lift*$($*Search*$(F))$, and hence a lower bound for the latter communication problem applies also to the former, as stated formally in the next observation.

Observation B.2.3. *Suppose that F is an unsatisfiable CNF formula. Then any two-player real communication protocol for* $\mathit{Search}(\mathit{Lift}_\ell(F))$ *where all selector variables $x_{i,j}$ in the* *same block are given to the same player can be adapted to a protocol for $Lift_\ell(Search(F))$ with the same parameters.*

We refer to, e.g., [106] for the easy proof (which is independent of the concrete communication model under consideration). Thanks to this observation, we can freely switch perspectives between $Lift_\ell(Search(F))$ and $Search(Lift_\ell(F))$ when we want to prove lower bounds for the latter problem. The reason that such lower bounds are interesting, in turn, is that if a CNF formula H has a CP refutation in short length and small space, then such a proof can be used to construct a round- and communication-efficient protocol for $Search(H)$.

Lemma B.2.4. *If a CNF formula H can be refuted by cutting planes in length L and formula space s, then for any partition of the variables of H between Alice and Bob there is a real communication protocol solving $Search(H)$ in $\lceil \log L \rceil$ rounds with total communication cost at most $s \cdot \lceil \log L \rceil$.*

Sketching the proof very briefly, given a truth value assignment α Alice and Bob can do binary search over the refutation $(\mathbb{L}_0 = \emptyset, \mathbb{L}_1, \ldots, \mathbb{L}_L)$ of H until they find a $t \in [L]$ such that $\mathbb{L}_t$ evaluates to true under α but $\mathbb{L}_{t-1}$ evaluates to false. Then the derivation step at time t must be a download of an axiom $C \in H$ falsified by α. For the details we can reuse the proof from [106] verbatim, just adding the one simple but absolutely crucial observation that the protocol obtained in this way is also round-efficient, since all communication needed to evaluate a particular configuration $\mathbb{L}_t$ can be performed in parallel.

It is worth noting that although we state Lemma B.2.4 for cutting planes here, there is nothing that really uses the syntactic properties of the cutting planes refutation. Thus, the proof works equally well for resolution, polynomial calculus, or any proof system for which configurations can be evaluated by round-efficient protocols where the communication scales as the space of the configuration.

B.2.5 Simulations of Protocols by Parallel Decision Trees

A *parallel decision tree* [178] for a search problem $S \subseteq \{0,1\}^m \times Q$ is a tree T such that each node v is labelled by a set of variables V_v and has exactly one outgoing edge for each of the $2^{|V_v|}$ possible assignments to these variables, and such that for every $\alpha \in \{0,1\}^m$ the path from the root of T defined by the edges consistent with α ends at a leaf labelled by some $q \in Q$ such that $(\alpha, q) \in S$ (where again the tacit assumption is that S is such that such a solution always exists). The *number of queries* of T is the maximal sum of set sizes $|V_v|$ along any path in T, and the *depth* of T is the length of a longest path.

Any decision tree T for a search problem S can be simulated by a communication protocol for the lifted problem $Lift(S)$ in a straightforward way, where if T wants to query the ith variable Alice and Bob can communicate to find y_{i,x_i} and then walk in T according to this value. Such a walk will end in a leaf labelled by a q such that

$\big((y_{1,x_1}, y_{2,x_2}, \dots, y_{m,x_m}), q\big) \in S$, i.e., a solution to the lifted search problem, and thus the query complexity of the original search problem provides an upper bound on the communication cost of the lifted problem. If in addition there is a parallel decision tree with small depth, then a protocol simulating such a tree will also be round-efficient. The main technical result of our paper is that simulating such a parallel decision tree is essentially the best any round-efficient protocol can do (provided that the lifting of the search problem is done with appropriate parameters).

Theorem B.2.5 (Simulation theorem). *Let S be a relation with domain $\{0,1\}^m$ and let $\ell = m^{3+\epsilon}$ for some constant $\epsilon > 0$. If there is an r-round real communication protocol in cost c that solves $Lift_\ell(S)$, then there is a parallel decision tree in depth r solving S using $O(c/\log \ell)$ queries.*

We remark that similar simulation theorems have previously been shown for both deterministic communication [158, 96] and real communication [43], but unfortunately they fail to take round efficiency into account. Our proof of Theorem B.2.5 follows the approach in these papers to build a decision tree for the original problem that simulates the communication protocol for the lifted problem. In order to obtain an efficient simulation we have to maintain (in an amortized sense) that the decision tree queries a variable only when a noticeable amount of communication has taken place. To prove that the decision tree constructed in this way is correct, we need to show that at the end of the simulation there exists a pair of inputs to Alice and Bob that are compatible both with the transcript and with a lift of the original input. Towards this end, during the simulation we maintain a set of such compatible inputs, which must not be allowed to shrink too fast.

In order for the proof to work we need to be able to handle two kinds of events: *communication events*, where we simulate the players communicating; and *query events*, where the decision tree under construction queries some variable and gets its actual value. Both of these events force us to prune the set of compatible communication inputs. In the first case we want to choose a communication message that removes as few inputs as possible, whereas in the second case we have to restrict the communication inputs to a subset that is compatible with the value returned by the decision tree query. We make sure to query a variable only when the transcript "reveals too much information" about Alice's and Bob's lifted input related to that variable, and thanks to this we can argue that query events do not happen too often and that the amount of communication provides an upper bound on the total number of queries.

Extending these techniques to round-efficient protocols and simulations by parallel decision trees causes significant additional complications, however. Very briefly, one issue is that we cannot let the tree query an individual variable as soon as sufficient information has been "revealed" about it during the simulation, but have to wait until we can issue a whole set of queries corresponding to a single message of the protocol. This makes it challenging to maintain a set of compatible inputs for variables we have not yet been allowed to query. Another issue is that, in contrast to deterministic communication protocols, real protocols do not partition the input domain into combinatorial rectangles. While this is not a big problem for a single comparison by the referee, it becomes more challenging when we want to handle a round consisting of many simultaneous comparisons.

B.2.6 From Decision Trees to Dymond–Tompa Trade-offs

The *Dymond–Tompa game* [74][2] is played in rounds on a DAG G by two players *Pebbler* and *Challenger*. In the first round, Pebbler places pebbles on a non-empty subset of vertices of G including the unique sink z and Challenger picks some vertex in this set. In all subsequent rounds, Pebbler places pebbles on some non-empty subset of vertices not yet containing pebbles, and Challenger either challenges a vertex in this new set (*jumps*) or re-challenges the previously chosen vertex (*stays*). This repeats until at the end of a round Challenger is standing on a vertex with all immediate predecessors pebbled (or on a source, for which the condition vacuously holds), at which point the game ends. Intuitively, Challenger is challenging Pebbler to "catch me if you can" and wants to play for as many rounds as possible, whereas Pebbler wants to "surround" Challenger as quickly as possible. We say that Pebbler *wins the r-round Dymond–Tompa game on G in cost c* if there is a strategy such that Pebbler can always finish the game in at most r rounds placing a total of at most c pebbles regardless of how Challenger plays.

In order to obtain lower bounds on the query complexity of parallel decision trees of bounded depth, we use an adversary argument and describe strategies that give as unhelpful answers as possible for variables queried by the decision trees. If we specialize this to the clause search problem for pebbling contradictions Peb_G, such adversary strategies are equivalent to Challenger strategies in the Dymond–Tompa game on G. For standard binary decision trees and the Dymond–Tompa game with unlimited number of rounds this was proven in [54],[3] and we show that this equivalence extends also to our more general setting where decision trees can issue queries in parallel and we account for the number of rounds in the Dymond–Tompa game.

Lemma B.2.6. *If there is a parallel decision tree for $Search(Peb_G)$ in depth r using at most c queries, then Pebbler has a winning strategy in the r-round Dymond–Tompa game on G in cost at most $c + 1$.*

It follows from this lemma that round-cost trade-offs for Dymond–Tompa pebbling implies depth-query trade-offs for parallel decision trees. To conclude the proof of the lower bound in our trade-off results, we need to find a family of graphs for which we can prove lower bounds for the cost of Pebbler strategies in the Dymond–Tompa game with bounded number of rounds. Towards this end, we establish that graphs that satisfy a certain connectivity property possess trade-offs between number of rounds and cost, and

[2]We give a slightly different, but essentially equivalent, description of the Dymond–Tompa game that is closer to recent papers such as [54, 56].

[3]This game on decision trees is called the *Raz–McKenzie game* in [54].

then exhibit such graphs. These graphs were inspired by graphs for which black-white pebbling trade-offs were shown in [130], but we need to make some modifications in order to obtain Dymond–Tompa trade-offs.

Lemma B.2.7. *For any $n, r \in \mathbb{N}^+$ there exists an explicitly constructible DAG $G(n, r)$ with $\mathrm{O}(rn \log n)$ vertices such that the cost of the r-round Dymond–Tompa game on $G(n, r)$ is at least $\Omega(n)$.*

The graph $G(n, r)$ is structured in $r + 1$ layers and we obtain the lemma by designing a strategy for Challenger such that as long as Pebbler does not place too many pebbles, Challenger can make sure that in the ith round the challenged pebble is above the ith layer. Hence, the game does not end within r rounds.

B.2.7 Proofs of Main Theorems

Combining all the components discussed above we can now prove the following trade-off lower bound.

Theorem B.2.8. *Let G be a DAG over m vertices such that any winning strategy for Pebbler in the r-round Dymond–Tompa game on G has cost $\Omega(c)$, and let $\epsilon > 0$ and $\ell = m^{3+\epsilon}$. Then $Lift_\ell(Peb_G)$ is a 6-CNF formula over $\Theta(m^{4+\epsilon})$ variables and $N = \Theta(m^{10+3\epsilon})$ clauses such that any cutting planes refutation of it in formula space less than $\frac{c}{r} \log N$, even with coefficients of unbounded size, requires length at least $2^{\Omega(r)}$.*

Proof. Suppose for the sake of contradiction that there is a cutting planes refutation of $Lift_\ell(Peb_G)$ in length $2^{\mathrm{o}(r)}$ and formula space less than $\frac{c}{r} \log N$. By Lemma B.2.4 this implies that there is a real communication protocol that solves $Lift_\ell(Search(Peb_G))$ in $\mathrm{o}(r)$ rounds and total cost $\mathrm{o}(c \log N)$. Using Theorem B.2.5 we obtain a parallel decision tree computing $Search(Peb_G)$ using $\mathrm{o}(c)$ queries and depth $\mathrm{o}(r)$. But if so, by Lemma B.2.6 Pebbler wins the $\mathrm{o}(r)$-round Dymond–Tompa game on G in cost $\mathrm{o}(c)$, which contradicts the assumption of the theorem. □

In order to attain our trade-off results we also need upper bounds on refutations of these formulas. Small-size upper bounds follow by essentially the same approach of lifting black pebbling upper bounds as in [28, 106], although more care is needed since our lifts are of non-constant length. For the small-space refutations, this technique does not work because the space loss due to the large lift length is larger than the upper bound we are aiming for. Luckily, we can instead prove upper bounds in the Dymond–Tompa game with unlimited rounds and then convert them into refutations in small space. Theorems B.1.1 and B.1.2 then follow from Theorem B.2.8 applied to an appropriate family of graphs that exhibit Dymond–Tompa trade-offs as in Lemma B.2.7.

The tools we have developed also allow us to prove the monotone circuit separation in Theorem B.1.3. The function that witnesses the separation is inspired by the

PYRAMID-GEN function of [158] adapted to the graphs in Lemma B.2.7. Then we translate the Dymond–Tompa trade-off into a lower bound for deterministic communication protocols with few rounds, which we then transform into a lower bound for circuits of small depth via the Karchmer–Wigderson game [118].

B.3 From Proofs to Communication Protocols

As mentioned in the preliminaries, length space trade-offs for a given proof system can be obtained via reduction to the falsified clause search problem. Exactly which communication model to consider for the search problem depends on the proof system. Given a sequential proof system $\mathcal{P}$ and a communication model $\mathcal{M}$, let $c_{\mathcal{P},\mathcal{M}}$ and $r_{\mathcal{P},\mathcal{M}}$ be the maximum cost and the maximum number of rounds, respectively, required to evaluate a line/formula of any configuration.

The idea behind the reduction is to, given a refutation as a sequence of configurations, do a binary search in this sequence in order to find two consecutive configurations such that the first is evaluated to true and the second to false. Since the proof system is sound, this false configuration must be due to an axiom download and this axiom must be falsified. Finally, observing that each line/formula of a configuration can be evaluated in parallel, we get the following lemma.

Lemma B.3.1. *Let π be a refutation in a sequential proof system $\mathcal{P}$ of a CNF formula F in length L and formula space s. Then, in any (deterministic) communication model $\mathcal{M}$ and for any partition of the variables of F between Alice and Bob there is a communication protocol that solves Search(F) in $r_{\mathcal{P},\mathcal{M}} \cdot \lceil \log L \rceil$ rounds with total communication cost at most $s \cdot c_{\mathcal{P},\mathcal{M}} \cdot \lceil \log L \rceil$.*

Proof. Suppose Alice and Bob are each given a part of an assignment α to F. Fix a $\mathcal{P}$-refutation $\pi = \{\mathbb{D}_0, \mathbb{D}_1, \ldots, \mathbb{D}_L\}$ of F as in the statement of the lemma. By definition of refutation, it must be the case that $\mathbb{D}_0 = \emptyset$ and $\bot \in \mathbb{D}_L$.

Alice and Bob consider the configuration $\mathbb{D}_{L/2}$ in the refutation at time $L/2$ and with joint efforts (involving some communication, which we will discuss shortly) evaluate the truth value of $\mathbb{D}_{L/2}$ under the assignment α. If $\mathbb{D}_{L/2}$ is true under α, they continue their search in the subderivation $\{\mathbb{D}_{L/2}, \mathbb{D}_1, \ldots, \mathbb{D}_L\}$. If $\mathbb{D}_{L/2}$ is false, the search continues in the first half of the refutation $\{\mathbb{D}_0, \mathbb{D}_1, \ldots, \mathbb{D}_{L/2}\}$ Note that $\mathbb{D}_0 = \emptyset$ is vacuously true under any assignment, and since $\bot \in \mathbb{D}_L$ this last configuration evaluates to false under any assignment. The binary search is carried out so as to maintain that the first configuration in the current subderivation under consideration evaluates to true and the last one evaluates to false under the given assignment α. Hence, after at most $\lceil \log L \rceil$ steps Alice and Bob find a $t \in [L]$ such that $\mathbb{D}_{t-1}$ is true under α but $\mathbb{D}_t$ is false. Since the proof system is sound, the derivation step to get from $\mathbb{D}_{t-1}$ to $\mathbb{D}_t$ cannot have been an inference or erasure, but must be a download of some axiom clause $C \in H$. This clause C must be false under α, and so Alice and Bob give C as their answer.

Now all that remains is to discuss how much communication is needed to evaluate a configuration in the refutation. Every line/formula in the configuration can be evaluated with cost at most $c_{\mathcal{P},\mathcal{M}}$ and in at most $r_{\mathcal{P},\mathcal{M}}$ rounds. Moreover, the $r_{\mathcal{P},\mathcal{M}}$ rounds to evaluate each line can be done in parallel by simply concatenating, at each round i, all the ith messages of the protocol for evaluating each line of the configuration. Since each configuration has at most s lines, it can be evaluated with cost at most $s \cdot c_{\mathcal{P},\mathcal{M}}$ and in at most $r_{\mathcal{P},\mathcal{M}}$ rounds. □

We note that, for randomized communication models (which we do not use in this paper, but are used for example in [106, 95]), the above theorem holds if $c_{P,M}$ and $r_{P,M}$ are defined to be the maximum cost and the maximum number of rounds, respectively, required to evaluate a line/formula of any configuration with high enough probability of success so that the union bound of the probability of error over all the evaluations of configurations is small enough.

Ideally, given a proof system P we want a communication model M such that $r_{P,M}$ and $c_{P,M}$ are constants, or at least a slow growing function. We only consider semantic versions of proof systems, where we specify the format of proof lines and use the fact that derivation rules, whichever they are, are semantically sound (as defined in [2]).

For example, if P is resolution, where lines are clauses of the form $\bigvee_j x_j^{b_j}$, then Alice and Bob can evaluate a line in two rounds and two bits of communication in the deterministic communication model.

If we consider polynomial calculus over any field $\mathbb{F}$, where lines are polynomials of the form $\sum_m \prod_j a_{m,j} x_j$ but the space measure is the number of monomials, Alice and Bob first check that the assignment is $\{0,1\}$-valued—and immediately output a falsified axiom otherwise—and then run the binary search protocol, where they can evaluate a monomial in two rounds and two bits of communication in the deterministic communication model.

For cutting planes with bounded coefficients, Alice and Bob can evaluate a line in two rounds and either $\mathrm{O}(\log N)$ bits of communication in the deterministic communication model if the bound is a polynomial in the size of the formula or $\mathrm{O}(\log L)$ bits if the bound is a polynomial in the size of the refutation.

For unrestricted cutting planes, Alice and Bob can evaluate a line in one round and one comparison in the real communication model.

The small but key difference from previous papers [106, 95] is that we explicitly consider rounds. The theorem states that if there exists a refutation in small space and small length, then there is a communication protocol that solves the falsified clause search problem not only with small communication cost, but also in few rounds.

It seems unlikely that the techniques used so far (without rounds) would yield true trade-offs; let us discuss why. For a trade-off of the form $s \cdot \log L \geq x$ to be a true trade-off there must exist a refutation in small space and another one in small length. The formulas for which trade-offs have been proven are lifted versions of pebbling formulas, which have refutations in small (linear) length, but not necessarily small space: the black-white pebbling number is a lower bound for resolution space, as proven in [28].

For the pyramid graphs studied in [106, 95], the black-white pebbling number is asymptotically equal to the Dymond–Tompa price, which in turn is an upper bound for the communication complexity, as we argued in Section B.2. Therefore, for the resolution proof system, the apparent trade-off is actually $s \cdot \log L = \Omega(s)$, giving only an uninformative length lower bound for the feasible range of space, and so the formula properties are better described by a space lower bound rather than a trade-off.

It seems plausible that the black-white pebbling number is also a lower bound for polynomial calculus space and cutting planes total space, and thus the "trade-offs" between PC-space or CP-total space and length, might also turn out to be unconditional space lower bounds.

Even if we consider other graph families, the best separation between black-white pebbling number and Dymond–Tompa price so far is logarithmic in the size of the graph [56], which still does not give meaningful results for resolution. It seems more promising to search for trade-offs in graphs where the black-white pebbling number is small but nonetheless have trade-offs in resolution, established by some means other than communication complexity.

To keep the discussion short and focused we only mention that trade-offs have been proven for stacks of superconcentrators [28] and Carlson–Savage graphs [144]. Yet in both cases the Dymond–Tompa price is too small to give meaningful trade-offs: in the first case, it is enough to note that the Dymond–Tompa price is at most the depth, and a stack of superconcentrators has small depth; for Carlson–Savage graphs, the proof is similar, but the depth argument is not enough.

To sum up, we showed that the graphs for which the previous techniques yield trade-offs are likely to have unconditional space lower bounds (but we cannot prove it), and that for graph families that may have trade-offs—and indeed we prove that this is the case for a special family of superconcentrators—the previous techniques cannot prove them.

B.4 From Real Communication to Parallel Decision Trees

We have reached the heart of the reduction. This is by far the most intriguing and also the most difficult part of the paper. The reader that first wishes to have an overview of the whole proof should skip Sections B.4.1 and B.4.2.

To prove Theorem B.2.5 we use the same high-level approach of [158, 43, 96]: we build a decision tree that simulates a protocol computing the composed search problem *Lift*(S) and it only queries a variable when, in a certain sense, the transcript reveals too much information about the index for that variable.

More precisely, we keep two sets of inputs A and B that are compatible with the communication so far and that are large enough. Additionally for A we enforce that for each coordinate i that we have not queried yet, if we fix every other coordinate to some value, there are still many choices for what to set the index x_i to.

To maintain the invariant, we need to handle two kinds of events: communication events where we guess a message and restrict A and B to the new compatible inputs, with some additional cleanup, and query events, where some coordinate i becomes too dependent on other coordinates. Since for each coordinate there are more choices for y_i than Bob can expect to communicate, we will be able to find an input for the players such that $\mathsf{select}(x_i, y_i)$ agrees with z_i, and then restrict A and B appropriately.

At the end of the protocol, if we were to query the remaining variables, we would have a pair of inputs (x, y) that are compatible with the transcript, therefore the protocol answers correctly, and such that $\mathsf{select}(x_i, y_i) = z_i$ in every coordinate, therefore the answer is also correct for the decision tree.

To argue that we do not make too many queries we keep a density function that measures how many choices we have for Alice's input over not queried coordinates. This function increases on communication, decreases after a query, and is nonnegative, which gives us a bound on the number of queries in terms of communication.

The description up to this point is common to all flavours of the simulation theorem, with or without rounds and with deterministic or real communication. The differences will surface once we try to implement it.

The first challenge we encounter is when exactly should we query a variable. If we do not have any bound on depth, then we can do that as soon as we detect that the invariant is broken and we need to restore it. Since we want to measure the effect of rounds, however, this exact approach will not work for us, because we might need to query a variable mid-message. Indeed, if Alice sends the message x_1x_2, we would first simulate sending some bits of x_1, then query z_1 and restrict the inputs to those such that $\mathsf{select}(x_1, y_1) = z_1$, keep simulating sending bits of x_1 and x_2, then query z_2 and restrict the inputs, and finish sending bits. We had to use two rounds of queries in a single round of communication.

It seems natural to delay querying the variables until the end of the message, but now we have another problem. Assume that Bob had sent that the 0-th bit of y_1 is a 0, and that Alice's message is x_1. If we guess that the message is $x_1 = 0$ but after we query z_1 we get that $z_1 = 1$, then the inputs compatible with the communication stop being compatible with the gadget select.

Our solution is to indeed delay querying the variables until the end of the message but still restrict the inputs as soon as the invariant breaks, in a way that, after fixing x_1, $\mathsf{select}(x_1, y_1)$ can take any value. During the interval of communication between the restriction and the query we keep a set C that acts as a proxy for B over the coordinates that we have not queried (and do not intend to). This is a harder task, but still feasible because only Alice speaks during this time, so we do not need to know B precisely.

The second challenge is to adapt the simulation to a real communication protocol. As opposed to deterministic protocols, the set of compatible inputs does not necessarily form a rectangle. However, as observed by [113], the result of one comparison splits the matrix of inputs in such a way that one quadrant—thus a rectangle—is monochromatic, and [43] uses this fact to decide the outcome of each comparison.

Since we want to query variables only at the end of each round we might not know what B is at the time we want to extract a rectangle from a comparison matrix, and unfortunately the proxy does not help. Therefore we need to extract rectangles from the input when we do know B, this is before we start simulating the message. We could easily extract a rectangle of size a $2^{-k} \times 2^{-k}$-fraction of the inputs, where k is the size of the message, but that would be equivalent to simulating the message in one go, which we argued does not work in the deterministic case.

Our solution is to extract a rectangle of size a $1/4 \times 2^{-k \log k}$-fraction of the inputs. Even though this is equivalent to simulating a long Bob message at once, it has the advantage that the equivalent message for Alice is very short, so we can still use the very same techniques to recover the invariants as in the deterministic case.

B.4.1 Simulation of Decision Trees by Deterministic Communication Protocols

We begin by proving the simulation theorem for the more common deterministic communication model. Throughout this section we assume that the arity of the select function is $\ell = m^{3+\epsilon}$ for some small constant $\epsilon > 0$, where m is the size of the input.

Theorem B.4.1. *If there is a deterministic communication protocol computing Lift(S) using communication c and r rounds, then there is a parallel decision tree computing S using $\mathrm{O}(c/\log\ell)$ queries and depth r.*

We mention in Section B.2 that we have to use a lift of polynomial length as opposed to constant length; this is needed for the simulation theorem to apply. In [96] and [158] the lift is of size m^{20}, while in [43] the lift is of size m^{14}, and a more careful analysis shows that $m^{4+\epsilon}$ is enough. Using a combinatorial approach to the analysis we can lower the lift length to $m^{3+\epsilon}$, but getting beyond this seems hard.

To be able to prove Theorem B.4.1 we need to introduce some notation. Let $\gamma = 1/(3+\epsilon)$, δ, λ, μ be numbers strictly between 0 and 1 such that the inequalities

$$\lambda - \gamma > \mu \tag{B.4.1a}$$

$$\mu + \delta - 1 > \gamma \tag{B.4.1b}$$

$$\gamma + \delta < 1 \tag{B.4.1c}$$

hold. Note that a solution exists iff $\gamma < 1/3$. For concreteness, we can think of $\gamma = 1/3 - 2\xi$, $\lambda = 1 - \xi$, $\mu = 2/3$, and $\delta = 2/3$ for some $\xi > 0$.

Let Π be an r-round deterministic communication protocol computing *Lift*(S) in cost c. Let X and Y be inputs to Alice and Bob. Let X^v and Y^v be the inputs compatible with node v of the protocol tree. We are going to keep two sets $A \subseteq [\ell]^m$ and $B \subseteq \{0,1\}^{\ell m}$ of inputs that are compatible with the communication so far, this is $A \subseteq X^v$ and $B \subseteq Y^v$, but have been cleaned up.

We will often be interested in the coordinates that the decision tree has not yet queried, which we denote $I \subseteq [m]$. We denote a vector with coordinates in I by $x_I = \{x_i : i \in I\}$ and, as a reminder, we denote a set of such vectors by S_I. We denote the projection of a set to I coordinates by $\pi_I(S_J) = \{x_I \in \{0,1\}^I : x \in S_J$ for some $x_{J \setminus I} \in \{0,1\}^{J \setminus I}\}$.

In order to formalize the property of having little information about a coordinate, we define $Graph_i(A_I)$ as the bipartite graph where left vertices x_i are elements of $[\ell]$, right vertices $x_{I\setminus\{i\}}$ are elements of $[\ell]^{|I|-1}$, and there is an edge between two vertices if $x_I \in A_I$. Analogously, let $Graph_i(B_I)$ be the bipartite graph where left vertices y_i are elements of $\{0,1\}^\ell$, right vertices $y_{I\setminus\{i\}}$ are elements of $\{0,1\}^{\ell(|I|-1)}$, and there is an edge between two vertices if $y_I \in B_I$. Now let $MinDeg_i(A_I)$ and $AvgDeg_i(A_I)$ be the minimum and average right degree of $Graph_i(A_I)$, both taken over the set of vertices of positive degree. We consider that we do not know too much about a coordinate i if $AvgDeg_i(A_I) \geq \ell^\lambda$. Moreover, we say A_I is *thick* if $MinDeg_i(A_I) \geq \ell^\mu$ for all $i \in I$, a property we will make sure to keep throughout the simulation. Observe that, since $|A_I|$ is the number of edges in $Graph_i(A_I)$ and $|\pi_{I\setminus\{i\}}(A_I)|$ is the number of right vertices with positive degree, the definition of average degree is equivalent to

$$AvgDeg_i(A_I) = \frac{|A_I|}{|\pi_{I\setminus\{i\}}(A_I)|} \ , \tag{B.4.2}$$

which is more convenient to work with.

A useful observation is that minimum degree (and therefore thickness) is monotone with respect to projections.

Observation B.4.2 ([158]). *$MinDeg_j(\pi_{I\setminus\{i\}}(A)) \geq MinDeg_j(\pi_I(A))$ for all $j \in I \setminus \{i\}$.*

Proof. For each index $j \in I \setminus \{i\}$, if there is an edge between $x_{I\setminus\{j\}}$ and x_j in $Graph_j(A_I)$, then there is also an edge between $x_{I\setminus\{i,j\}}$ and x_j in $Graph_j(\pi_{I\setminus\{i\}}(A_I))$. Formally,

$$N_j(x_{I\setminus\{i,j\}}) = \{x_j : x_{I\setminus\{i\}} \in \pi_{I\setminus\{i\}}(A)\} = \{x_j : \exists x_i \; x_I \in \pi_I(A)\} \tag{B.4.3}$$

$$= \bigcup_{x_i} \{x_j : x_I \in \pi_I(A)\} = \bigcup_{x_i} N_j(x_{I\setminus\{j\}}) \ , \tag{B.4.4}$$

so for any x_i with positive right degree we have

$$|N_j(x_{I\setminus\{i,j\}})| \geq |N_j(x_{I\setminus\{j\}})| \geq MinDeg_j(\pi_I(A)) \ . \qquad \square$$

Finally, we define two density measures for inputs over non-queried coordinates. For $A_I \subseteq [\ell]^{|I|}$, let $\alpha(A_I) = -\log \frac{|A_I|}{\ell^{|I|}} = |I| \log \ell - \log |A_I|$. Analogously, for $B_I \subseteq \{0,1\}^{\ell|I|}$, let $\beta(B_I) = -\log \frac{|B_I|}{2^{\ell|I|}} = |I|\ell - \log|B_I|$. We will make sure to keep these measures small, so that at any point there are many inputs compatible with the communication so far.

In fact, density even increases with projections.

Observation B.4.3 ([96]). $\alpha(\pi_{I\setminus\{i\}}(A)) = \alpha(\pi_I(A)) - \log \ell + \log AvgDeg_i(\pi_I(A))$

Proof. By definition of α, this is equivalent to $|\pi_{I\setminus\{i\}}(A)| = |\pi_I(A)|/AvgDeg_i(\pi_I(A))$, which follows from the definition of average degree (B.4.2). □

We have an auxiliary procedure `prune`, which we use to restore the thickness of A after Alice speaks, with the following properties.

Lemma B.4.4 (Thickness Lemma [158]). *If $AvgDeg_i(\pi_I(A)) \geq \ell^{\lambda}/4$ for all $i \in I$, then the return value A' of* `prune`(A, I) *satisfies*

1. *$\pi_I(A')$ is thick;*

2. *$\alpha(\pi_I(A')) \leq \alpha(\pi_I(A)) + 1$.*

To restrict the inputs on one coordinate i to a set $U \subseteq [\ell]$ we write $\rho_{i,U}(A) = \{x \in A : x_i \in U\}$, and similarly for $V \subseteq \{0,1\}^{\ell}$, we have $\rho_{i,V}(B) = \{y \in B : y_i \in V\}$. The set of b-monochromatic colourings of U is $V^b(U) = \{w \in \{0,1\}^{\ell} : \forall j \in U\ w_j = b\}$.

We have another auxiliary procedure `project`, which we use to pick the appropriate U after we make a query in order to recover the density of A with respect to the remaining coordinates, with the following properties. Note that in the description of Algorithm B.4 we employ sets C_I that act as a proxy for $\pi_I(B)$. We denote ${C_{I\setminus\{i\}}}^{(b)}(U) = \pi_{I\setminus\{i\}}(\rho_{i,V^b(U)}(C_I))$.

Lemma B.4.5 (Projection Lemma). *If $\pi_I(A)$ is thick, and $\beta(C_I) \leq 2\ell^{\gamma}\log^2 \ell$, then the return value U of* `project`(A, C_I, I, i) *satisfies*

1. *$\pi_{I\setminus\{i\}}(\rho_{i,U}(A))$ is thick ;*

2. *$\alpha(\pi_{I\setminus\{i\}}(\rho_{i,U}(A))) \leq \alpha(\pi_I(A)) - \log \ell + \log AvgDeg_i(\pi_I(A))$;*

3. *$\beta({C_{I\setminus\{i\}}}^{(0)}(U) \cap {C_{I\setminus\{i\}}}^{(1)}(U)) \leq \beta(C_I) + 1$.*

The decision tree that witnesses Theorem B.4.1 is Algorithm `eval`. The difference from the algorithm in [158, 96] is that, when we need to restrict the sets A and B, instead of making a query we consider both possible outcomes for Bob. We can delay committing to either outcome until the moment before Bob starts to speak, at which point we make all queries at once. We assume that Q is a queue, this is, its elements are sorted in insertion order. We also assume that $\arg\max$ decides arbitrarily in case of tie, and observe that if $b = \arg\max|\pi_I(A \cap X^{v_b})|$, then $\alpha(\pi_I(A \cap X^{v_b})) \leq \alpha(\pi_I(A)) + 1$.

Lemma B.4.6 (Main Lemma). *If Π is a protocol that computes $Lift(S)$ using communication $c < \frac{m}{2}(1-\lambda)\log \ell$ and r rounds, then* `eval` *computes S using at most $2c/(1-\lambda)\log \ell$ queries and depth r.*

```
let $A = [\ell]^m$, $B = \{0,1\}^{\ell m}$, $I = [m]$, $v$ be the root of $\Pi$
while $v$ is not a leaf do
    let $Q = \emptyset$, $C_I = \pi_I(B)$
    while $v$ is a node where Alice speaks do
        while $\exists i \in I$ such that $AvgDeg_i(\pi_I(A)) < \ell^\lambda$ do
            let $U_i$ = project $(A, C_I, I, i)$
            let $A = \rho_{i,U_i}(A)$, $C_{I\setminus\{i\}} = C_{I\setminus\{i\}}{}^{(0)}(U_i) \cap C_{I\setminus\{i\}}{}^{(1)}(U_i)$
            let $I = I \setminus \{i\}$, $Q = Q \cup \{i\}$
        let $b = \arg\max|\pi_I(A \cap X^{v_b})|$
        let $A = \text{prune}(A \cap X^{v_b}, I)$, $v = v_b$
    query coordinates $Q$ to get string $z_Q$
    for $i \in Q$ do
        let $B = \rho_{i,V}(B)$, where $V = V^{z_i}(U_i)$
    while $v$ is a node where Bob speaks do
        let $b = \arg\max|\pi_I(B \cap Y^{v_b})|$
        let $B = B \cap Y^{v_b}$, $v = v_b$
return the answer at $v$
```

Figure B.4: Procedure eval(Π,z)

Observe that Theorem B.4.1 follow from this lemma, since if $c \geq \frac{m}{2}(1-\lambda)\log\ell$ then a parallel decision tree that queries all variables in depth 1, satisfies the theorem. Before proving Lemma B.4.6, let us consider some possible protocols and what Algorithm B.4 does.

Consider the trivial protocol where Alice in one round sends all of her input to Bob, who outputs the answer. To be fair, this is a bit too much communication for Lemma B.4.6 to apply, but let us look over this fact and try to get an intuition of what happens. While Alice is speaking, the algorithm has to commit to the values of her coordinates, and therefore, for all coordinates i, $AvgDeg_i(\pi_I(A))$ eventually becomes too small and i is marked to be queried. After she finishes speaking the algorithm queries all coordinates and restricts Bob's input to be compatible with the queries and with Alice's message, i.e., it only keeps inputs that have the queried value on the corresponding position. Finally, the algorithm answers what Bob would output for any of the compatible inputs. Note that the decision tree in this case is the depth-1 decision tree that queries all coordinates at once and answers accordingly.

Another possible protocol is the one that follows a parallel decision tree T, as explained in Section B.2: Alice and Bob communicate to find the value of the coordinates queried on each node and then continue on T according to these values. For this protocol, Algorithm B.4 marks to be queried all the coordinates Alice and Bob talk about because the corresponding average degree gets too low. Note that the final decision tree is exactly the same as the one Alice and Bob were following.

Now we consider what happens in a more extreme case. Suppose the protocol is very unbalanced in the sense that Alice's first bit is a 0 if her first coordinate is 42, and otherwise is a 1. In this case, when at line 9 the algorithm chooses a bit for Alice to speak, it will always choose 1 since it is compatible with the most inputs.

Proof of Lemma B.4.6. Let R^v be the rectangle of inputs compatible with node v, and let c_v^A (resp. c_v^B) be the number of bits sent by Alice (resp. Bob) up to node v. Let χ be the number of queries so far, i.e., $\chi = m - |I|$. We show that the following invariants hold throughout the algorithm:

1. $\pi_I(A)$ is thick;
2. $A \times B \subseteq R^v$;
3. $\chi \leq 2c_v^A/(1-\lambda)\log \ell$;
4. $\beta(C_I) \leq \chi + c_v^B$;

and the following invariant holds at the beginning of each round:

5. $\beta(\pi_I(B)) \leq \chi + c_v^B$;
6. $\mathsf{select}(x_i, y_i) = z_i$ for all $(x,y) \in A \times B$ and $i \notin I$.

All six invariants are true at the beginning of the algorithm. To prove invariants 1 through 4, we assume they hold up to the current point of the algorithm and prove they hold after executing the next line.

The set A is modified at lines 7 and 10, both of which ensure that $\pi_I(A)$ is thick, therefore invariant 1 holds. We need to argue, though, that the assumptions of the corresponding lemmas hold and therefore we are allowed to apply them. For line 7 we need to argue that Lemma B.4.5 applies, and that is the case, since we assume invariant 1 holds before this point, and since invariants 3 and 4 together with the assumption of Lemma B.4.6 that $c_v^A + c_v^B \leq m \log \ell = \ell^\gamma \log \ell$ imply that $\beta(C_I) \leq \chi + c_v^B \leq 2c_v^A/(1-\lambda)\log \ell + c_v^B \leq c_v^A + c_v^B \leq \ell^\gamma \log \ell$. For line 10 we need to argue that Lemma B.4.4 applies and, indeed, we have that

$$AvgDeg_i(\pi_I(A \cap X^{v_b})) = \frac{|\pi_I(A \cap X^{v_b})|}{|\pi_{I\setminus\{i\}}(A \cap X^{v_b})|} \geq \frac{\frac{1}{2}|\pi_I(A)|}{|\pi_{I\setminus\{i\}}(A)|} = \frac{1}{2} AvgDeg_i(\pi_I(A)) \geq \ell^\lambda/2 \ . \tag{B.4.5}$$

We note that the conditions for Lemmas B.4.4 and B.4.5 to apply are more relaxed than what we prove. This is so because we will apply the same lemmas when dealing with real communication and there we will need this extra slack.

For invariant 2, first note that A and B never increase. Moreover, the rectangle of compatible inputs R^v changes when v is modified at lines 10 and 16, and in both cases we restrict A (resp. B) to a subset of X^b (resp. Y^b).

To see that we make at most $2c_v/(1-\lambda)\log\ell$ queries, we observe that each query comes from a call to `project` in line 6, which decreases α by at least $(1-\lambda)\log\ell$ because $AvgDeg_i(A_I) < \ell^\lambda$. However, $\alpha(\pi_I(A)) \leq 2c_v^A$ since, at line 9, b is chosen so that $\alpha(\pi_I(A \cap X^{v_b})) \leq \alpha(\pi_I(A)) + 1$ and thus, by Lemma B.4.4, α increases by at most 2 at line 10, i.e., after one bit of communication. Since $\alpha \geq 0$ at all times by definition, the upper bound in invariant 3 follows.

Note that C_I is only updated at lines 3 and 7. At line 3 it holds by the fact that invariant 5 is true at the beginning of a round. At line 7, Lemma B.4.5 guarantees that $\beta({C_{I\setminus\{i\}}}^{(0)}(U) \cap {C_{I\setminus\{i\}}}^{(1)}(U)) \leq \beta(C_I) + 1$. Note that it is possible to apply Lemma B.4.5 as argued before. Therefore, invariant 4 follows.

To prove that invariant 5 holds at the end of a round we distinguish whether Alice or Bob spoke. If Bob spoke, B was updated at line 16 and the invariant clearly holds since each bit b that Bob says is chosen such that $\beta(\pi_I(B \cap Y^{v_b}))$ increases by at most 1.

If Alice spoke, B is updated in line 13. Let $Q = \{i_1, \dots, i_{|Q|}\}$. Let I_0 be the non-queried coordinates at the beginning of the round and $I_\eta = I_{\eta-1} \setminus \{i_\eta\}$, for $\eta \in [|Q|]$. Moreover, let B_0 be the value of B at the beginning of the round and $B_\eta = \rho_{i,V}(B_{\eta-1})$, for $\eta \in [|Q|]$ and $i = i_\eta$. We prove by induction over η that $\pi_{I_\eta}(B_\eta) \supseteq C_{I_\eta}$. Recall that C_{I_η} is a set whose elements are vectors with coordinates in I_η and is not the projection of a set C to I_η. Therefore, the subscript serves not only as a reminder of the form of its elements, but also as an identifier of the set. At the beginning of the round $\pi_{I_0}(B_0) = C_{I_0}$. At each iteration,

$$\pi_{I_\eta}(B_\eta) = \pi_{I_\eta}(\rho_{i,V}(B_{\eta-1})) \supseteq \pi_{I_\eta}(\rho_{i,V}(\pi_{I_{\eta-1}}(B_{\eta-1}))) \tag{B.4.6}$$

$$\supseteq \pi_{I_\eta}(\rho_{i,V}(C_{I_{\eta-1}})) = {C_{I_{\eta-1}}}^{(z_i)}(U_i) \supseteq C_{I_\eta} \tag{B.4.7}$$

so at the end $\beta(\pi_{I_{|Q|}}(B_{|Q|})) \leq \beta(C_{I_{|Q|}}) \leq \chi + c_v^B$, where the last inequality follows from invariant 4.

For invariant 6, recall that A and B never increase. Furthermore, each time I is modified at line 8, we add to Q the coordinate i for which invariant 6 no longer holds (since the element i was removed from I and it has not been queried yet). Then we restore the invariant before the next iteration by restricting B at line 13. Indeed, if $(x, y) \in A \times B$, then $x_i \in U_i$ and $y_i \in V^{z_i}(U_i)$, so by definition of V it holds that $y_{i x_i} = z_i$.

We have finished proving the invariants. From now on we assume that the algorithm ended and reached a leaf v. It is clear that the decision tree has depth r and the total number of queries is at most $2c/(1-\lambda)\log\ell < m$ by invariant 3. It remains to prove correctness, that is, for any $z \in \{0,1\}^m$ that we fix, the decision tree answered $S(z)$.

In order to prove this, we show that there exists an input $(x, y) \in R_v$ such that $\mathsf{select}(x, y) = z$, which implies

$$\mathtt{eval}(\Pi, z) = \Pi(x, y) = (\mathit{Lift}(S))(x, y) = S(\mathsf{select}(x, y)) = S(z)\ . \tag{B.4.8}$$

```
1 let $A_I = \pi_I(A)$
2 while $\exists i$ such that $MinDeg_i(A_I) < \ell^\mu$ do
3     let $x'_{I\setminus\{i\}} \in V(Graph_i(A_I))$ be a right vertex of degree less than $d$
4     let $A_I = \{x \in A_I : x_{I\setminus\{i\}} \neq x'_{I\setminus\{i\}}\}$
5 return $\{x \in A : x_I \in A_I\}$
```

Figure B.5: Procedure prune(A, I)

This is done by restricting A and B to A' and B' so that any input in A' and any input in B' are compatible with z and finally arguing that A' and B' are non-empty. For every queried coordinate i, we have already restricted A and B so that for any $x \in A$ and any $y \in B$, $\mathsf{select}(x_i, y_i) = z_i$; thus we are left to deal with the non-queried coordinates.

Note that $Graph_i(\pi_{\{i\}}(A))$ has only one vertex on the right with label $\emptyset$, and an edge from this vertex to an element $x_i \in [\ell]$ if $x_i \in \pi_{\{i\}}(A)$. Although $Graph_i(\pi_\emptyset(A))$ is not defined (which is the reason we have to apply Claims B.4.7 and B.4.8 directly and not Lemma B.4.5), the projection $\pi_\emptyset(A)$ is well-defined and it is either the empty set, in which case A is empty, or it is the singleton set containing the empty string, in which case there exists at least one element $x \in A$. The same holds for B, $\pi_\emptyset(B)$ is either empty or it is the singleton set containing the empty string. Therefore, by the definition of α and β, if $\alpha(\pi_\emptyset(A))$ (resp. $\beta(\pi_\emptyset(B))$) is finite, then A (resp. B) is non-empty.

We start with $A' = A$, $B' = B$ and $I' = I$ such that $\pi_{I'}(A')$ is thick, $\beta(\pi_{I'}(B')) \le \chi + c_v^B \le \ell^\gamma \log \ell$ and I' is non-empty since we queried less than m coordinates. While I' is not empty, we choose $i \in I'$ and apply Claims B.4.7 and B.4.8 to get a set U such that $\pi_{I'\setminus\{i\}}(\rho_{i,U}(A')) = \pi_{I'\setminus\{i\}}(A')$ and $\beta(\pi_{I'\setminus\{i\}}(B')^{(0)}(U) \cap \pi_{I'\setminus\{i\}}(B')^{(1)}(U)) \le \beta(\pi_I(B')) + 1$. We then set $A' = \rho_{i,U}(A')$ and $B' = \rho_{i,V}(B')$, where $V = V^{z_i}(U)$, and thus, by definition of V, for all $x \in A'$ and $y \in B'$ it holds that $\mathsf{select}(y_i, x_i) = z_i$. Finally we set $I' = I' \setminus \{i\}$ and repeat. Note that these claims can indeed be applied while I' is not empty, since the fact that thickness is monotone (Observation B.4.2) implies $\pi_{I'}(A')$ is thick, and since $\beta(\pi_{I'}(B')) \le c_v^B + (\chi + |I|) \le c + m < 2\ell^\gamma \log \ell$, because $\beta(\pi_{I'}(B'))$ is increasing by at most 1 for every $i \in I$.

At the end of this process, the set B' is such that $\beta(\pi_{I'}(B')) \le 2\ell^\gamma \log \ell < \infty$, and thus B' is non-empty. Moreover, we have that $\pi_\emptyset(A') = \pi_\emptyset(A)$, and since A is non-empty (because $\alpha(\pi_I(A)) \le 2c_v^A$), $\pi_\emptyset(A')$ is the singleton set containing the empty string and, therefore, A' is non-empty. □

Now we explain the auxiliary procedures. Procedure prune just removes vertices with too small degree. Thus, we only need to argue that this process does not last for too long.

Proof of Lemma B.4.4. $A'_I = \pi_I(A')$ is thick by construction. It remains to show that $|A'_I| \ge |\pi_I(A)|/2$. For each coordinate $i \in I$ we observe that, by the definition of

```
1 let $U \subseteq [\ell]$ be a set such that
2     $\pi_{I\setminus\{i\}}(\rho_{i,U}(A)) = \pi_{I\setminus\{i\}}(A)$
3     $\beta({C_{I\setminus\{i\}}}^{(0)}(U) \cap {C_{I\setminus\{i\}}}^{(1)}(U)) \geq \beta(C_I) + 1$
4 return $U$
```

Figure B.6: Procedure `project`(A, C_I, I, i)

average degree, there are at most $|\pi_I(A)|/\mathit{AvgDeg}_i(\pi_I(A))$ right vertices of positive degree in $\mathit{Graph}_i(\pi_I(A))$, and since $\mathit{Graph}_i(A'_I) \subseteq \mathit{Graph}_i(\pi_I(A))$, there are at most $|\pi_I(A)|/\mathit{AvgDeg}_i(\pi_I(A))$ right vertices of positive degree in $\mathit{Graph}_i(A'_I)$. Clearly this is an upper bound on the number of iterations of the procedure `prune` for coordinate i, since every iteration removes a right vertex of positive degree. Since $\mathit{AvgDeg}_i(\pi_I(A)) \geq \ell^\lambda/4$ for all $i \in I$, the total number of iterations for each coordinate is at most

$$|\pi_I(A)|/\mathit{AvgDeg}_i(\pi_I(A)) \leq 4|\pi_I(A)|/\ell^\lambda \ , \tag{B.4.9}$$

which makes a total of at most $4|I||\pi_I(A)|/\ell^\lambda \leq 4\ell^{\gamma-\lambda}|\pi_I(A)|$ iterations overall. Each iteration removes $\deg(X'_{I\setminus\{i\}}) < \ell^\mu$ elements from A'_I, for a total of at most $4\ell^{\gamma+\mu-\lambda}|\pi_I(A)| \leq |\pi_I(A)|/2$ elements removed. The last inequality holds because of assumption (B.4.1a). Thus, at least $|\pi_I(A)|/2$ elements remain. □

Our Lemma B.4.5 is slightly stronger than the projection lemmas in [158, 96] because it needs to handle both outcomes of the query to z_i, so we care not only about each of $\rho_{i,V^b(U)}(B)$ separately but about $B^{(0)}(U) \cap B^{(1)}(U)$. Nonetheless, essentially the same proof of [158] using the probabilistic method works—but not that of [96], where the probabilities are too small for us. Procedure `project`, therefore, just asserts the existence of a return value U with the required properties.

The probabilities in the claims below are with respect to U picked uniformly among all subsets of $[\ell]$ of size ℓ^δ.

Claim B.4.7 ([158]). *If A is thick, then $\pi_{I\setminus\{i\}}(\rho_{i,U}(A)) = \pi_{I\setminus\{i\}}(A)$ with probability* $1 - o(1)$.

Claim B.4.8. *If $\beta(C_I) \leq 2\ell^\gamma \log \ell$, then $\beta({C_{I\setminus\{i\}}}^{(0)}(U) \cap {C_{I\setminus\{i\}}}^{(1)}(U)) \leq \beta(C_I) + 1$ with probability* $1 - o(1)$.

Proof of Lemma B.4.5. By union bound there exists U that satisfies Claim B.4.7 and Claim B.4.8. Item 1 then follows from Observation B.4.2, and item 2 from Observation B.4.3. □

Proof of Claim B.4.7. The equality holds if every right vertex of $\mathit{Graph}_i(\pi_I(A))$ with positive degree has an edge into U. Since $\mathit{MinDeg}_i(\pi_I(A)) \geq \ell^\mu$, the probability that U

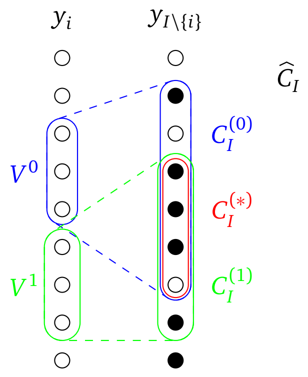


Figure B.7: Sets used in the proof of Claim B.4.8

is contained within the non-neighbours of any right vertex $x_{I\setminus\{i\}}$ is

$$\frac{\binom{\ell-|N_i(x_{I\setminus\{i\}})|}{\ell^\delta}}{\binom{\ell}{\ell^\delta}} \leq \frac{\binom{\ell-\ell^\mu}{\ell^\delta}}{\binom{\ell}{\ell^\delta}} \leq (1-\ell^{\mu-1})^{\ell^\delta} \leq e^{-\ell^{\mu+\delta-1}} \quad . \tag{B.4.10}$$

By a union bound over all right vertices, the probability of the equality not holding is at most

$$\ell^{|I|-1}e^{-\ell^{\mu+\delta-1}} < e^{|I|\log\ell-\ell^{\mu+\delta-1}} \leq e^{\ell^\gamma\log\ell-\ell^{\mu+\delta-1}} = o(1) \ . \tag{B.4.11}$$

The last equality holds because of assumption (B.4.1b). □

The proof of Claim B.4.8 follows [158] except that our version of Claim B.4.10 is stronger and we apply it twice.

Proof of Claim B.4.8. We begin by observing that ${C_{I\setminus\{i\}}}^{(b)}(U)$, the set of right vertices that can be completed to b over U, is equal to $N_j(V^b(U))$ (see Figure B.7). We want to prove that the set $C^{(*)}_{I\setminus\{i\}}(U) = {C_{I\setminus\{i\}}}^{(0)}(U) \cap {C_{I\setminus\{i\}}}^{(1)}(U)$ of right vertices that can be completed to any colour over U is large, namely that $|C^{(*)}_{I\setminus\{i\}}|/2^{\ell(|I|-1)} \geq \psi/2$ where $\psi = |C_I|/2^{\ell|I|} = 2^{-\beta(C_I)}$.

Right vertices of degree larger than $2^\ell\psi/4$ can be completed to any colour with high probability. Indeed, $|N_i(y_{I\setminus\{i\}})| \geq 2^\ell \cdot \psi/4 = 2^\ell \cdot 2^{-\beta(C)}/4 \geq 2^\ell \cdot 2^{-2\ell^\gamma\log^2\ell}/4 \geq 2^\ell \cdot 2^{-3\ell^\gamma\log^2\ell}$, so for each $b \in \{0,1\}$ we can apply Claim B.4.10 with $\phi = 3\log^2\ell$ to show that $N_i(y_{I\setminus\{i\}}) \cap V^b(U) \neq \emptyset$ with probability $1-o(1)$. Taking a union bound, we can assume that $N_i(y_{I\setminus\{i\}}) \cap V^b(U) \neq \emptyset$ holds for both $b \in \{0,1\}$ except with probability $o(1)$. In other words, $y_{I\setminus\{i\}} \in C^{(b)}_{I\setminus\{i\}}$ for $b \in \{0,1\}$, so $y_{I\setminus\{i\}} \in C^{(*)}_{I\setminus\{i\}}$.

We have shown that for every $y_{I\setminus\{i\}} \in \widehat{C}_{I\setminus\{i\}} = \{y_{I\setminus\{i\}} : \deg(y_{I\setminus\{i\}}) \geq 2^\ell\psi/4\}$ it holds that $y_{I\setminus\{i\}} \in C^{(*)}_{I\setminus\{i\}}$ with probability $1-o(1)$. By Observation B.4.9, with probability

$1 - o(1)$,

$$|C^{(*)}_{I\setminus\{i\}}| \geq 2/3 \cdot |\widehat{C}_{I\setminus\{i\}}| \ . \tag{B.4.12}$$

In fact, 2/3 can be chosen to be any arbitrary number strictly smaller than 1, and the statement would still hold.

Therefore it is enough to prove that the set $\widehat{C}_{I\setminus\{i\}}$ of right vertices with large degree is large. Indeed, we have

$$2^{\ell|I|}\psi = |C_I| \leq |\widehat{C}_{I\setminus\{i\}}| \cdot 2^{\ell} + |\{0,1\}^{\ell(|I|-1)} \setminus \widehat{C}_{I\setminus\{i\}}| \cdot 2^{\ell} \cdot \psi/4 \tag{B.4.13}$$

$$\leq |\widehat{C}_{I\setminus\{i\}}| \cdot 2^{\ell} + 2^{\ell(|I|-1)} \cdot 2^{\ell} \cdot \psi/4 = |\widehat{C}_{I\setminus\{i\}}| \cdot 2^{\ell} + 2^{\ell|I|} \cdot \psi/4 \ , \tag{B.4.14}$$

from where

$$|\widehat{C}_{I\setminus\{i\}}| \geq 3/4 \cdot \psi \cdot 2^{\ell(|I|-1)} \ . \tag{B.4.15}$$

and the claim follows by combining equations (B.4.12) and (B.4.15). We observe that the 1/4 in the definition of right vertices with large degree can be any arbitrarily small constant. □

Observation B.4.9. *Let T be a set and S a set-valued random variable. If* $\Pr[s \in S] \geq p$ *for every $s \in T$, then* $\Pr[|S| \geq q|T|] \geq (p-q)/(1-q)$*.*

Proof. Let $x = \Pr[|S| \geq q|T|]$. Then

$$p \leq \Pr[s \in S] = \Pr[s \in S \mid |S| \geq q|T|]\Pr[|S| \geq q|T|] + \Pr[s \in S \mid |S| < q|T|]\Pr[|S| < q|T|] \tag{B.4.16}$$

$$\leq 1 \cdot x + q \cdot (1-x) \ , \tag{B.4.17}$$

from which the observation follows. □

As before, we think of $W \subseteq \{0,1\}^{\ell}$ as a set of binary colourings of $[\ell]$ and denote by $\pi_U(W)$ the projection of W to a subset $U \subseteq [\ell]$, i.e. $\pi_U(W) = \{w_U \in \{0,1\}^U : w \in W \text{ for some } w_{[\ell]\setminus U} \in \{0,1\}^{[\ell]\setminus I}\}$. Note that this is the same operation as $\pi_I(A)$, except they apply to different domains.

Claim B.4.10. *Let $W \subseteq \{0,1\}^{\ell}$ be any set of size at least $2^{-\phi\ell^{\gamma}} \cdot 2^{\ell}$, where ϕ is any function of ℓ such that* $\log\phi = o(\log\ell)$*. Let U be a uniformly random subset of $[\ell]$ of size ℓ^{δ}. Then, for any $b \in \{0,1\}$, $\{b\}^{|U|} \in \pi_U(W)$ with probability at least* $1 - o(1)$*.*

In the original paper [158] the constants are set to $\gamma = 2/20$, $\delta = 5/20$. The same probabilistic argument works for any choice of constants such that $\gamma + 3\delta/2 < 1$. Here we present a combinatorial proof that works for any constants such that $\gamma + \delta < 1$, and in particular holds for $\gamma = 1/3 - \xi$ and $\delta = 2/3$, for any $\xi > 0$.

Proof. We will prove the following equivalent statement: if $\Pr_U[\{b\}^{|U|} \notin \pi_U(W)] \geq q$ (for, say, $q = \phi/\log\ell$), then $|W| \leq 2^{\ell - \phi\ell^{\gamma}}$. We will only use the fact that $\gamma + \delta < 1 + \frac{\log\frac{q}{\phi}}{\log\ell}$ (for $q = \phi/\log\ell$ this says that $\gamma + \delta < 1 - \frac{\log\log\ell}{\log\ell}$). The following statement, which is a corollary of Kruskal–Katona Theorem, will be proved later on.

Claim B.4.11. *Let $\mathcal{U}$ be any family of subsets of $[\ell]$, where every $U \in \mathcal{U}$ is of size u. If $|\mathcal{U}| \geq \sum_{i=0}^{t} \binom{\ell-1-i}{\ell-u-i}$ and if $W \subseteq \{0,1\}^\ell$ is such that $\{b\}^{|U|} \notin \pi_U(W)$ for every $U \in \mathcal{U}$, then it holds that $|W| \leq 2^\ell - \sum_{j=0}^{\ell-u} \sum_{i=0}^{t} \binom{\ell-1-i}{\ell-u-i-j}$.*

Let $\mathcal{U}$ be the set of all $U \subset [\ell]$ of size ℓ^δ such that $\{b\}^{|U|} \notin \pi_U(W)$. We have that

$$\frac{|\mathcal{U}|}{\binom{\ell}{\ell^\delta}} = \Pr_U[\{b\}^{|U|} \notin \pi_U(W)] \geq q \ . \tag{B.4.18}$$

Hence we get

$$|\mathcal{U}| \geq q\binom{\ell}{\ell^\delta} = q\frac{\ell}{\ell^\delta}\binom{\ell-1}{\ell^\delta-1} \geq \sum_{i=0}^{q\frac{\ell}{\ell^\delta}} \binom{\ell-1-i}{\ell^\delta-1} = \sum_{i=0}^{q\frac{\ell}{\ell^\delta}} \binom{\ell-1-i}{\ell-\ell^\delta-i} \ . \tag{B.4.19}$$

We can therefore apply Claim B.4.11 with $u = \ell^\delta$ and $t = q\frac{\ell}{\ell^\delta}$ to get

$$|W| \leq 2^\ell - \sum_{j=0}^{\ell-\ell^\delta} \sum_{i=0}^{q\frac{\ell}{\ell^\delta}} \binom{\ell-1-i}{\ell-\ell^\delta-i-j} \leq 2^{\ell-q\frac{\ell}{\ell^\delta}+1} + 2^{\ell^\delta \log \ell} \leq 2^{\ell-\phi\ell^\gamma} \ , \tag{B.4.20}$$

where the second inequality is a straightforward calculation that we prove in Claim B.4.15 and the last inequality follows from $\gamma + \delta < 1 + \frac{\log \frac{q}{\phi}}{\log \ell}$. □

To prove Claim B.4.11 we need to introduce some terminology from extremal combinatorics.

We use the following terminology from [115]. If w is a binary colouring of $[\ell]$ (i.e. a binary vector of length ℓ), we say a *neighbour* of a w is a colouring which can be obtained from w by flipping one of its 1-entries to 0. A *shadow* of a set $A \subseteq \{0,1\}^\ell$ of binary colourings is the set $\partial(A)$ of all its neighbours. A set A is *k-regular* if every colouring in A colours exactly k elements 1. Note that in this case $\partial(A)$ is $(k-1)$-regular. The best possible lower bounds for the size of $\partial(A)$ were obtained independently by Kruskal [127] and Katona [119].

Theorem B.4.12 (Kruskal–Katona Theorem). *If $A \subseteq \{0,1\}^\ell$ is k-regular, and if*

$$|A| = \binom{a_k}{k} + \binom{a_{k-1}}{k-1} + \ldots + \binom{a_s}{s}$$

then

$$|\partial(A)| \geq \binom{a_k}{k-1} + \binom{a_{k-1}}{k-2} + \ldots + \binom{a_s}{s-1} \ .$$

Let $\mathcal{P}(S)$ denote the power set of S. In [85] it is proven that there exists an explicit compression function $C : \mathcal{P}(\{0,1\}^\ell) \to \mathcal{P}(\{0,1\}^\ell)$ such that, given a k-regular set $A \subseteq \{0,1\}^\ell$, $C(A)$ is k-regular, $|C(A)| = |A|$ and $|\partial(C(A))|$ matches the lower bound in Theorem B.4.12, and, furthermore, the following proposition holds.

Proposition B.4.13. $\partial(C(A)) \subseteq C(\partial(A))$.

Although this follows directly from the proposition in Section 2 of [85], the formulation above is from [10].

We define the *iterated shadow* of a k-regular set $A \subseteq \{0,1\}^\ell$ to be $\partial^{\leq k}(A) = \cup_{j=0}^{k} A_j$, where $A_0 = A$ and $A_j = \partial(A_{j-1})$ for $0 < j \leq k$. The following corollary follows immediately from Theorem B.4.12 and Proposition B.4.13.

Corollary B.4.14. *If $A \subseteq \{0,1\}^\ell$ is k-regular, and if*

$$|A| = \binom{a_k}{k} + \binom{a_{k-1}}{k-1} + \ldots + \binom{a_s}{s}$$

then

$$|\partial^{\leq k}(A)| \geq \sum_{j=0}^{k} \binom{a_k}{k-j} + \binom{a_{k-1}}{k-1-j} + \ldots + \binom{a_s}{s-j} .$$

Proof of Claim B.4.11. Given $\mathcal{U}$, define $W_{\mathcal{U}}$ to be the largest set of colourings $\{0,1\}^\ell$ such that $\{b\}^{|U|} \notin \pi_U(W)$ for all $U \in \mathcal{U}$. For simplicity, we will consider $b = 0$, i.e. $W_{\mathcal{U}}$ is the set that contains all the colourings of $[\ell]$ that do not colour any $U \in \mathcal{U}$ completely 0.

Let $\mathcal{U}' \subseteq \mathcal{U}$ be a set of size exactly $\sum_{i=0}^{t} \binom{\ell-1-i}{\ell-u-i}$. Obviously, $W_{\mathcal{U}'}$ is at least as large as $W_{\mathcal{U}}$.

Let $\mathbf{1}_{U^C} \in \{0,1\}^\ell$ be the indicator functions for the complement of a set U, i.e. $\mathbf{1}_{U^C} = 1 - \mathbf{1}_U$. Let A be the set of $\mathbf{1}_{U^C} \in \{0,1\}^\ell$ for $U \in \mathcal{U}'$. Note that A is $(\ell - u)$-regular and that the iterated shadow of A is exactly the set of colourings that are not in $W_{\mathcal{U}'}$. Applying Corollary B.4.14 to A, we get

$$|\partial^{\leq k}(A)| \geq \sum_{j=0}^{k} \sum_{i=0}^{t} \binom{\ell-1-i}{k-i-j} = \sum_{j=0}^{\ell-u} \sum_{i=0}^{t} \binom{\ell-1-i}{\ell-u-i-j}.$$

Therefore, $|W_{\mathcal{U}}| \leq |W_{\mathcal{U}'}| = 2^\ell - |\partial^{\leq k}(A)| \leq 2^\ell - \sum_{j=0}^{\ell-u} \sum_{i=0}^{t} \binom{\ell-1-i}{\ell-u-i-j}$. □

For completeness we include the calculations needed in Claim B.4.10.

Claim B.4.15. *It holds that*

$$\sum_{j=0}^{\ell-u} \sum_{i=0}^{t} \binom{\ell-1-i}{\ell-u-i-j} \geq 2^\ell - 2^{\ell-t+1} + 2^{u \log \ell} .$$

Proof. The claim follows from the following sequence of elementary calculations

$$\begin{aligned}
&\sum_{j=0}^{\ell-u}\sum_{i=0}^{t}\binom{\ell-1-i}{\ell-u-i-j}\\
&=\sum_{j=0}^{\ell-u}\sum_{i=0}^{t}\binom{\ell-1-i}{\ell-1-j} && (*)\\
&=\sum_{j=0}^{\ell-u}\left(\sum_{i=0}^{\ell-1}\binom{i}{\ell-1-j}-\sum_{i=0}^{\ell-t}\binom{i}{\ell-1-j}\right)\\
&=\sum_{j=0}^{\ell-u}\left(\binom{\ell}{\ell-j}-\binom{\ell-t+1}{\ell-j}\right) && (**)\\
&=\sum_{j=0}^{\ell-u}\binom{\ell}{j}-\sum_{j=0}^{\ell-u-t+1}\binom{\ell-t+1}{j} && (*)\\
&=2^{\ell}-\sum_{j=\ell-u+1}^{\ell}\binom{\ell}{j}-2^{\ell-t+1}+\sum_{j=\ell-u-t+2}^{\ell-t+1}\binom{\ell-t+1}{j}\\
&\geq 2^{\ell}-2^{\ell-t+1}+\sum_{j=0}^{u-1}\binom{\ell}{j}\\
&\geq 2^{\ell}-2^{\ell-t+1}+\ell^{u}=2^{\ell}-2^{\ell-t+1}+2^{u\log\ell}\quad ,
\end{aligned}$$

where the equalities in $(*)$ follow from renaming of variables and the fact that $\binom{n}{k}=\binom{n}{n-k}$; the equality in $(**)$ follows from $\sum_{i=0}^{n-1}\binom{i}{k-1}=\binom{n}{k}$. □

B.4.2 Simulation of Decision Trees by Real Communication Protocols

In this section we show how to adapt the simulation theorem to real communication.

Theorem B.4.16. *If there is a real communication protocol computing Lift(S) using communication c and r rounds, then there is a parallel decision tree computing S using* $\mathrm{O}(c/\log\ell)$ *queries and depth r.*

The proof follows the same strategy as in the deterministic case, this is we are going to construct a decision tree by simulating a real communication protocol and only querying the coordinates where the communication protocol would have too much information on x_i.

The major difference in analyzing real communication protocols as opposed to deterministic ones is that the set of compatible inputs is not a rectangle, but a monotone set as defined next.

Definition B.4.17. A Boolean matrix M is *monotone* if $M_{i_1 j_1}\leq M_{i_2 j_2}$, for all pairs of entries such that $i_1\leq i_2$ and $j_1\leq j_2$.

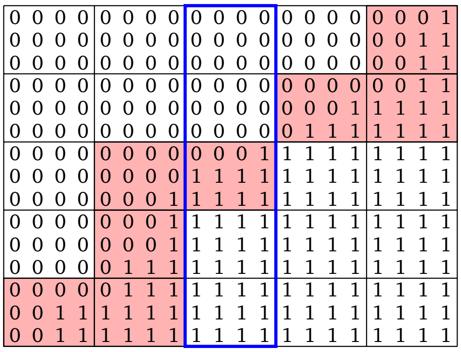


Figure B.8: Monotone matrix partitioned in 5×5 blocks; the 3rd block-column has 4 monochromatic blocks.

Recall that each communication step is a comparison $\phi(x) \le \psi(y)$ and that we restrict our attention to inputs in a set $A \times B$. We lay out the results of the comparison in the matrix $(⟦\phi(x) \le \psi(y)⟧)_{x,y}$ indexed by $x \in A$, $y \in B$, with rows sorted decreasingly according to ϕ and columns increasingly according to ψ. Note that we use the Iverson bracket notation

$$⟦Z⟧ = \begin{cases} 1 & \text{if the Boolean expression } Z \text{ is true;} \\ 0 & \text{otherwise.} \end{cases} \tag{B.4.21}$$

The communication matrix is monotone: if $\phi(x_1) \ge \phi(x_2)$ and $\psi(x_1) \le \psi(x_2)$ then $\phi(x_2) \le \phi(x_1) \le \psi(y_1) \le \psi(y_2)$.

The fact that the set of compatible inputs is not a rectangle can be circumvented, since as observed in [113] in every monotone matrix there exists one quadrant—thus a large rectangle—that is monochromatic. It is therefore possible to restrict the set of compatible inputs to a quadrant when we want to choose the outcome of a comparison, as done in [43].

However, this is not enough for us. Since we want to query variables only at the end of each round of k comparisons, and after using procedure `project` we no longer know what B is, we need to restrict the inputs to rectangles beforehand. This means we have to avoid shrinking A too much, and definitely less than the 2^k factor we would get by picking quadrants.

Our solution is to partition the matrix into $(k+1) \times (k+1)$ blocks of size $|A|/(k+1) \times |B|/(k+1)$ and then restrict Bob's input to one of the $(k+1)$ block-columns, so that Alice's input forms a rectangle in k out of the $k+1$ block-rows (see Figure B.8). Formally, we have the following lemma.

Lemma B.4.18. *Let M be a monotone matrix partitioned into $(k+1) \times (k+1)$ blocks. There is a block-column such that k of its blocks are monochromatic.*

Proof. Consider a non-monochromatic block. Since its bottom-right corner has value 1, all blocks below and to the right are 1-blocks. This is, if we consider non-monochromatic blocks left-to-right in a sequence, then a block cannot be below its predecessor, which means that there are at most $2k+1$ non-monochromatic blocks overall. By the pigeonhole principle, at least one of the column-blocks contains at most one non-monochromatic block. □

We take advantage of this lemma with the following construction. Given two sets $A \subseteq [\ell]^m$ and $B \subseteq \{0,1\}^{\ell m}$, we define the b-monochromatic part of A with respect to B as $A[b,B]_{\phi,\psi} = \{x \in A : \forall y \in B \ [\![\phi(x) \leq \psi(y)]\!] = b\}$.

For technical reasons we want each element in $\pi_I(A)$ to have a unique completion to A. Therefore we define a new operator $\sigma_I(A) = \{x \in A : \forall x' \in A \text{ if } x'_I = x_I \text{ then } x < x'\}$, where the order is, say, the lexicographic order. In other words, each element of $\sigma_I(A)$ is the minimum among all elements of A that share the same I coordinates. Observe that $\pi_I(A) = \pi_I(\sigma_I(A))$. We define $\sigma_I(B)$ analogously.

We have all the ingredients to explain the simulation procedure `eval`. Note that the comparisons at lines 4 and 9 are the same.

Lemma B.4.19 (Main Lemma). *If* Π *is a real protocol that computes* Lift(S) *using communication* $c < \frac{m}{2}(1-\lambda)\log\ell$ *and* r *rounds, then* `eval` *computes* S *using* $5c/(1-\lambda)\log\ell$ *queries and depth* r.

Proof. The proof is very similar to the proof of Lemma B.4.6. Let R^v be the set (not necessarily a rectangle) of inputs compatible with node v, let c_v be the amount of communication up to node v, and let r_v be the number of rounds up to node v. Let χ be the number of queries so far, i.e., $\chi = m - |I|$. We show that the following invariants hold throughout the algorithm:

1. $\pi_I(A)$ is thick;
2. $A \times B \subseteq R^v$;
3. $\chi \leq (2c_v + 3r_v)/(1-\lambda)\log\ell$;
4. $\beta(C_I) \leq (c_v + k_v)\log(c_v+1) + \chi$;

and the following invariants hold at the beginning of each round:

5. $\beta(\pi_I(B)) \leq c_v \log(c_v+1) + \chi$;
6. $\mathsf{select}(x_i, y_i) = z_i$ for all $(x,y) \in A \times B$ and $i \notin I$.

All five invariants are true at the beginning of the algorithm.

The main difference with the proof of Lemma B.4.6 is proving invariant 1, because we modify A not only at lines 12 and 15, but also at line 7. At each point A is modified, the corresponding procedure ensures that $\pi_I(A)$ is thick. We need to argue, though, that the

```
1  let $A = [\ell]^m$, $B = \{0,1\}^{\ell m}$, $I = [m]$, $v$ be the root of $\Pi$
2  while $v$ is not a leaf do
3      let $A' = \sigma_I(A)$, $B = \sigma_I(B)$
4      foreach comparison $\phi$ vs $\psi$ do
5          let $B = \arg\max_{|B'|=|B|/(k_v+1)} |A'[0,B']_{\phi,\psi} \cup A'[1,B']_{\phi,\psi}|$
6          let $A' = A'[0,B]_{\phi,\psi} \cup A'[1,B]_{\phi,\psi}$
7      let $A$ = prune $(A', I)$
8      let $Q = \emptyset$, $C_I = \pi_I(B)$
9      foreach comparison $\phi$ vs $\psi$ do
10         while $\exists i \in I$ such that $AvgDeg_i(\pi_I(A)) < \ell^\lambda$ do
11             let $U_i$ = project $(A, C_I, I, i)$
12             let $A = \rho_{i,U_i}(A)$, $C_{I\setminus\{i\}} = C_{I\setminus\{i\}}{}^{(0)}(U_i) \cap C_{I\setminus\{i\}}{}^{(1)}(U_i)$
13             let $I = I \setminus \{i\}$, $Q = Q \cup \{i\}$
14         let $b_j = \arg\max |\pi_I(A[b,B]_{\phi,\psi})|$
15         let $A$ = prune$(A[b_j,B]_{\phi,\psi}, I)$
16     query coordinates $Q$ to get string $z_Q$
17     for $i \in Q$ do
18         let $B = \rho_{i,V}(B)$, where $V = V^{z_i}(U_i)$
19     let $v = v_{b_1,\ldots,b_k}$
20 return the answer at $v$
```

Figure B.9: Procedure eval(Π,z)

assumptions of the corresponding lemmas hold, and therefore it is correct to apply them. The argument for applying prune in line 15 is the same as in the proof of Lemma B.4.6. For line 12, we note that, by invariants 4 and 3, $\beta(C_I) \le (c_v + k_v) \cdot \log(c_v + 1) + \chi \le \frac{m}{2}(\log \ell) \cdot 2(\log m) + 5c_v \le \ell^\gamma \log^2 \ell + 5\ell^\gamma \log \ell \le 2\ell^\gamma \log^2 \ell$. Hence we only need to prove that we can apply Lemma B.4.4 in line 7.

We begin by observing that at line 3, $AvgDeg_i(\pi_I(A')) \ge \ell^\lambda$, since $\pi_I(A') = \pi_I(A)$. Furthermore, at line 6, the size of A' decreases by at most a $1-1/(k_v+1)$ fraction. Indeed, if we divide the comparison matrix $(\llbracket \phi(x) \le \psi(y) \rrbracket)_{x,y}$ into $(k_v+1) \times (k_v+1)$ blocks of size $|A'|/(k_v+1) \times |B|/(k_v+1)$, by Lemma B.4.18 at least one of the column-blocks contains k_v monochromatic blocks, i.e., a $1-1/(k_v+1)$ fraction is monochromatic.

Since we have at most k_v comparisons, the size of A' at line 7 is at least a $(1-1/(k_v+1))^{k_v} \ge 1/4$ fraction of the original. Also, after line 3 there is a bijection between A' and $\pi_I(A')$, so the size of $\pi_I(A')$ is also at least a $1/4$ fraction and Lemma B.4.4 applies.

For invariant 2, note that A and B never increase and that the set of compatible inputs R^v only changes when v is modified at line 19. However, A was restricted at line 15 so that $\llbracket \phi(x) \le \psi(y) \rrbracket = b$ for every $x \in A$ and $y \in B$; in other words $A \times B \subseteq R^{v_b}$, therefore invariant 2 holds. Note that we can only restrict A in this manner because we had already restricted B in line 5.

To see that we make at most $(2c_v + 3r_v)/(1-\lambda)\log \ell$ queries, we observe that each query comes from a call to `project` in line 11, which decreases α by at least $(1-\lambda)\log \ell$ because $\mathit{AvgDeg}_i(A_I) < \ell^{\lambda}$. However, α only increases (by at most 2) at line 15, i.e., after one bit of communication, and (by at most 3) at line 7, i.e., once per round. Since $\alpha \geq 0$ at all times by definition, the upper bound in invariant 3 follows.

To prove invariants 4 and 5 we observe that B shrinks at two points. One is at line 5, where β increases by $\log(k_v + 1) \leq \log c$ with every bit of communication. Therefore, when we set $C_I = \pi_I(B)$ at line 8 invariant 4 holds. In line 12, Lemma B.4.5 guarantees that $\beta(C_I)$ increases by at most 1 with every query, therefore $\beta(C_I) \leq (c_v + k_v)\log(c+1) + \chi$ holds at all times. Finally, we note that $\beta(\pi_I(B)) \leq \beta(C_I)$, by the same argument as in the proof of invariant 5 of Lemma B.4.6, and that c_v is updated to $c_v + k_v$ before the next round.

For invariant 6, recall that A and B never increase. Moreover, each time I is modified at line 13, we add the coordinate i for which invariant 6 breaks to Q. Then we restore the invariant before the next iteration by restricting B at line 18. Indeed, if $(x, y) \in A \times B$, then $x_i \in U$ and $y_i \in V^{z_i}(U)$, so by definition of V it holds that $y_{i x_i} = z_i$.

It is clear that the decision tree has depth r and the total number of queries is at most $5c/(1-\lambda)\log \ell$ by invariant 3. The proof of correctness is identical to that of Lemma B.4.6. □

B.5 From Parallel Decision Trees to Dymond–Tompa Games

In this section we prove that the adversary argument on a parallel decision tree for the falsified search problem of a pebbling contradiction gives a Pebbler strategy for the Dymond–Tompa game.

It is more convenient to work with the Dymond–Tompa game when there is a challenged pebble at all times. Therefore in this and the following section we use an alternative but equivalent definition. Initially the unique sink has a pebble and it is challenged, and then the game starts without a special first round. The number of rounds is the number of actual rounds, not counting the setup, and the cost is the total number of pebbles, including the initial pebble on the sink.

Lemma B.2.6 (Restated). *If there is a parallel decision tree for $\mathit{Search}\big(\mathit{Peb}_G\big)$ in depth r using at most c queries, then Pebbler has a winning strategy in the r-round Dymond–Tompa game on G in cost at most $c + 1$.*

We prove that, in fact, the parallel decision tree complexity of the falsified clause search problem of a pebbling contradiction is equivalent to the Dymond–Tompa game on the graph with an extra sink on top. Formally, we define $\widehat{G}$ as a graph with vertices $V(G) \cup \{t\}$ and edges $E(G) \cup \{(z, t)\}$, where z is the unique sink of G. Clearly the game on $\widehat{G}$ needs as many pebbles as G, and one more pebble is enough, so Lemma B.2.6 follows from Lemma B.5.1.

Lemma B.5.1. *There is a parallel decision tree for* $Search\big(Peb_G\big)$ *in depth r using c queries if and only if Pebbler has a winning strategy in the r-round Dymond–Tompa game on $\widehat{G}$ in cost $c + 1$.*

Proof. Assume there is a parallel decision tree for $Search\big(Peb_G\big)$ in depth r using c queries. We construct a strategy for Pebbler in r rounds and $c + 1$ pebbles. We say that a vertex s reaches a vertex t if there is a (possibly empty) path from s to t where all intermediate vertices are not queried. We keep these invariants.

1. The challenged pebble in the Dymond–Tompa game is false.
2. A false vertex is reachable from another false vertex if and only if it is not challenged in the Dymond–Tompa game.
3. In the subtree of the challenged pebble a vertex has a pebble if and only if it has been queried.

When it is Pebbler's turn, Pebbler looks at the decision tree and places pebbles in those vertices being queried that can reach the challenged pebble. After Challenger's turn, Pebbler follows the branch in the decision tree in which the challenged pebble is false and other vertices are false if they are reachable from a false vertex or true otherwise.

Dymond–Tompa moves are valid and the invariants are kept. When we reach a leaf in the decision tree we made at most c queries in r rounds by assumption, therefore Pebbler also used at most c pebbles on vertices of G plus one pebble on the extra sink and r rounds. It remains to show that the Dymond–Tompa game also ended. The decision tree points to a falsified clause, which is not the sink axiom because the sink is always false. Therefore we have a false vertex whose predecessors are true. By item 2, that false pebble is challenged, and by item 3 all of its predecessors have pebbles, therefore the Dymond–Tompa game also ended.

Assume there is a Pebbler strategy in r rounds and $c + 1$ pebbles. We construct a parallel decision tree for $Search\big(Peb_G\big)$ in depth r using c queries.

We look at the strategy for Pebbler and add a node to the decision tree that queries the variables corresponding to vertices being pebbled that can reach the challenged pebble. For each branch, we simulate a Challenger move. We consider the set of new vertices coloured false and that are not reachable by any false vertex. If this set is empty, then Challenger stays. Otherwise Challenger jumps to any of these vertices.

Dymond–Tompa moves are valid and the invariants are kept. When the Dymond–Tompa game ends, Pebbler has used at most $c + 1$ pebbles in r rounds by assumption, one of which outside G, therefore the decision tree also made at most c queries in r rounds. It remains to show that we can label the leaves of the decision tree in such a way that the assignment induced by the decision tree falsifies a clause. At the end of the Dymond–Tompa game, all of the predecessors of the challenged pebble have pebbles. By item 1 it is false, and by item 3 its predecessors are queried. By item 2, its predecessors are true, therefore we can label the leaf with the clause claiming that if the predecessors of the challenged vertex are true then the challenged vertex is true.

□

B.6 Dymond–Tompa Trade-offs

In this section we prove upper and lower bounds for the Dymond–Tompa game on graphs of a given family. The lower bounds are the final missing piece in order to get length-space trade-offs for cutting plane proofs, and the upper bounds will be used to obtain space-efficient proofs, as explained in Section B.7.

Our goal is to prove the following lemma.

Lemma B.6.1. *For any $n, d \in \mathbb{N}^+$ such that n is a power of 2, there exists an explicitly constructible DAG $G(n,d)$ of depth d with $\mathrm{O}(dn)$ vertices and indegree at most 2 such that:*

1. *for any $r \leq d$, the cost of an r-round DT game is at most* $\min\{r2(2^{\lceil d/r\rceil}-1), rn(2^{\lceil\lceil\log d\rceil/r\rceil}-1)\}$*;*

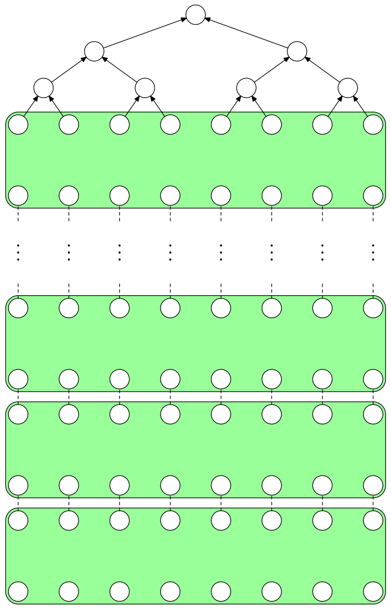
2. *for any $r \leq d$, the cost of an r-round DT game is at least* $\min\{\frac{r2^{d/r}}{8}, \frac{n}{8}\}$.

We first define a family of graphs for which we will prove the lemma.

Definition B.6.2 (Butterfly graph). A *k-dimensional butterfly graph G* is a DAG with vertices labelled by pairs (w,i) for $0 \leq w \leq 2^k-1$ and $0 \leq i \leq k$, and with edges from vertex (w,i) to $(w',i+1)$ if the binary representations of w and w' are equal except for possibly in the $(i+1)$st most significant bit. Note that G has $(2^k+1)k$ vertices, has 2^k sources and 2^k sinks, and that all vertices that are not sources have indegree two.

Moreover, if H is a graph with n sinks and n sources, we say a graph is a stack of s Hs, if it consists of s copies of H such that sources on level i are identified with sinks on level $i+1$ for $i \in \{1,\ldots,s-1\}$.

For any $n, d \in \mathbb{N}$ such that n is a power of 2, the graph $G(n,d)$ we will consider for the Dymond–Tompa game consists of a (possibly fractional) stack of butterfly graphs of dimension $\log n$, with an attached binary tree on top such that the depth of this graph is exactly d (see Figure B.10a). Note that if d is a multiple of $\log n$, then this graph has exactly $d/\log n$ blocks (the 1st block is a binary tree). Moreover, if $d \leq \log n$, then $G(n,d,)$ is just a binary tree of depth d. Observe that, if $d \geq \log n$, $G(n,d)$ has $(d-\log n)n+2n-1$ vertices.


(a) Stack of graphs with binary tree on top (dashed lines represent vertex identification)

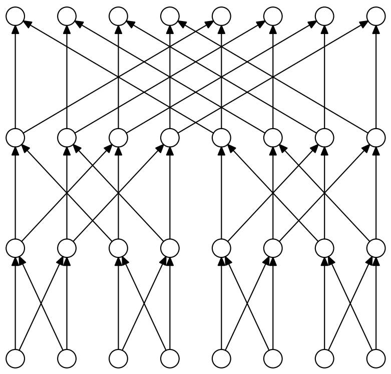

(b) 3-dimensional butterfly graph

Figure B.10: Stack of butterflies

B.6.1 Upper Bounds for the Cost of the Dymond–Tompa Game on Butterfly Graphs

Given a graph G, we say T is a Pebbler strategy for G if the following holds.

1. Each node $x \in V(T)$ is labelled with a set of vertices $S(x) \subset V(G)$ (corresponding to a valid set of vertices where Pebbler can place pebbles at the current stage of the game). We note that in order for $S(x)$ to be a valid move for Pebbler at node x, it must be the case that if P is the path from the root of T to x, then $S(x) \cap (\bigcup_{y \in V(P)} S(y)) = \emptyset$ (Pebbler cannot repebble a vertex) and $S(x) \neq \emptyset$ if any vertex in $\bigcup_{y \in V(P)} S(y)$ has an immediate predecessor that is unpebbled (if the game has not ended, Pebbler must place at least one pebble).

2. Each edge leaving a node $x \in V(T)$ is labelled with a set of vertices $S_{xy} \subseteq S(x) \cup S_{p(x)x}$ of possible Challenger moves (corresponding to pebbles that Challenger challenges and that lead to the same Pebbler strategy), where $p(x)$ is the parent of x in T. In the case where x is the root of the tree, define $S_{p(x)x} = \emptyset$. In order for T to be a complete strategy, i.e., for T to describe how to deal with all possible Challenger moves, it must be the case that at every node x either

$\bigcup_{y:xy\in E(T)} S_{xy} = S(x) \cup S_{p(x)x}$ or all immediate predecessors of $S(x) \cup S_{p(x)x}$ are pebbled, and in this latter case x is a leaf.

Proposition B.6.3. *If there is a winning Pebbler strategy tree T with max degree a, depth d and such that the label of each node is of size at most b, then the cost of a r-round DT game, for any $r \leq d$, is at most $rb(a^{\lceil d/r \rceil} - 1)$.*

Proof. Pebbler's strategy will be as follows: at round i Pebbler will be playing according to the strategy tree T_i. At the first rounds, let $T_1 = T$. Let $r \leq d$. For every $i \leq r$, let T_i' be a subtree of T_i consisting of all nodes at distance less than $\lceil d/r \rceil$ of the root. Note that there are at most $a^{\lceil d/r \rceil} - 1$ nodes in T_i'. At rounds i, Pebbler places pebbles on all the vertices that are in some label of nodes in $V(T_i')$. Since each node has at most b labels, Pebbler places at most $b(a^{\lceil d/r \rceil} - 1)$ pebbles at each rounds. If Challenger challenges a vertex v that does not have all its immediate predecessors pebbled, then v must be in the label of some edge xy where x is a leaf of T_i'. Let T_{i+1} be the subtree of T_i having y as root.

This is a valid strategy for Pebbler and since for every i, the depth of T_{i+1} is $\lceil d/r \rceil$ smaller than the depth of T_i and $r\lceil d/r \rceil \geq d$, the game will end in at most r rounds. Thus the total number of pebbles placed is at most $rb(a^{\lceil d/r \rceil} - 1)$. □

We state an immediate corollary of Proposition B.6.3, which is to weak to imply the upper bounds we stated, but has the advantage that it doesn't depend on strategy trees and might be useful for other purposes.

Corollary B.6.4. *If Pebbler has a winning strategy in d rounds and x pebbles per round, then the cost of a r-round DT game, for any $r \leq d$, is at most $rx((x+1)^{\lceil d/r \rceil} - 1) \leq r(x+1)^{\lceil d/r \rceil + 1}$.*

All that is left to prove the upper bounds is to show that there exists Pebbler strategy trees with certain properties. We prove two propositions below which together with Proposition B.6.3 implies the upper bounds in Lemma B.6.1

Proposition B.6.5. *There is a Pebbler strategy tree T for the graph $G(n,d)$ with max degree 2, depth d and such that the label of each vertex is of size at most 2.*

The proof follows from the following straightforward claim.

Claim B.6.6. *For any graph with depth d and indegree at most α, there is a winning Pebbler strategy in d rounds and using at most α pebbles per round. Moreover, this strategy is such that if Challenger stays the game immediately ends.*

Proof. The Pebbler strategy is simply to, at every round, pebble all in-neighbours of the challenged vertex. □

Proposition B.6.7. *There is a winning Pebbler strategy tree T for the graph $G(n,d)$ with max degree 2, depth $\lceil \log d \rceil$ and such that the label of each vertex is of size at most n.*

Proof. The winning Pebbler strategy is to do a binary search in the rows of $G(n,d)$. The strategy only depends on whether Challenger stays or moves, but does not depend on what particular pebble Challenger chooses to pebble. Thus the Pebbler strategy tree T has max degree 2. The proposition then follows from the fact that $G(n,d)$ has depth d and has at most n vertices per row. □

B.6.2 Lower Bounds for the Cost of the Dymond–Tompa Game on Butterfly Graphs

Now we would like to show that the strategies described in the previous subsection are essentially the best Pebbler can do. As a warm up, and to give some intuition on the strategy, we prove a special case of Lemma B.6.1. In order to keep the proof simple, we use the alternative definition of the Dymond–Tompa game and consider a stack of butterflies with an extra vertex on top, $\widehat{G}$, as defined in Section B.5.

Lemma B.6.8. *For any $n, r \in \mathbb{N}^+$ such that n is a power of 2, there exists an explicitly constructible DAG $\widehat{G}(n, r\log n)$ of depth $r\log n + 1$ with $\mathrm{O}(nr\log n)$ vertices and indegree at most 2 such that for any $r \leq d$, the cost of an r-round DT game is at least $\frac{n}{4}$.*

As we observe further on, this lemma holds not only for stacks of butterfly graphs, but also for stack of other kinds of graphs, provided Claim B.6.9 holds.

Throughout the Dymond–Tompa game, we say a vertex t is reachable from s if there is a path from s to t with no pebbles neither on internal vertices of the path nor on the vertex s (but t may be pebbled). We say a sink at level ℓ is *good* if it is unpebbled and is reachable by at least $n/2+1$ sources at level ℓ. Furthermore, we say a source s is disconnected from a sink t if there is no completely (including end points) unpebbled path from s to t, and we consider the number of source-sink disconnections in a graph as the number of pairs (t,s) such that s is disconnected from t. The proof uses the following claim.

Claim B.6.9. *Given a butterfly graph with n sinks, if at most $n/4-1$ vertices are removed, there still are at least $n/2+1$ good sink-vertices.*

Proof of Claim B.6.9. If there are less than $n/2+1$ good sink-vertices, then the number of source-sink disconnections is at least $n/2 \cdot n/2$ (at least $n/2$ non-good sinks are not reached by at least $n/2$ sources). Note that any vertex in a butterfly graph is in exactly n distinct source-sink paths So if a is the number of vertices removed, then there are at most an source-sink disconnections. This implies that $a \geq n/4$. □

Note that the proof above goes through for any graph that satisfy the following properties: the graph has n sources and sinks; every source can reach every sink; and removing any set of at most a vertices causes at most an source-sink disconnections.

We can now proceed to the proof of the warm-up lemma.

Proof of Lemma B.6.8. We give a strategy for Challenger in the Dymond–Tompa game over the graph $\widehat{G}(n, r \log n)$ defined above so that, assuming Pebbler has at most $n/4-1$ pebbles, the game will not end within r rounds.

At a high level, Challenger's strategy will be to keep in mind, before every round, a good sink that can reach the challenged vertex; more precisely, before round $\ell+1$ Challenger will have a good sink at level ℓ in mind, say t_ℓ. After the Pebbler places pebbles on the graph, Challenger chooses a good sink at level $\ell+1$ that is reachable from t_ℓ and decides to have that in mind. He will then check if there are any new pebbles that are causing the challenged vertex to be unreachable from t_ℓ, and if so, challenges one that is reachable from t_ℓ.

The proof goes as follows. We maintain the invariant that before round ℓ, Challenger's chosen vertex t_ℓ is a good sink at level ℓ and reaches the challenged vertex. Before the first rounds, the challenged vertex is the sink of $\widehat{G}$ (the vertex that is not in G) and Challenger's chosen vertex is the original sink of G, t_1, which clearly is a good sink at level 1 (it is unpebbled and reachable from all sources at level 1) and reaches the challenged vertex.

Suppose that the invariant was true until round ℓ, i.e. suppose that before Pebbler's $(\ell-1)$st move, Challenger's chosen vertex $t_{\ell-1}$ is a good sink at level $\ell-1$ and reaches the challenged vertex. At round $\ell-1$, Pebbler places some pebbles. Since Pebbler has at most $n/4-1$ pebbles in total, we can conclude by Claim B.6.9 that there are at least $n/2+1$ good sinks at level ℓ. Since $t_{\ell-1}$ was a good sink before round $\ell-1$, there must be a good sink at level ℓ, say t_ℓ, which was reachable from $t_{\ell-1}$ before round $\ell-1$. Challenger decides t_ℓ will be the next chosen vertex. Since before round $\ell-1$, t_ℓ reached $t_{\ell-1}$ and $t_{\ell-1}$ reached the challenged vertex, the only possible pebbles that are disconnecting t_ℓ from the challenged vertex are the newly put pebbles. If there are no such blocking pebbles, i.e., if t_ℓ reaches the challenged vertex, Challenger stays. If there are newly put pebbles that block all paths from t_ℓ to the challenged vertex, Challenger challenges one that is reachable from t_ℓ. Thus, before round ℓ, Challenger's chosen vertex t_ℓ is a good sink at level ℓ and reaches the challenged vertex, and the invariant is maintained.

We conclude that before round $(r+1)$, Challenger's chosen vertex t_{r+1} is an unpebbled vertex global source (which would have been a good sink at level $r+1$, if such a level had existed) that reaches the challenged vertex, and hence the game has not ended. □

Now to prove the lower bound in Lemma B.6.1 in its full generality, we must allow any number of rounds (at most the depth) and still get a good bound on the cost of the game. We again describe a strategy for Challenger; the difference is that Challenger cannot afford to jump $\log n$ rows every round. Intuitively, we do not think of the graph as a stack of blocks, but as a continuous block such that any consecutive $\log n$ rows is isomorphic to a butterfly graph.

Note that given any vertex $v \in V(G(n,d))$ at distance d' from the set of sources, the subgraph induced by all vertices that reach v is isomorphic to $G(n,d')$. We therefore refer to the top binary tree of the subgraph $G(n,d')$ as the tree induced by the vertices that reach v and are at distance at most $\log n$ from v.

We give a more general definition of a good vertex and define a partially good vertex. Let T be a complete directed binary tree rooted at v. We say v (or T) is good if v can be reached by strictly more than half of the leaves. If T has n leaves this is equivalent to requiring that, for any $h' \le \log n$, v can be reached by strictly more than $2^{h'}/2$ vertices at distance h' from v. Given a vertex $u \in V(T)$ at distance h from the leaves, we say u (or the subtree of T rooted at u) is T-partially good if, for any $h' \le h$, u can be reached by strictly more than $2^{h'}/2$ vertices at distance h' from u. When T is clear from the context, we just say u is partially good.

We are now ready to prove the lower bound.

Proof of Lemma B.6.1, item 2. We actually prove something stronger: we allow Pebbler to place some pebbles before the game begins, provided that the top binary tree remains good. We charge only for the pebbles placed outside the binary tree. Challenger is not allowed to challenge any pebble that was placed in this initial stage. We denote this game DT*.

Formally, we prove the following claim. Given a graph G and a challenged vertex on this graph, if there is a vertex v in G that reaches the challenged vertex and that is the sink of a graph $G(n,d)$, then cost of the r-round DT* game on G is at least $\min\{\frac{\alpha 2^{d/\alpha}}{8}, \frac{n}{8}\}$, where $\alpha = \min\{d, r\}$.

We prove this claim by induction on α. For $\alpha = 1$, either $d > r = 1$ or $d = 1$. If $d > r = 1$, $G(n,d)$ consists of at least a binary tree of depth $d' = \min\{d, \log n\}$ with $2^{d'+1} - 1$ vertices and such that the sink reaches the challenged vertex. Since after Pebbler places the initial pebbles the binary tree must be a good tree, at least half of the tree reaches the challenged vertex (actually, strictly more than half of the pebbles in every row must reach the challenged vertex, which makes a total of at least $2^{d'} + d'$ vertices that reach the challenged vertex). Clearly Pebbler must pebble all the vertices that reach the challenged vertex in order to finish the game in one round, therefore the cost is more than $\min\{\frac{2^d}{8}, \frac{n}{8}\}$. If $d = 1$, then clearly at least 1 pebble is needed and $1 \ge 1/4 = \alpha 2^{d/\alpha}/8$, so the base case holds.

Now suppose $\alpha \ge 2$ and that Pebbler has placed some initial pebbles on the graph, but maintaining the top binary tree good. Pebbler then starts the first round by placing some pebbles. Let $x \ge 1$ be the number of pebbles Pebbler placed in the top binary tree in the first round (note we are not counting the initial pebbles placed before the game began). If $d \le \lceil \log 4x \rceil$ (i.e., if the graph is shallow or if Pebbler placed too many pebbles), the claim holds since this implies $x \ge \frac{2^d}{8}$ and clearly $\frac{2^d}{8} \ge \frac{2^{d/\alpha + \log \alpha}}{8} = \frac{\alpha 2^{d/\alpha}}{8}$, for any α and d that satisfy $2 \le \alpha \le d$. We thus assume $d > \lceil \log 4x \rceil$.

Note that the row that is at distance $\lceil \log 4x \rceil$ from the root of the top binary tree has exactly $y = 2^{\lceil \log 4x \rceil} \ge 4x$ vertices. Before the first round, at least $y/2$ of these vertices were partially good (with respect to the top binary tree). Since $x \le y/4$ pebbles were placed, at least $y/4$ of these partially good trees were untouched at this round. We will show that, provided that there are less than $n/8$ pebbles in the graph, then at least one of these partially good trees is totally good.

Fix a set of $y/4$ partially good trees that were untouched at this round. If $d \leq \log n$, then the partially good trees are all totally good. If $d > \log n$, we consider the set S of all the (pebbled or unpebbled) vertices at distance $\log n$ from the root of the top binary tree that are in one of these $y/4$ partially good trees. Since these trees are disjoint there are at least $y/4 \cdot (n/y) = n/4$ such vertices. Consider the block consisting of vertices at distance at most $\log y$ from S. Note that the number of source-sinks paths in this block is at least $n/4 \cdot y$ and any vertex in this block is in at most y such paths. Therefore, if there are less than $n/8$ pebbles, then there are more than $ny/8$ unpebbled source-sink paths in this block. This means that at least one of the $y/4$ partially good tree has more than $n/2$ unpebbled source-sink paths in this block, which implies that it is a totally good tree.

Let v be the root of this totally good tree. We know that v reached the challenged vertex before this rounds. This implies that if v no longer reaches the challenged vertex then there are newly placed pebbles blocking a path between v and the challenged vertex. If this is the case, Challenger challenges a newly placed pebble that is in such a path and that is closest to v (i.e., is reachable from v). The graph induced by all vertices that reach v satisfies the invariants and has depth $d - \log y \geq d - \log 8x$. We observe that, the x pebbles we account for at this round were placed on the binary tree, and therefore are not counted again when applying the induction hypothesis.

If $r - 1 > d - \log y$ (i.e., the number of rounds left is larger than the depth of the remaining subgraph), we argue that claim follows. To show this, we consider two cases: $r < d/2$ and $d/2 \leq r$. In the first case, we show that the number of pebbles placed has to be large since in one round the depth of the remaining subgraph was reduced by a lot. Note that $r < d/2$ this implies that $d - \log y < d/2 - 1$ and thus $8x \geq y > 2^{d/2+1}$. Moreover, $\alpha = \min\{r, d\} = r < d/2$ and since $d/\alpha + \log \alpha$ is a monotone decreasing function for $\alpha \in [2, d/2]$, we have that $8x > 2^{d/2+1} \geq 2^{d/\alpha + \log \alpha}$ and the claim follows. In the second case, the intuition is that the claim we want to prove is not so strong. Indeed, note that $d/2 \leq r$ implies that $d/2 \leq \alpha = \min\{r, d\} \leq d$, and thus $2^{d/\alpha + \log \alpha} \leq 2d$, so it is enough to show that at least $2d/8$ pebbles are needed. Applying the induction hypothesis on the subgraph induced by all vertices that reach v we have that the cost of this subgraph is at least $\min\{\frac{2(d - \log y)}{8}, \frac{n}{8}\}$. The claim follows by noting that $8x + 2(d - \log y) \geq 2d + 8x - 2\log 8x \geq 2d$, where the last inequality follows since $x \geq 1$.

We thus assume $r - 1 \leq d - \log y$ (which implies $r \leq d$ and $\alpha = r$), and apply the induction hypothesis on the subgraph induced by all vertices that reach v to get that the cost of this subgraph with one less round is at least $\min\{\frac{(r-1)2^{(d - \log 8x)/(r-1)}}{8}, \frac{n}{8}\}$.

It suffices to show that

$$(r-1)2^{(d - \log 8x)/(r-1)} + 8x \geq r2^{d/r}.$$

Let $a = \frac{d}{r} - \frac{d-\log 8x}{r-1}$, so that $8x = 2^{d/r+a(r-1)}$. Rewriting the equation above we have

$$\begin{aligned}(r-1)2^{(d-\log 8x)/(r-1)} + 8x &= (r-1)2^{d/r-a} + 2^{d/r+a(r-1)} \\ &= r2^{d/r}\left(\frac{(r-1)2^{-a}}{r} + \frac{2^{a(r-1)}}{r}\right) . \qquad \text{(B.6.1)}\end{aligned}$$

Note that, for any $r \geq 1$, $(r-1)2^{-a} + 2^{a(r-1)} \geq r$. Indeed, for any fixed $r \geq 1$, a straightforward calculation shows that the real function $f(a) = (r-1)2^{-a} - r + 2^{a(r-1)}$ is minimized at $a = 0$ and $f(0) = 0$. This concludes the proof. □

To conclude this section, we define a more general class of previously studied graphs and prove that Lemma B.6.8 also applies for stacks of any graph in this class.

Definition B.6.10 (Connector). A *n-connector* is a DAG with n sources S and n sinks T, and that satisfies the following property: for any subsets $S' \subseteq S$ of sources and $T' \subseteq T$ of sinks of size $|S'| = |T'|$ and for any specification M of which source in S' should be connected to which sink in T' (one-to-one correspondence), it holds that there are $|S'|$ vertex-disjoint paths between S' and T' satisfying M.

As mentioned previously, for the proof of Lemma B.6.8 for $r = d/\log n$ the only property of the graph that is needed is that each stack satisfies Claim B.6.9. We show that a n-connector does satisfy the properties required by Claim B.6.9.

Proposition B.6.11. *A n-connector satisfies the following two properties:*

1. *any source can reach any sink;*
2. *the removal of any set of a vertices causes at most an source-sink disconnections, i.e., the sum over all sinks v of the number of sources that cannot reach v is at most an.*

Proof. Let G be a n-connector. Obviously G satisfies property 1. Let A be any set of vertices in G. Let $a = |A|$. We will show that the removal of A causes at most an source-sink disconnections, thus concluding that G also satisfies property 2. Let G' be the graph that results from G after the removal of A.

Let $H = ((S, T), E)$ be a bipartite graph, where S correspond to the sources in G and T to the sinks, and there is an edge (s, t) if source s doesn't reach sink t in G'. Let q' be the size of a maximum matching in H. This implies that H has a vertex cover of size q' (Kőnig's theorem). Since every vertex in H has degree at most n, we conclude that H has at most $q'n$ edges, which means that A caused at most $q'n$ source-sink disconnections.

Suppose $q' > a$, and let $M = \{(s_1, t_1), (s_2, t_2), \ldots, (s_{a+1}, t_{a+1})\}$ be a matching of size $a + 1$ in H. Given the set $S' = \{s_1, s_2, \ldots, s_{a+1}\}$ of sources, the set $T' = \{t_1, t_2, \ldots, t_{a+1}\}$ of sinks and M as the specification of which source should be connected to which sink, we have that in G there are $a + 1$ disjoint paths connecting S' to T' according to M. But this is a contradiction, since all paths must contain a vertex in A.

Therefore, we conclude that $q' \leq a$ and A caused at most $rn \leq an$ source-sink disconnections. □

Interestingly, butterfly graphs and connectors also relate in that connecting two k-dimensional butterfly graphs in a certain back-to-back fashion gives a 2^k-connector. A description of this construction and a proof of this fact can be found, e.g., in [146].

It is known [154] that n-connectors require $\Omega(n \log n)$ edges. However, the following question remains open: Does there exist an explicitly constructable family $\{G_n\}$ of DAGs with n sources and sinks and of size linear on n such that, for any $r \in \mathbb{N}^+$, the cost of an r-round DT game on a stack of r graphs G_n is also linear on n?

B.7 Upper Bounds for Size and Space

We prove the upper bounds in terms of the weaker *resolution* proof system. A resolution configuration $\mathbb{C}$ is a set of clauses. A resolution refutation of a CNF formula F is a sequence of configurations $\mathbb{C}_0, \ldots, \mathbb{C}_\tau$ such that $\mathbb{C}_0 = \emptyset$, the empty clause $\bot \in \mathbb{C}_\tau$, and for $t \in [\tau]$ we obtain $\mathbb{C}_t$ from $\mathbb{C}_{t-1}$ by one of the following steps:

Axiom download $\mathbb{C}_t = \mathbb{C}_{t-1} \cup \{C\}$, for $C \in F$.

Inference $\mathbb{C}_t = \mathbb{C}_{t-1} \cup \{C \vee D\}$, where $C \vee D$ is inferred by the resolution rule

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D} \; .$$

Erasure $\mathbb{C}_t = \mathbb{C}_{t-1} \setminus \{C\}$, for some $C \in \mathbb{C}_{t-1}$.

The length of a refutation is the number of axiom downloads and inferences. The line space of a configuration is the number of clauses, and the total space is the number of literals. The (line/total) space of a refutation is the maximum over all configurations.

It is easy to see that cutting planes can simulate the resolution rule using at most w additions and one division, where w is the width of the shortest clause, and therefore a resolution refutation in length L, width w and space s gives a cutting planes refutation in size $O(w^2 L)$ and space $s + 1$ where the largest coefficient is 2. The refutation that we construct in Lemma B.7.3 is of constant width, so cutting planes can simulate it with constant overhead, and in Lemma B.7.7 it is not but we can ignore polynomial factors.

The search depth of a formula F is the minimum number of queries of a decision tree for the search problem of F. As observed in [135, 26], a search tree for the falsified clause search problem is equivalent to a tree-like resolution refutation. We can construct a refutation essentially by replacing each internal node labelled with a variable x in the search tree with the result of resolving its two children over the variable x. It is straightforward to check that this is indeed a valid resolution refutation.

Lemma B.7.1 ([77]). *If a CNF formula has search depth d, then it has a refutation in length 2^d, width d, and space d simultaneously.*

Proof. Consider the refutation tree T equivalent to a minimal depth search tree. Traversing the refutation tree in depth-first order it is straightforward to reconstruct a tree-like refutation of length $|T| \le 2^d$, width d, and space d, where $|T|$ is the order of T. □

To show length upper bounds we simulate a black pebbling in resolution and then lift that refutation, as done in for instance [28].

The *black pebble game* is played by a single player on a DAG. The allowed moves are to place a pebble on a vertex if its predecessors have pebbles and to remove a pebble from any vertex. A pebbling is a sequence of moves that begin with the empty graph and end with a pebble on the sink. The number of moves of a pebbling is called the time, and the maximum number of pebbles on the graph at the same time the space. An excellent survey of pebbling up to ca 1980 is [153], and some more recent developments are covered in the upcoming survey [146].

Lemma B.7.2. *If there is a black pebbling for an indegree* 2 *graph G in space s and time τ, then there is a resolution refutation of Peb_G in length* $O(\tau)$*, width* 3*, and total space* $O(s)$*.*

Proof. We build a refutation π of Peb_G by keeping in memory the unit clause v for every vertex v that has a pebble. This is trivial for sources because these clauses are already axioms. For a vertex v with predecessors u_1 and u_2, when we place a pebble over v its predecessors have pebbles, therefore the clauses u_1 and u_2 are in memory. We download the axiom $\overline{u_1} \lor \overline{u_2} \lor v$, resolve it with u_1 and u_2 to obtain the clause v, and then delete intermediate clauses. □

We can use a generic procedure to transform any refutation into a refutation for the corresponding lifted formula (see Lemma 4.3 in the ECCC version of [28]). However, we obtain better upper bounds if we take the structure of the refutation into account.

Lemma B.7.3. *Let G be a graph of indegree* 2 *with a black pebbling in space s and time τ. Then there is a refutation of $Lift_\ell(Peb_G)$ in size* $O(\tau \cdot \ell^3)$ *and total space* $O(s \cdot \ell)$*.*

Proof. Let π be the refutation of Peb_G given by Lemma B.7.2. We build a refutation π' of $Lift(F)$ by deriving, for each unit clause v, the ℓ clauses $Lift(v)$. This is trivial in the axiom download and erasure cases, and we are left with inference. The only inference steps we need to deal with are of the form

$$\dfrac{\dfrac{\overline{u_1} \lor \overline{u_2} \lor v \qquad u_1}{\overline{u_2} \lor v} \qquad u_2}{v} \tag{B.7.1}$$

and we handle both inference steps at once.

Recall that for a lifted formula to have constant width we have to split the wide auxiliary clauses (B.2.3a), introducing extension variables, but we were not explicit about how to do that. We split the clause $\bigvee_{a=1}^{\ell} x_{a,u}$ into a clause $x_{1,u} \lor s_{2,u}$, $\ell - 2$ clauses of the form $\overline{s}_{a,u} \lor x_{a+1,u} \lor s_{a,u}$, and a clause $\overline{s}_{\ell,u} \lor x_{\ell,u}$.

$$\dfrac{\dfrac{\dfrac{\overline{x}_{1,u_1}\lor\overline{y}_{1,u_1}\lor B \quad \overline{x}_{1,u_1}\lor y_{1,u_1}}{\overline{x}_{1,u_1}\lor B} \qquad x_{1,u_1}\lor s_{2,u_1}}{s_{2,u_1}\lor B} \qquad \dfrac{\dfrac{\overline{x}_{2,u_1}\lor\overline{y}_{2,u_1}\lor B \quad \overline{x}_{2,u_1}\lor y_{2,u_1}}{\overline{x}_{2,u_1}\lor B} \qquad \overline{s}_{2,u_1}\lor x_{2,u_1}\lor s_{3,u_1}}{\overline{s}_{2,u_1}\lor s_{3,u_1}\lor B}}{s_{3,u_1}\lor B}$$

$$\vdots \qquad\qquad\qquad\qquad \vdots$$

$$\dfrac{s_{\ell,u_1}\lor B \qquad\qquad\qquad \overline{s}_{\ell,u_1}\lor B}{B}$$

Figure B.11: Simulation of a pebbling step

First we fix a clause $C \in \mathit{Lift}(v)$ that we want to derive. Then we fix a clause $B \in \mathit{Lift}(\overline{u_2} \lor v)$ that contains C as a subclause. We can derive B by resolving the clauses $\overline{x}_{a,u_1} \lor \overline{y}_{a,u_1} \lor B$, which are actual axioms of $\mathit{Lift}(\mathit{Peb}_G)$, first with $\overline{x}_{a,u_1} \lor y_{a,u_1}$, which are in memory by hypothesis, and then with the axioms $\overline{s}_{a,u_1} \lor x_{a,u_1} \lor s_{a+1,u_1}$ that result of breaking $\bigvee_{a=1}^{\ell} x_{a,u_1}$ into clauses of constant width. See Figure B.11 for details. Such a derivation requires $O(\ell)$ steps and constant space.

We repeat this procedure for all of the ℓ clauses in $B \in \mathit{Lift}(\overline{u_2} \lor v)$ that contain C as a subclause, using at most $O(\ell^2)$ steps and space $\ell + O(1)$. Now we have all the clauses required to derive C by repeating the above procedure with the clauses $\mathit{Lift}(\overline{u_2}) \lor C$ that we just derived, the clauses $\overline{x}_{a,u_1} \lor y_{a,u_1}$, which are also in memory by hypothesis, and the axioms $\overline{s}_{a,u_1} \lor x_{a,u_1} \lor s_{a+1,u_1}$. Such a derivation requires an additional $O(\ell)$ steps and constant additional space, for a total of $O(\ell^2)$ steps and space $\ell + O(1)$. Finally we repeat the whole procedure ℓ times, once for each clause $C \in \mathit{Lift}(v)$, for a total of $O(\ell^3)$ steps and space $2\ell + O(1)$.

Observe that all clauses are of constant width, so the size and total space are also $O(\ell^3)$ and $O(\ell)$, and furthermore we can simulate the resolution proof in cutting planes with constant overhead. $\square$

If we only care about optimizing size, then a strategy that places pebbles in topological order and never removes a pebble is a valid pebbling of any graph of order m in time m and space m, which gives a short refutation in size $O(m\ell^3)$ and space $O(m\ell)$.

Lemma B.7.4. *Let G be a graph of order m and indegree 2. For any $\ell \geq m^3$ there is a refutation of $\mathit{Lift}_\ell(\mathit{Peb}_G)$ in size $O(N)$ and total space $O(N^{2/5})$, where $N = \Theta(m\ell)$ is the size of $\mathit{Lift}_\ell(\mathit{Peb}_G)$.*

In terms of space, even the most space-efficient pebbling strategy would give a refutation in space $O(\ell)$, which is too weak. Therefore to obtain good space upper bounds we go through the Dymond–Tompa game and search depth instead of black pebbling. The following Lemma follows from Lemma B.5.1 and was first proved in [54].

Lemma B.7.5 ([54]). *If there is a Dymond–Tompa pebbling strategy for a graph G in space s, then the formula Peb_G has search depth s.*

If we lay out the extension variables so that their indices form an ordered binary tree and attach two nodes labelled $x_{a,u}$ and $x_{a+1,u}$ to each leaf $s_{a,u}$ we get a search tree that finds a selector variable set to true by any assignment that respects auxiliary clauses. We can use this tree to build search trees for a lifted formula.

Lemma B.7.6. *Given a CNF formula F of search depth d, the lifted formula $Lift_\ell(F)$ has search depth $d \log \ell$.*

Proof. Given a decision tree T_1 for the falsified clause search problem on F of depth d and a decision tree T_2 that finds a selector variable set to true of depth $\log \ell$, we build a decision tree T_3 for the falsified clause search problem on $Lift_\ell(F)$ of depth $d \log \ell$ by composing the trees as follows.

First we modify T_2. We reinterpret the leaves as queries to selector variables $x_{a,u}$, and we attach two new nodes to every selector variable query. We label the 0-leaf of $x_{a,u}$ with the falsified clause $\overline{s}_{a,u} \vee x_{a,u} \vee s_{a+1,u}$, and we label the 1-node with the main variable $y_{a,u}$. We add two unlabelled leaves to the $y_{a,u}$ node.

Then, starting at the root of T_1, we apply the following recursive procedure. If the root is an inner vertex labelled with a variable u, then we add a copy of T_2 that queries variables corresponding to u. To each 0-leaf we attach the result of this procedure on the 0-subtree of T_1, and to each 1-leaf we attach the result of this procedure on the 1-subtree.

Finally, for each leaf of T_3 that we did not label yet, there is a corresponding leaf in T_1 labelled with a clause C. C is falsified by the assignment α induced by the branch leading to C. By construction, the assignment β induced by the branch in T_3 respects auxiliary clauses and, for every variable $u \in Vars(C)$ it sets $x_{a,u} = 1$ and $y_{a,u} = \alpha(u)$ for some $a \in [\ell]$. Therefore we can label the leaf of T_3 with the main clause $\bigvee_{u \in Vars(C)} \overline{x}_{a,u} \vee y_{a,u}^{1-\alpha(u)}$. □

Lemma B.7.7. *Let G be a graph of order m and indegree 2 with Dymond–Tompa price s. For any $\ell \geq m^3$ there is a refutation of $Lift_\ell(Peb_G)$ in size $2^{O(s \log N)}$ and space $O(s \log N)$, where $N = \Theta(m\ell^3)$ is the size of $Lift_\ell(Peb_G)$.*

Proof. This follows immediately from Lemmas B.7.5, B.7.6, and B.7.1. □

B.8 Putting the Pieces Together

By the technical result proved in Section B.2, Theorem B.2.8, we know that if G is a graph over m vertices such that the r-round Dymond–Tompa game on G costs $\Omega(c)$, then for $\ell = m^{3+\epsilon}$, $Lift_\ell(Peb_G)$ is a 6-CNF formula over $\Theta(m^{4+\epsilon})$ variables and $N = \Theta(m^{10+3\epsilon})$ clauses such that for any CP refutation of $Lift_\ell(Peb_G)$ even with coefficients of unbounded size in formula space less than $\frac{c}{r}\log N$ requires length greater than $2^{\Omega(r)}$. This fact together with the lower and upper bounds proven in Sections B.6 and B.7 yield the following theorem.

Theorem B.8.1. *There is an explicitly constructible two-parameter family of unsatisfiable 6-CNF formulas $F(n,d)$, for $n,d \in \mathbb{N}^+$, of size $N = \Theta((dn)^{10+\epsilon})$ such that:*

1. *$F(n,d)$ can be refuted by CP with small coefficients in size $\mathrm{O}(N)$ and total space $\mathrm{O}(N^{2/5})$.*
2. *$F(n,d)$ can be refuted by CP with small coefficients in total space $\mathrm{O}(d\log N)$ and size $2^{\mathrm{O}(d\log N)}$.*
3. *For any $r \leq d$, any CP refutation even with coefficients of unbounded size of $F(n,d)$ in formula space less than $\min\{\frac{2^{d/r}\log N}{8}, \frac{n\log N}{8r}\}$ requires length greater than $2^{\Omega(r)}$.*

Proof. Let G be a stack of depth d of butterfly graphs of dimension $\log n$ which has a total of $\Theta(dn)$ vertices. Let $F(n,d) = \mathit{Lift}_\ell(\mathit{Peb}_G)$.

Item 1 follows directly from Lemma B.7.4. Item 2 follows from setting $r = d$ in the upper bound stated in part 1 of Lemma B.6.1 and combining it with Lemma B.7.7.

By Lemma B.6.1 we get that for any $r \leq d$, the r-round Dymond–Tompa game played on G has cost at least at least $\min\{\frac{r2^{d/r}}{8}, \frac{n}{8}\}$. Thus, by Theorem B.2.8, we get item 3. □

Choosing the right values for d and r in Theorem B.8.1, we get the following to corollaries. These are generalizations of Theorems B.1.1 and B.1.2.

Corollary B.8.2. *For any positive constant K, there exists a family of 6-CNF formulas $\{F_N\}_{N=1}^{\infty}$ of size $\Theta(N)$ such that:*

1. *F_N can be refuted by CP with small coefficients in size $\mathrm{O}(N)$ and total space $\mathrm{O}(N^{2/5})$.*
2. *F_N can be refuted by CP with small coefficients in total space $\mathrm{O}(\log^{K+2} N)$ and size $2^{\mathrm{O}(\log^{K+2} N)}$.*
3. *Any CP refutation even with coefficients of unbounded size of F_N in formula space less than $N^{1/10-\epsilon}$ requires length greater than $2^{\Omega(\log^K N)}$, for any constant $\epsilon > 0$.*

Proof. The proof follows from setting $d = \log^{K+1} n$ and $r = d/\log n$ in Theorem B.8.1 and for every N, choosing n to be a power of 2 such that N is at most a factor off from $(nd)^{10+\epsilon}$.

We note that $\log N = \Theta(\log n)$, so 2 holds. Moreover, $N = \mathrm{o}((nd)^{10+2\epsilon})$, thus $N^{1/10-\epsilon} = \mathrm{o}((nd)^{(10+2\epsilon)(1/10-\epsilon)})$ and

$$\begin{aligned}(nd)^{(10+2\epsilon)(1/10-\epsilon)} &= (n\log^{K+1} n)^{(10+2\epsilon)(1/10-\epsilon)} \\ &\leq (n \cdot n^{\epsilon})^{(1-9\epsilon)} < n^{(1-8\epsilon)} \\ &< \frac{n}{8\log^K n} < \frac{n}{8r}\log N,\end{aligned}$$

and therefore 3 also holds. □

Corollary B.8.3. *For any positive constant K, there exists a family of 6-CNF formulas $\{F_N\}_{N=1}^{\infty}$ of size $\Theta(N)$ such that:*

1. *F_N can be refuted by CP with small coefficients in size $\mathrm{O}(N)$ and total space $\mathrm{O}(N^{2/5})$.*
2. *F_N can be refuted by CP with small coefficients in total space $\mathrm{O}\big(N^{\frac{1}{10(K+1)}}\big)$ and size $2^{\mathrm{O}\left(N^{\frac{1}{10(K+1)}}\right)}$.*
3. *Any CP refutation even with coefficients of unbounded size of F_N in formula space less than $N^{\frac{K-1}{10(K+1)}-\epsilon}$ requires length greater than $2^{\Omega\left(N^{\frac{1}{10(K+1)}}\right)}$, for any constant $\epsilon > 0$.*

Proof. The proof follows from setting $d = n^{1/K}\log n$ and $r = d/\log n$ in Theorem B.8.1 and for every N, choosing n to be a power of 2 such that N is at most a factor off from $(nd)^{10+\epsilon}$.

We note that $N^{\frac{1}{10(K+1)}} = \Theta((nd)^{\frac{10+\epsilon}{10(K+1)}})$ and $(nd)^{\frac{10+\epsilon}{10(K+1)}} > d\log N$, hence 2 holds. Moreover, $N = \mathrm{o}((nd)^{10+2\epsilon})$, thus $N^{\frac{K-1}{10(K+1)}-\epsilon} = \mathrm{o}((nd)^{(10+2\epsilon)(\frac{K-1}{10(K+1)}-\epsilon)})$ and

$$\begin{aligned}(nd)^{(10+2\epsilon)(\frac{K-1}{10(K+1)}-\epsilon)} &= (n^{(K+1)/K}\log n)^{(10+2\epsilon)(\frac{K-1}{10(K+1)}-\epsilon)}\\ &< n^{(10+2\epsilon)(\frac{K-1}{10K}-\epsilon)} \cdot n^{\epsilon} < \frac{n^{(K-1)/K}}{8}\\ &< \frac{n^{(K-1)/K}}{8}\log N = \frac{n}{8r}\log N,\end{aligned}$$

and, therefore 3 also holds. □

B.9 Exponential Separation of the Monotone AC Hierarchy

Unsurprisingly, we follow the same approach as [158]. Our function is a restriction of the GEN function, except that instead of restricting the valid instances to pyramid graphs, which are unconditionally hard, we restrict the valid instances to the graphs from Section B.6 that exhibit round-space trade-offs. We then use our round-aware simulation theorem to lift the trade-off to communication complexity and the Karchmer–Wigderson game to translate it to a trade-off for monotone circuits.

Definition B.9.1. The *Karchmer–Wigderson game* [118] is the following communication problem: given a monotone function f, Alice gets an input x such that $f(x) = 1$ and Bob gets an input y such that $f(y) = 0$. Their task is to compute a coordinate i such that $x_i = 1$ and $y_i = 0$

Theorem B.9.2 ([118]). *If there is a monotone circuit for f of size 2^c and depth r, then there is a protocol for the Karchmer–Wigderson game of communication rc and r rounds.*

Proof. The proof is a simple induction on the depth of the circuit. If the circuit has no gates, the players just return the index of the output variable. Otherwise, assume the output gate is an OR-gate, i.e., $f = \bigvee g_i$. Then $g_i(y) = 0$ for all i and there exists i such that $g_i(x) = 1$. Alice sends i with cost at most c and we apply the induction hypothesis on a circuit of one less level. If the output gate is an AND-gate, Bob acts analogously. □

Definition B.9.3. Given a graph G of indegree 2 and $\ell \in \mathbb{N}$, the G-GEN Boolean function is defined as follows. There is a variable (v, a) for every source $v \in G$ and index $a \in [\ell]$. There is a variable $(v \vee \overline{u_1} \vee \overline{u_2}, a, b, c)$ for every non-source vertex $v \in G$ with predecessors u_1 and u_2 and triple $(a, b, c) \in [\ell]^3$. There is a variable $(\overline{z}, a)$ for every index $a \in [\ell]$. A pair (v, a) is reachable if v is a source and (v, a) is 1, or there exist (b, c) such that $(v \vee \overline{u_1} \vee \overline{u_2}, a, b, c)$ is 1, (u_1, b) is reachable, and (u_2, c) is reachable. The value of G-GEN is 1 if there exists some index $a \in [\ell]$ such that (z, a) is reachable and $(\overline{z}, a)$ is 1.

Lemma B.9.4. *There is a monotone circuit that computes G-GEN in depth $2d$, fan-in ℓ^2, and size $O(m\ell^3)$, where d is the depth of G.*

Proof. The circuit computes whether each of the pairs (v, a) is reachable. For v a source we just have a variable. For a non-source v, we have, for each pair $(b, c) \in [\ell]^2$, an AND-gate of fan-in 3 and inputs the gate that computes (u_1, b), the gate that computes (u_2, c), and the variable $(v \vee \overline{u_1} \vee \overline{u_2}, a, b, c)$, and one OR-gate of fan-in ℓ^2 with inputs all of these gates. Finally, we have ℓ AND-gates with inputs the gate that computes (z, a) and the variable $(\overline{z}, a)$, and an OR-gate of fan-in ℓ. □

Lemma B.9.5. *If there is a deterministic communication protocol for the Karchmer–Wigderson game on G-GEN of communication c and r rounds, then there is a deterministic communication protocol for Lift(Search(Peb_G)) of communication c and r rounds.*

Proof. Let (x, y) be an input to assignment to *Lift*(*Search*(Peb_G)), this is a vector of indices and a vector of binary strings such that $\mathsf{select}(x_v, y_v)$ is an assignment to a variable v of Peb_G.

Alice builds a 1-input for the Karchmer–Wigderson game on G-GEN as follows. For every source $v \in G$, Alice sets the variable (v, x_v) to 1, and for every non-source vertex $v \in G$, Alice sets the variable $(v \vee \overline{u_1} \vee \overline{u_2}, x_v, x_{u_1}, x_{u_2})$ to 1. Alice sets $(\overline{z}, x_z)$ to 1. The remaining variables are 0.

Bob builds an input as follows. For every source $v \in G$, Bob sets the variable (v, a) to $(y_v)_a$, and for every non-source vertex $v \in G$, Bob sets the variable $(v \vee \overline{u_1} \vee \overline{u_2}, a, b, c)$ to 0 if $(y_v)_a = 0$, $(y_{u_1})_b = 1$, and $(y_{u_2})_c = 1$, and to 1 otherwise. Bob sets $(\overline{z}, a)$ to $1 - (y_z)_a$. Observe that, in Bob's input, if a pair (v, a) is reachable, then $(y_v)_a = 1$. For the sink, this means that if (z, a) is reachable then $(\overline{z}, a) = 1 - (y_z)_a = 0$, so the input evaluates to 0.

Both players then simulate the protocol for the Karchmer–Wigderson game and they get a variable that Alice set to 1 and Bob set to 0. If it is (v, x_v), then $(y_v)_{x_v} = 0$, so axiom v is falsified. If it is $(v \vee \overline{u_1} \vee \overline{u_2}, x_v, x_{u_1}, x_{u_2})$, then $(y_v)_{x_v} = 0$, $(y_{u_1})_{x_{u_1}} = 1$, and

$(y_{u_1})_{x_{u_1}} = 1$, so axiom $v \vee \overline{u_1} \vee \overline{u_2}$ is falsified. If it is (t, x_v), then $(y_z)_{x_v} = 1$, so axiom $\overline{z}$ is falsified. □

Theorem B.1.3 (Restated). *For every $i \in \mathbb{N}$ there is a Boolean function over n variables that can be computed by a monotone circuit of depth $\log^i n$, fan-in $n^{4/5}$, and size $O(n)$, but for which every monotone circuit of depth $q \log^{i-1} n$ requires size $2^{\Omega(n^{1/(10+4\epsilon)q})}$.*

Proof. Let G be a stack of $\log^{i-1} n$ butterflies with $w = (n^{1/(10+3\epsilon)})/(\log^{i-1} n)$ sources and sinks. This is a graph of depth $d \leq \log^i n/(10+3\epsilon)$ and size $m \leq n^{1/(10+3\epsilon)}$, so we can set $\ell = m^{3+\epsilon} \leq n^{(3+\epsilon)/(10+3\epsilon)}$, and the number of variables of G-GEN is indeed at most $m\ell^3 \leq n$. By Lemma B.9.4, there is a monotone circuit of depth $2d \leq \log^i n$, fan-in $\ell^2 \leq n^{4/5}$, and size $O(n)$ that computes G-GEN. By Theorem B.9.2, Lemma B.9.5, Theorem B.4.1, Lemma B.2.6, and Lemma B.6.1, any circuit of depth at most $q \log^{i-1} n$ that computes G-GEN requires size $2^{(w^{1/q} \log \ell)/(4r)} = 2^{\Omega(n^{1/(10+4\epsilon)q})}$. □

B.10 Concluding Remarks

In this paper we report the first true size-space trade-offs for cutting planes, exhibiting CNF formulas which have small-size and small-space proofs with constant-size coefficients but for which any short proofs must use a lot of memory, even when using exponentially large coefficients and even when we measure just the number of lines (i.e., inequalities) rather than total size. Furthermore, these results also hold for resolution and polynomial calculus, and are thus the first trade-offs to uniformly capture the proof systems underlying the currently best SAT solvers.

The main technical component in our proof is a reduction to communication complexity as in [106, 95], but with the crucial difference that we reduce to round-efficient protocols in the real communication model of [125]. Extending the techniques in [158, 96, 43] to this more general setting, and combining them with new trade-off results for Dymond–Tompa pebbling [74], yields our results. Using the same approach we are also able to obtain an exponential separation between monotone-AC^{i-1} and monotone-AC^{i}, improving on the superpolynomial separation in [158].

An interesting challenge would be to extend our reduction to stronger communication models such as two-party randomized or multi-party real communication, which would yield trade-offs for stronger proof systems. A recent result in this direction is [92], but unfortunately it seems hard to incorporate round-efficiency in this framework.

Another question concerns the size of the lifting gadgets we need to construct formulas exhibiting trade-offs. Our gadgets have large polynomial size, which incurs a substantial loss in the results. It would be nice to construct constant-size gadgets, which could lead to tighter trade-off results.

Many proof complexity trade-offs have been obtained by reducing to the *black-white pebble game* [67], but in this paper we use the Dymond–Tompa game. It would be desirable to obtain a better understanding of the role of these games and what kind of trade-offs can be obtained from them.

Finally, from a proof complexity perspective we have very few examples of formula families that exhibit size-space trade-offs. Apart from the pebbling formulas studied in this work, the only natural examples[4] are the Tseitin contradictions over long, narrow grids in [19, 23]. It would be interesting to prove size-space trade-offs for the latter formulas also in cutting planes, or to find other formulas with size-space trade-offs for this or other proof systems.

Acknowledgements

The authors wish to acknowledge Mladen Mikša, who participated in the initial stages of this work and has kept contributing helpful remarks throughout the project, and Massimo Lauria, with whom we have had many fruitful discussions on time-space trade-offs and other topics in proof complexity. We want to thank Siu Man Chan for introducing us to the wonderful world of Dymond–Tompa pebbling. Different subsets of the authors are grateful for detailed and very helpful discussions on communication complexity with Joshua Brody, Arkadev Chattopadhyay, Prahladh Harsha, Johan Håstad, Troy Lee, Jaikumar Radakrishnan, and Anup Rao. Finally, we are thankful to Dieter van Melkebeek and Ryan Williams for help with references for general SAT time-space trade-offs.

The authors were funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 279611 as well as by the Swedish Research Council grant 621-2012-5645.

[4]Ignoring trade-offs obtained in [143] by gluing together disjoint copies of unrelated formulas.

Paper C

Cumulative Space in Black-White Pebbling and Resolution

Joël Alwen, Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals

Full length version of the article published in Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS '17), January 2017

Abstract

We study space complexity and time-space trade-offs with a focus not on peak memory usage but on overall memory consumption throughout the computation. Such a cumulative space measure was introduced for the computational model of parallel black pebbling by [Alwen and Serbinenko '15] as a tool for obtaining results in cryptography. We consider instead the nondeterministic black-white pebble game and prove optimal cumulative space lower bounds and trade-offs, where in order to minimize pebbling time the space has to remain large during a significant fraction of the pebbling.

We also initiate the study of cumulative space in proof complexity, an area where other space complexity measures have been extensively studied during the last 10–15 years. Using and extending the connection between proof complexity and pebble games in [Ben-Sasson and Nordström '08, '11], we obtain several strong cumulative space results for (even parallel versions of) the resolution proof system, and outline some possible future directions of study of this, in our opinion, natural and interesting space measure.

C.1 Introduction

The time and space complexity measures are at the heart of understanding computation. Unfortunately, there is little we can say about general computation models such as

Boolean circuits, let alone Turing machines. But if we allow ourselves to work with simpler models of computation, then we have a better chance at understanding these resources, and in fact there has been impressive progress in restricted models like bounded-depth circuits.

One of the first success stories in this direction are pebble games. The original *(black) pebble game* is played by a single player on a directed acyclic graph (DAG) with a single sink and all vertices having bounded indegree and consists of two simple rules:

1. we can add a pebble to a vertex if all its direct predecessors have pebbles, and

2. we can remove a pebble from a vertex at any time.

The goal of the game is to place a pebble on the sink of the graph. Time is measured as the number of moves to reach this goal, and the space is the maximum number of pebbles needed simultaneously at any point during the pebbling.

Quite surprisingly, this seemingly simple and innocent game can be used to obtain strong results even for general computation models, as it is at the core of the $\mathsf{DTIME}(t) \subseteq \mathsf{SPACE}(t/\log t)$ space upper bound for Turing machines in [104]. Pebbling was first used in [149] to study flowcharts and recursive schemata, and different variants of the game have later been applied to a rich selection of problems in computer science, including register allocation [171], algorithmic time and space trade-offs [58], parallel time [74], communication complexity [158], monotone space complexity [57, 81], cryptography [8, 73], and proof complexity [27, 29, 43] (where it should be emphasized that the above list of references is far from exhaustive). An excellent overview of pebbling up to ca 1980 is given in [153] and another in-depth treatment of some pebbling-related questions can be found in chapter 10 of [167]. Some more recent developments are discussed in the upcoming survey [146].

Let us briefly discuss what is known about space in proof complexity, since this is one of the two topics we are focusing on in this paper. The study of space in proof complexity was initiated in [77], which introduced the *clause space* measure for the well-known resolution proof system, a measure that has subsequently been thoroughly investigated. Informally, the clause space of a resolution proof can be defined as the maximal number of additional clauses—on top of the clauses in the original CNF formula—that a verifier needs to keep in memory at any time while checking the correctness of the proof.[1] While some formulas have proofs requiring only a small, sometimes even just constant, space overhead during verification, other formulas require a linear amount of extra space [2, 25, 77], and as shown in [77] no formulas require more than linear clause space in resolution.

Other papers have studied how space relates to other proof complexity measures. With respect to proof length, which can be viewed as a measure of (nondeterministic) running time, there is a wide range of trade-off results. It has been shown that there are

[1]Though slightly different from the definition in [77], this is equivalent up to a small additive constant.

formulas which have both short and space-efficient proofs, but as one of these measures is optimized the other one can blow up to almost worst-case behaviour [28]. Not only this, but there are even formulas where short proofs require more than the worst-case linear space [19, 23]. Yet other papers have studied other space measures such as *total space* [2, 38, 40], measuring the total number of symbols in a proof, and space complexity has also been considered for other proof systems than resolution. We refer the reader to the survey [145] for more details (although, for obvious reasons, it fails to cover the very latest results on total space).

All the space measures discussed above have in common the fact that they refer to the maximum space used at some point in the proof, but they are far from providing a complete picture of space usage during the whole proof. If we only know that a formula has high space complexity, it is not possible to distinguish between a formula that requires large space only at the beginning of the proof, say, and another that requires large space throughout the whole proof. This distinction might not be so important if we are considering the memory requirements of a verifier, since in this case we are chiefly interested in the maximum. However, it could be relevant for proof search: an algorithm that searches for a proof by producing clauses needs to discard many of them or risk exhausting its available memory. In this case, the difference between needing large space once versus at all times is the difference between making one lucky choice of which clauses to keep in memory versus being lucky all the time.

A similar issue occurs with so-called *memory-hard functions* in the context of cryptography. The idea behind memory-hard functions is that they should require a large amount of memory to evaluate, so that in order to compute such a function for many inputs as a part of a brute-force attack either an infeasible amount of memory is needed or the attack needs to be carried out sequentially. Yet, if the function only requires a large amount of memory during a limited time of the computation, then it is possible to reuse memory for different computations overlapping suitably in time as observed in [7]. Therefore, a more appropriate measure to analyse memory-hard functions is *cumulative space complexity* as introduced in [8], where one measures not the maximum memory consumption but the total memory usage aggregated over the time of the computation.

Although with hindsight this cumulative space complexity measure appears to be a very natural way of quantifying memory usage, it does not seem to have received too much attention in computational complexity theory, and to the best of our knowledge it has not been considered at all in the context of proof complexity. One of the main contributions of this paper is to transfer the concept of cumulative space to proof complexity and to initiate a study of this complexity measure for the resolution proof system.

Pebble games turn out to be a useful tool also for analysing cumulative space. For pebbling strategies cumulative space is straightforwardly defined as the sum over all steps of the pebbling of the number of pebbles on the DAG at each point in time. Thus, in the standard pebble game discussed above any DAG with n vertices can trivially be pebbled in time n and cumulative space $O(n^2)$ by placing pebbles on all vertices in topological order. Since every vertex needs to be pebbled at some point, a trivial lower bound for the cumulative space is n. However, depending on the intended application one needs to consider other variations of this pebble game as discussed next.

In a proof complexity setting we need to study the *black-white pebble game*, which was introduced in [67] with the objective of modelling nondeterministic computations. Here white pebbles, corresponding to nondeterministic guesses, can be placed at any vertex at any time, but such a white pebble can only be removed from a vertex when all direct predecessors have (black or white) pebbles, corresponding to that the correctness of the nondeterministic guess can be verified.

To model parallel computation in a cryptographic setting, [8] introduced yet another pebble game, namely the *parallel (black) pebble game*. In this game, all the pebbling moves that are legal at some point in time can be performed simultaneously in one single step. This change of rules does not affect the maximal space required to pebble a DAG, but typically changes the pebbling time. Any connected DAG with a single sink requires linear time to pebble sequentially, but for a parallel pebbling it is easy to see that the time required is upper-bounded by the depth of the graph (i.e., the length of a longest path). We remark that an attractive feature of parallel pebbling is that it better captures the difference between maximal and cumulative space. Note that in any sequential pebbling game placing s pebbles requires s time steps, and during the last $s/2$ steps there will be at least $s/2$ pebbles on the DAG. Thus, any pebbling in maximal space s requires cumulative space $\Omega\big(s^2\big)$. In contrast, in a parallel pebbling the cumulative space can be small even when the maximal space is large.

C.1.1 Our Pebbling Contributions

In this paper, we study the cumulative space measure in the context of black-white pebbling. In order to do so, we also extend black-white pebbling to a parallel version. As pebble games go, this is a very powerful model, since it turns out that any DAG can be pebbled with a parallel black-white pebbling in constant time and linear cumulative space. Perhaps somewhat surprisingly, however, it is still possible to prove nontrivial time-space trade-offs. It can be shown that the parallel and sequential versions of black-white pebbling are closely connected (as discussed in more detail later in the paper), and therefore in this overview the exposition is focused on sequential black-white pebbling.

The first question we address is how the large cumulative space can be in the worst case for sequential black-white pebbling. As noted above, a trivial (black-only) pebbling in linear time and space has cumulative space $O\big(n^2\big)$ for any graph over n vertices. In the other direction, the $\Omega(n/\log n)$ space lower bound in [89] already gives a $\Omega(n^2/\log^2 n)$ cumulative space lower bound for sequential black-white pebbling, as explained above. One cannot get a better cumulative space lower bound by this simple argument from maximal space lower bounds, however, since any DAG of constant indegree can be pebbled in maximal space $O(n/\log n)$ [104].

We prove that the family of *grate graphs* in [169] require $\Omega(n^2)$ cumulative space for sequential black-white pebbling. This shows that for cumulative space it is not possible to improve on the trivial quadratic upper bound, in contrast to the maximal space measure where it is always possible to save a logarithmic factor from the trivial linear upper bound. This is also different from the parallel black pebble game, where there is a $o(n^2)$ worst-case upper bound for cumulative space [7] and the best known cumulative space lower bound is $\Omega\bigl(n^2/\log n\bigr)$ [9]. In fact, it turns out that the difference between the sequential black-white and parallel black pebble games can be very large. We also prove that (a modified version of) the *butterfly graphs* in [173] require cumulative space $\Omega(n^2/\log n)$ in the sequential black-white pebble game but can be pebbled in linear cumulative space in the parallel black pebble game. Butterfly graphs also show that graphs that require large cumulative space do not necessarily require large maximal space, as they have logarithmic depth and thus can be pebbled in logarithmic space as observed in [104]. We obtain these results by studying the lower bounds on cumulative space in parallel black pebbling in [9] in terms of *depth-robustness* of graphs, and extending these lower bounds to other pebble games and other families of graphs.

Our next set of results concern trade-offs between time and space. Here our starting point is the family of *bit-reversal permutation graphs* studied in [130] which can be pebbled either with 3 pebbles or (as any graph) in linear time, but for which any pebbling in time t and space s must satisfy $t = \Omega(n^2/s^2)$, where as before n is the number of vertices in the graph.

We strengthen this trade-off to cumulative space, proving that pebblings of these graphs in space s require cumulative space $\Omega\bigl(n^2/s\bigr)$, which in particular implies that a pebbling in time $O\bigl(n^2/s^2\bigr)$ must use space $\Omega(s)$ not only at some point but most of the time.[2] Furthermore, we establish an unconditional $\Omega\bigl(n^{3/2}\bigr)$ cumulative space lower bound, which provides another example of graphs that require (at least somewhat) large cumulative space but can be pebbled in very small (even constant) maximal space. Our proofs of these results work by adapting the *dispersion* technique from [9]. This technique has the advantage that it isolates an abstract combinatorial property of the graph that makes the lower bound argument go through, and this cleaner approach enables us to prove these results not only for bit-reversal graphs but also for *random permutation graphs* (by showing that these graphs possess the required combinatorial property with high probability). To the best of our knowledge no trade-offs (even non-cumulative ones) were known for such graphs before for any flavour of the pebble game.

Finally, we consider a very concrete, extremal question regarding pebbling time-space trade-offs. It is an easy observation that any sequential black-white pebbling in constant space s can be carried out in time $O\bigl(n^s\bigr)$, since there are only $\sum_{k=0}^{s} 2^k \binom{n}{k}$ possible different configurations of s pebbles in the graph, and no configuration repeats in a pebbling (or else the intermediate moves can be removed). In fact, a bit more thought reveals that this time bound can be sharpened to $O\bigl(n^{s-1}\bigr)$, since every configuration in space s is immediately followed by a pebble removal, and so we only need to consider distinct

[2]Note, importantly, that such a space lower bound is *not* implied by the simple "space s implies cumulative space $\Omega\bigl(s^2\bigr)$" argument discussed previously.

configurations of $s-1$ pebbles. It is a natural question whether this simple counting argument is in fact tight, so that there are graphs that can be pebbled in space s but where any such pebbling requires time $\Omega\bigl(n^{s-1}\bigr)$.

For pebbling space $s = 3$, the minimum space in which any nontrivial pebbling strategy is possible, the bit-reversal graphs in [130] discussed above show that the answer to this question is affirmative. It is not hard to see that by stacking $s-2$ bit-reversal DAGs on top of one another, identifying the top layer in one graph with the bottom layer in the graph above, one obtains graphs that are pebblable in space s but where the obvious pebbling strategy achieving this bound requires time $\mathrm{O}\bigl(n^{s-1}\bigr)$. We prove that this trivial upper bound is indeed asymptotically tight for any constant s.

C.1.2 Our Proof Complexity Contributions

Turning now to proof complexity, we consider the main contribution of our paper to be that we initiate the study of the cumulative space measure. While the concept of cumulative space seems to be as natural as maximal space, we are not aware of it having been studied in the context of proof complexity before. As was the case for the first papers on (maximal) space complexity in resolution [77], in this first paper on cumulative space in proof complexity we focus on the resolution proof system.

An immediate observation is that proof length is always a lower bound on cumulative space, and so exponential lower bounds on proof length—as shown for resolution in [61, 98, 177] and many later papers—trivially imply exponential lower bounds on cumulative space. Therefore, it seems that the cumulative space measure will be of independent interest mostly for formulas which have reasonably short proofs. An obvious candidate family to study are pebbling formulas [29], which have proofs in linear length, but which exhibit a rich variety of properties with respect to space complexity depending on the underlying graphs in terms of which they are defined.

However, we also need to decide on an appropriate model of the resolution proof system in which to study cumulative space. In the context of pebbling we concluded that cumulative space makes most sense for parallel versions of the pebble games, and so it is natural to ask whether one should consider a parallel version of resolution when studying cumulative clause space. It is not hard to argue that such a parallel model of resolution could be interesting in its own right, since it might be useful as a tool to analyse attempts to parallelize state-of-the-art SAT solvers using so-called *conflict-driven clause learning (CDCL)* [18, 136].

We define and study several different versions of the resolution proof systems with varying degrees of parallelity. The running time of parallel CDCL solvers has previously been analysed using resolution depth and the related *conflict resolution depth* and *schedule makespan* measures introduced in [120], and our models of parallel resolution allow us to reason about space in addition to time.

Similarly to what is the case for pebble games, our most general model of parallel resolution, where clauses can be inferred not just by syntactic application of the resolution rule but by semantic inference, is extremely powerful, so much so that it can deal with any formula in a constant number of steps and linear space. Since we can establish a tight relation between space and parallel speedup also for resolution, however, we can still obtain lower bounds when the maximal space is limited.

Studying pebbling formulas in these different models of resolution, and revisiting the reductions between resolution and pebble games in [27, 28], we can translate the pebbling results in Section C.1.1 to results for the resolution proof system. Summarizing very briefly, we exhibit different formulas that have

- proofs in linear length but require quadratic cumulative space,
- proofs in logarithmic space but require $\Omega(n^2/\log n)$ cumulative space, and
- trade-offs between proof length and cumulative space.

C.1.3 Paper Outline

The rest of this paper is organized as follows. In Section C.2 we present a more detailed overview of our pebbling results, introducing formal definitions of the pebble games and measures discussed above, and we give an analogous overview for resolution in Section C.3. Section C.4 contains detailed proofs of our pebbling theorems, and Section C.5 explains how the reduction from pebbling to resolution in [27, 28] can be used to derive cumulative space results for resolution even in a parallel setting. We conclude in Section C.6 with a discussion of possible directions for future research.

C.2 Pebbling Results Overview

Let us start our pebbling overview by giving formal definitions of the basic concepts.

C.2.1 Definition of Pebble Games and Basic Properties

We say that a directed acyclic graph (DAG) $G = (V, E)$ with $|V| = n$ has *size* n. A vertex $v \in V$ has *indegree* δ if it has δ incoming edges $\{(u_1, v), \ldots, (u_\delta, v)\} \subseteq E$, $u_i \neq u_j$ for $i \neq j$, and we say that G has indegree δ if the maximum indegree of any vertex of G is δ. A vertex with no incoming edges is called a *source* and a vertex with no outgoing edges is called a *sink*. We say that a vertex u is a *predecessor* of a vertex v if there exists a directed path from u to v; moreover, if this path consists of only one edge then u is a *direct predecessor* of v. We denote by $\mathsf{parents}(v)$ the set of all direct predecessors of v. For technical reasons, it will sometimes be convenient to allow paths of length 0 in the definition above, so that a vertex can be a predecessor of itself. We will sometimes consider graphs obtained from other graphs by removing subsets of vertices, and for $U \subseteq V$ we write $G - U = \bigl(V \setminus U, E \setminus ((U \times V) \cup (V \times U))\bigr)$ to denote the DAG obtained from G by removing the vertices in U and all edges incident to U.

To get a unified description of all flavours of the pebble game discussed in Section C.1, it is convenient to define pebbling as follows.

Definition C.2.1 (Pebble games). Let $G = (V, E)$ be a DAG with a unique sink vertex z. The black-white pebble game on G is the following one-player game. At any time i, we have a *black-white pebbling configuration* $\mathbb{P}_i = (B_i, W_i)$ of black pebbles B_i and white pebbles W_i on the vertices of G, at most one pebble per vertex. The rules of how a pebble configuration $\mathbb{P}_{i-1} = (B_{i-1}, W_{i-1})$ can be changed to $\mathbb{P}_i = (B_i, W_i)$ are as follows:

1. A black pebble may be placed on a vertex v only if all immediate predecessors of v are covered by pebbles in both $\mathbb{P}_{i-1}$ and $\mathbb{P}_i$, i.e.,
$$v \in (B_i \setminus B_{i-1}) \Rightarrow \mathsf{parents}(v) \subseteq \mathbb{P}_{i-1} \cap \mathbb{P}_i \ .$$
Note that, in particular, a black pebble can always be placed on a source vertex.

2. A black pebble on any vertex v in $\mathbb{P}_{i-1}$ can be removed in $\mathbb{P}_i$.

3. A white pebble can be placed on any vertex v in $\mathbb{P}_i$.

4. A white pebble on a vertex v in $\mathbb{P}_{i-1}$ may be removed in $\mathbb{P}_i$ only if all immediate predecessors of v are covered by pebbles in both $\mathbb{P}_{i-1}$ and $\mathbb{P}_i$, i.e.,
$$v \in (W_{i-1} \setminus W_i) \Rightarrow \mathsf{parents}(v) \subseteq \mathbb{P}_{i-1} \cap \mathbb{P}_i \ .$$
In particular, a white pebble can always be removed from a source vertex.

A *legal pebbling* $\mathcal{P}$ of G is a sequence $\mathcal{P} = (\mathbb{P}_0, \ldots, \mathbb{P}_\tau)$ where every configuration $\mathbb{P}_i$ can be obtained from $\mathbb{P}_{i-1}$ using the rules 1–4. A *complete pebbling* $\mathcal{P} = (\mathbb{P}_0, \ldots, \mathbb{P}_\tau)$ is a legal pebbling where $\mathbb{P}_0 = \mathbb{P}_\tau = (\emptyset, \emptyset)$ and $z \in \bigcup_{i=0}^{\tau}(B_i \cup W_i)$ (i.e., the sink is pebbled at some point).

A *black pebbling* is a pebbling where $W_i = \emptyset$ for all $i \in [\tau]$. A pebbling is *sequential* if only a single application of a single rule 1–4 is used to get from from $\mathbb{P}_{i-1}$ to $\mathbb{P}_i$ for all $i \in [\tau]$. In a *(fully) parallel* pebbling an arbitrary number of applications of the rules 1–4 can be made to $\mathbb{P}_{i-1}$ to obtain $\mathbb{P}_i$ (but note that all pebble placements and removals have to be legal with respect to $\mathbb{P}_{i-1}$, and cannot make use of any pebble placements or removals made in parallel). Finally, we will also consider *parallel-black sequential-white* pebblings, which allows parallel applications of black pebble rules 1–2 to $\mathbb{P}_{i-1}$ to obtain $\mathbb{P}_i$, but only a single application of the white pebble rules 3–4. Note that, in the parallel setting, a simultaneous application of rules 1 and 4 on a same vertex replaces a white pebble by a black one.

The *time* of a pebbling $\mathcal{P} = (\mathbb{P}_0, \ldots, \mathbb{P}_\tau)$ is $t(\mathcal{P}) = t$; the (maximal) *space* is $s(\mathcal{P}) = s = \max_{i \in [t]} |B_i| + |W_i|$; and the *cumulative space* is $c(\mathcal{P}) = c = \sum_{i \in [t]} |B_i| + |W_i|$ (where we observe that $c \leq st$).

Parallel black pebbling was introduced in [8], where it was pointed out that for certain graphs parallel pebblings can be much more efficient than sequential, while for others they cannot do any better. For example, if we are considering time-space tradeoffs, any sequential black pebbling in space s and time t of the bit-reversal graph must satisfy $st = \Omega\big(n^2\big)$ [130], while in the parallel black game one can pebble such graphs in linear time and space $\mathrm{O}(\sqrt{n})$ [8]. In contrast, it was shown in [9] that there are graphs that can be pebbled sequentially in space s and time t satisfying $st = \mathrm{O}\big(n^2/\log n\big)$, but where these graphs even in the parallel model require not only $st = \Omega\big(n^2/\log n\big)$ but also cumulative space $\Omega\big(n^2/\log n\big)$.

Unlike the case of the black pebble game, we show that time and space in the black-white sequential and parallel games are closely related. Up to constant factors, it holds that if a parallel black-white pebbling $\mathcal{P}$ has maximal space s, then it is possible to save a factor s, but not more than a factor s, in time compared to a sequential black-white pebbling in the same space s.

Observation C.2.2. *Let $\mathcal{P}$ be a parallel black-white pebbling of a DAG G in time t, space s, and cumulative space c. Then there is a sequential black-white pebbling of G in time $2ts$, space $2s$, and cumulative space cs.*

Proof. Each parallel move places at most s pebbles and removes at most s pebbles, therefore we can simulate it by $2s$ sequential moves (making the pebble placements first, to make sure that these moves remain legal). □

Lemma C.2.3. *Let $\mathcal{P}$ be a sequential black-white pebbling of G in time t, space s, and cumulative space c, and let k be a positive integer. Then there is a parallel black-white pebbling of G in time $3\lceil t/k\rceil$, space $s + \lceil k/2\rceil$, and cumulative space $3\lceil c/k\rceil + t$.*

Proof. We divide $\mathcal{P}$ into $\lceil t/k\rceil$ intervals of (at most) k moves. We reorder the pebbling moves within each of these intervals so that we do all placements first and removals afterwards. This is still essentially a valid pebbling, because each configuration is a superset of the corresponding configuration in $\mathcal{P}$, except that we can possibly have vertices temporarily covered by several pebbles. The space usage in any intermediate configuration increases to at most $s + \lceil k/2\rceil$. We then collapse each subsequence into one parallel placement of white pebbles, one step replacing white pebbles with black pebbles as needed, and one parallel removal of black pebbles (this allows us to make all black pebble placements in parallel even though later black pebbles might be dependent on earlier pebble placements in the sequential pebbling). This decreases the time to $3\lceil t/k\rceil$.

To bound the cumulative space, note that if there is a configuration $\mathbb{P}_i$ in a sequential interval that has space s_i, then the corresponding three parallel configurations have aggregate space at most $3s_i + 2x_j$, where x_j is the number of placements in that interval. Now consider a partition of the sequential pebbling into k subsequences $\mathbb{P}_i, \mathbb{P}_{i+k}, \ldots, \mathbb{P}_{i+k(\lceil t/k\rceil-1)}$ of t/k configurations, evenly spaced, starting at $i \in [1,k]$. By an averaging argument, at least one of these k subsequences has a cumulative space of at most $\lfloor c/k \rfloor$. Hence the total cumulative space is at most $\sum_{j\in[t/k-1]}(3s_{i+j} + 2x_{i+j}) = 3\sum_{j\in[t/k-1]} s_{i+j} + 2\sum_{j\in[t/k-1]} x_j \geq 3c/k + 2t/2$. $\square$

Observe that when $k = \Theta(s)$ the cumulative space in Lemma C.2.3 is dominated by the term t, so we only save a factor s in cumulative space when the sequential pebbling has cumulative space $c = \Theta(st)$. Since the graphs we will discuss in what follows have cumulative space lower bounds of this form, studying the sequential game already gives us all the information we want about the parallel game.

Corollary C.2.4. *Let $\mathcal{P}$ be a black-white pebbling of G in time t and space s. Then there is a parallel black-white pebbling of G in time $\lceil t/2s \rceil$, space $4s$, and cumulative space $2t$.*

C.2.2 Robustness and High Cumulative Space Complexity

We proceed to define the concept of *depth-robustness* of graphs, which is inspired by [76, 150] and which will be central to our work.

Definition C.2.5 ($\mathcal{G}$ -robustness). Let $\mathcal{G}$ be a family of DAGs and let $e, d \in \mathbb{N}^+$ be positive integers. We say that a DAG $G = (V, E)$ is *(e, d)-$\mathcal{G}$-robust* if for every subset of vertices $U \subseteq V$ of size at most e it holds that $G - U$ contains a subgraph $H \in \mathcal{G}$ of size at least d.

When $\mathcal{G}$ is the class of directed paths, then we say that G is *depth-robust*, and when $\mathcal{G}$ is the class of DAGs with one sink the DAG G is said to be *predecessor-robust*.[3]

For our pebbling lower bounds we are interested in graphs with very high robustness, i.e., for as large values of e and d as possible. Depth-robustness was first studied by Erdős, Graham and Szemerédi [76] who showed how to construct DAGs with indegree $\Theta(\log(n))$ possessing $(\Omega(n), \Omega(n))$-depth-robustness. However, in our applications it is important that the graphs have *constant* indegree. Valiant [179] showed that for constant indegree and linear depth the best we can hope for is $(\mathrm{O}(n/\log n), \mathrm{O}(n))$-depth-robustness. Fortunately for us, it was shown in [9, 150] that such extremal $(\Theta(n/\log n), \Theta(n))$-depth-robust graphs do exist. Conversely, if we want constant indegree with the parameter e linear in the graph size, then $(\epsilon n, n^{1-\epsilon})$-depth-robustness is the best we can hope for [179]. In [169] a family of constant-indegree $(\Theta(n), \Theta(n^{1-\epsilon}))$-depth-robust graphs were presented.

The connection between depth-robustness and cumulative space was made in [9], where it was shown that an (e, d)-depth-robustness graph requires parallel black cumulative space at least ed. In this work, we give a more general theorem of this form for the case of $\mathcal{G}$-robustness. We then use this theorem to obtain the following lower bounds for depth-robust and predecessor-robust graphs.

[3]This choice of terminology is inspired by [150], which discusses the dual notions of "depth-separators" and "predecessor-separators."

Lemma C.2.6. *If G is an (e, d)-depth-robust DAG, then G requires sequential black-white cumulative space at least ed, and parallel-black sequential-white cumulative space at least $e\sqrt{d}$.*

Lemma C.2.7. *If G is an (e, d)-predecessor-robust DAG, then G requires black-white cumulative space at least ed.*

Focusing on the range of parameters discussed above, we can see that, it follows from Lemmas C.2.6 and C.2.7 that a $(\Theta(n/\log n), \Theta(n))$-depth-robust graph has sequential black-white cumulative space complexity $\Omega\big(n^2/\log n\big)$ and parallel-black sequential-white pebbling space complexity $\Omega\big(n^{3/2}/\log n\big)$.

A class of DAGs that are predecessor-robust are *grates*—graphs with n' sources and n' sinks such that after the removal of an arbitrary set of kn' vertices (for some constant k) there are still a linear number of sources and sinks that are all pairwise connected.[4] *Butterfly graphs* [173] are grates with $n = n' \log n'$ vertices that are $(\Theta(n/\log n), \Theta(n/\log n))$-predecessor-robust. Moreover, it is not hard to show that if we append n' single-sink DAGs of size $\log n'$, one to each source of the butterfly graph, the resulting graph is $(\Theta(n/\log n), \Theta(n))$-predecessor-robust. This implies that these graphs require cumulative space $\Omega(n^2/\log n)$. Note that butterfly graphs (also in the modified version just described) can be pebbled with $O(\log n)$ pebbles (since the graphs have depth $O(\log n)$), and thus it is not the case that high cumulative space implies large maximal space.

It has been established that extremal depth-robustness is both a necessary [7] and sufficient [9] condition to have high cumulative space in the parallel black game. In particular, using the fact that no graph of size n with constant indegree is $(\omega(n/\log n), \Theta(n))$-depth-robust, it was shown in [7] that in the parallel black pebble game, for any constant $\epsilon > 0$, any such graph has cumulative space complexity $o\big(n^2/\log^{1-\epsilon} n\big)$. A natural question is if this also holds for black-white pebbling. We show that this is not the case: there are graphs that have maximum cumulative space complexity $\Omega(n^2)$ in the black-white pebble game. This follows from Lemma C.2.7 and the existence of grates of size linear in the number of sources and sinks [169].

C.2.3 Dispersion and Cumulative Space Trade-Offs

Another property of graphs that is important in the current paper is *dispersion*. This notion was used in [9] to obtain another condition ensuring high parallel black cumulative space complexity. We define two similar concepts and then use them to obtain cumulative space trade-offs. The results we get are for two classes of *permutation graphs*—graphs that consist two ordered paths of vertices $1, 2, \ldots, n$, where in addition an edge is added

[4]Strictly speaking, these graphs have multiple sinks and so do not conform to the DAG requirements in Definition C.2.1. However, it is easy to turn any multi-sink DAG of interest into a single-sink DAG with essentially the same properties—we refer to Section C.4 for the details—and so we ignore this technicality here.

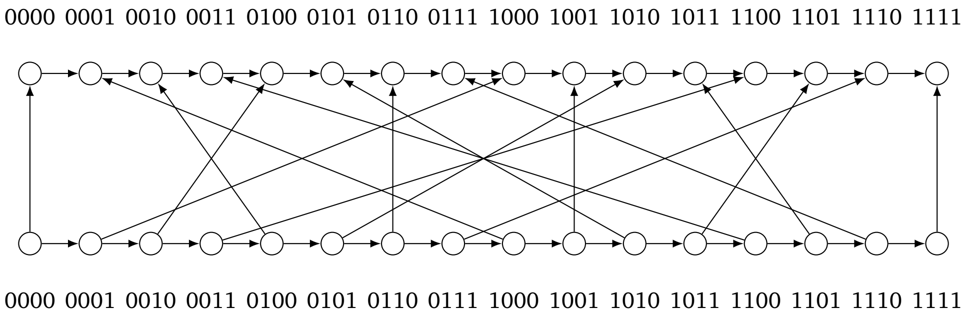


Figure C.1: A bit reversal permutation graph

from each vertex i in the first path to its image under some specified permutation σ in the second path.

A family of permutations that will be of particular interest to us are the so-called *bit-reversal permutations*, which are defined for $n = 2^m$ and which simply reverse the binary representations of numbers. That is, if $j = (b_1 \cdots b_m)_{(2)}$, then the bit-reversal permutation σ sends j to $\sigma(j) = (b_m \cdots b_1)_{(2)}$ (see Figure C.1). It was previously known [130] that any sequential black-white pebbling of a bit-reversal permutation graph on $2n$ vertices in time t and space s satisfies $st = \Omega\bigl(n^2/s\bigr)$. Moreover, it was shown in [130] that this is tight up to constant factors and that there is a black-white pebbling in time t and space s such that $st = \mathrm{O}\bigl(n^{3/2}\bigr)$.

We observe that while bit-reversal graphs are *not* $(2\sqrt{n}, 2\sqrt{n})$-depth-robust, they can be shown to be $(\sqrt{n}, n)$-predecessor-robust. Therefore, in constrast to [9], where it was not possible to establish a parallel black cumulative space lower bound of $n^{3/2}$ using depth-robustness, we are able to obtain a black-white cumulative space lower bound of $n^{3/2}$ using predecessor-robustness.

Our reason for studying dispersion properties of bit-reversal graphs is to characterize how cumulative space increases when space decreases. As a corollary of Lemma C.4.10 we can show that the time-space trade-off in [130] can be strengthened to a cumulative space trade-off. Our result implies that if $\mathcal{P}$ is a sequential black-white pebbling of a bit-reversal graph in space s and time n^2/s^2, then it needs to use space s not only at some point of the pebbling, but during a large part of the time.

An advantage of our approach is that we identify a general property of graphs that imply cumulative space trade-offs, so that the task of establishing a trade-off reduces to proving that the graph has this desired property. As a consequence of this simplification, we are able to prove the same kind of trade-off results not only for bit-reversal graphs but also for random permutation graphs, a class of graphs for which it seems nothing was known before.

Theorem C.2.8. *If G is a random permutation graph, then it holds asymptotically almost* *surely that in the sequential black-white pebble game G requires cumulative space $\Omega(n^{3/2})$ and any pebbling $\mathcal{P}$ of G in maximal space s has cumulative space $\Omega(n^2/s)$.*

C.2.4 Pebblings in Small Space Can Require Maximum Length

Let us finally consider the question of how long a shortest sequential pebbling of a graph can be given constraints on the maximal pebbling space. Without loss of generality, a black pebbling in space s takes time at most $\binom{n}{\leq s} \leq n^s$, simply because there is no need to repeat any pebble configuration. A moment of thought reveals that in fact we get the upper bound $\binom{n}{s-1} + \binom{n}{\leq s-1} \leq n^{s-1}$, since every configuration in maximal space s is followed by an erasure yielding a space-$(s-1)$ configuration, and these configurations also do not repeat. For black-white pebbling the upper bound becomes $2^{s-1}\left(\binom{n}{s-1} + \binom{n}{\leq s-1}\right) \leq 2^{s-1}n^{s-1}$.

As discussed in the introduction, it can be read off from [130] that for space-3 pebblings the $\mathrm{O}\big(n^2\big)$ upper bound is tight up to constant factors—bit-reversal DAGs are examples of graphs for which pebblings in optimal space 3, or indeed any constant space, require quadratic time. We extend this result to any $s = \mathrm{O}(1)$ by exhibiting graphs that can be pebbled in space s but where any such pebbling requires time $\Omega\big(n^{s-1}\big)$. We do this by generalizing permutation graphs to multiple layers, where we have k directed path graphs of length n and $k-1$ layers of permutations between the vertices $1, 2, \ldots, n$ in consecutive paths (so that the permutation graphs considered in [130] are 2-layer bit-reversal graphs with paths of length n). We state two theorems below for the black and black-white sequential pebble games, and just as for the 2-layer graphs in [130] our bounds can be stated not just for minimal space but also an arbitrary space parameter s greater than this minimum.

Theorem C.2.9. *Let G be a k-layer bit-reversal graph with paths of length n. Then for any s such that $k+1 \leq s \leq \sqrt{n}$ there exists a sequential black pebbling of G in space s and time $\mathrm{O}(n^k/s^{2k-3})$. Furthermore, every sequential black pebbling of G in space s requires time $\Omega(n^k/s^{2k-3})$.*

Theorem C.2.10. *Let G be a k-layer bit-reversal graph with paths of length n. Then for any s such that $k+1 \leq s \leq \sqrt{n}$ there exists a sequential black-white pebbling of G in space s and time $\mathrm{O}(n^k/s^{2k-2})$. Furthermore, every sequential black-white pebbling of G in space s, requires time $\Omega(n^k/s^{2k-2})$.*

Our proofs of these results are inspired by the reasoning in [130] for 2-layer permutation graphs, but we also need to overcome some new challenges. The essence of the argument is that in order to place a pebble on the jth layer we need to do some work on the preceding layer. If we only have two layers the argument ends here, but when we want to apply the argument recursively we need to be more careful. Indeed, placing pebbles on the $(j-1)$st layer will now require placing more pebbles on the $(j-2)$nd layer, but if we choose the order in which we do the pebble placements wisely, we may be able to reuse part of the work in the $(j-2)$nd layer for several pebble placements in the $(j-1)$st layer. We are able to find a strategy to exploit this insight and obtain optimal upper bounds, but also to make the lower bound argument resilient enough to get asymptotically matching lower bounds.

C.3 Cumulative Space for the Resolution Proof System

We now proceed to describe in more detail the proof complexity results in our paper. We start this section by a brief review of some standard proof complexity preliminaries, after which we discuss how to refine the definition of the resolution proof system to be able to make meaningful and precise claims about maximal space and cumulative space. This then allows us to make the connection to the pebbling results in Section C.2 and what proof complexity implications they have.

A *literal* over a Boolean variable x is either x itself (a *positive literal*) or its negation $\overline{x}$ (a *negative literal*). A *clause* $C = a_1 \vee \cdots \vee a_k$ is a disjunction of literals a_i over pairwise disjoint variables. A *k-clause* is a clause that contains at most k literals. A *CNF formula* $F = C_1 \wedge \cdots \wedge C_m$ is a conjunction of clauses and a *k-CNF formula* is a CNF formula consisting of k-clauses. We think of clauses and CNF formulas as sets: order is irrelevant and there are no repetitions.

The standard definition of a *resolution refutation* $\pi : F \vdash \bot$ of an unsatisfiable CNF formula F—or a *resolution proof* for (the unsatisfiability of) F—is as an ordered sequence of clauses $\pi = (D_1, \ldots, D_t)$ such that $D_t = \bot$ is the empty clause containing no literals, and each clause D_i, $i \in [t]$, is either an *axiom* $D_i \in F$ or is derived from clauses D_j and D_k, $j, k < i$, by the *resolution rule*

$$\frac{B \vee x \qquad C \vee \overline{x}}{B \vee C}\ , \tag{C.3.1}$$

where we refer to $B \vee C$ as the *resolvent over x* of $B \vee x$ and $C \vee \overline{x}$.

In order to study space in general, and cumulative space in particular, we refine the above definition into a family of proof systems as follows.

Definition C.3.1 (Resolution). A resolution refutation $\pi : F \vdash \bot$ of a CNF formula F is a sequence of *configurations*, or sets of clauses, $\pi = (\mathbb{C}_0, \ldots, \mathbb{C}_t)$ such that $\mathbb{C}_0 = \emptyset$, $\bot \in \mathbb{C}_t$, and for all $i \in [t]$ we obtain $\mathbb{C}_i$ from $\mathbb{C}_{i-1}$ by applying exactly one of the following type of rules:

Axiom download Add $A \in F$.

Inference Add D derived from clauses in $\mathbb{C}_{i-1}$.

Erasure Remove clauses from $\mathbb{C}_{i-1}$.

We say that a refutation is (a) *sequential* if at every time step we apply the chosen rule exactly once; (b) *inference-parallel* if only one clause can be downloaded but the inference rule can be applied an arbitrary number of times (but always deriving from $\mathbb{C}_{i-1}$); and (c) *fully parallel* (or just *parallel*) if both axiom download and inference rules can be applied an arbitrary number of times (but note that we cannot mix applications of different rules in the same step). Furthermore, a refutation is said to be (1) *syntactic* if inferences use the resolution rule (C.3.1) and (2) *semantic* if instead any clause D such that $\mathbb{C}_{i-1} \vDash D$ can be inferred immediately.

The *length* of a resolution refutation π is the number of derivation steps t and the *size* is the total number of clauses introduced in downloads and inference steps (counted with repetitions). [5] The *maximal (clause) space*, or just *space*, of π is $\max\{|\mathbb{C}_i| : \mathbb{C}_i \in \pi\}$ and the *cumulative (clause) space* is $\sum_{\mathbb{C}_i \in \pi} |\mathbb{C}_i|$.

Note that Definition C.3.1 yields a total of six different flavours of resolution 1(a)–2(c) depending on the amount of parallelism and on whether inferences are syntactic or semantic. In what follows, we will discuss our motivation for considering these different models and what we can say about them.

A first, general comment is that from a proof complexity point of view we are mainly interested in *syntactic* versions of the proof systems in Definition C.3.1. Strictly speaking, the *semantic* versions are not even propositional proof system in the sense of Cook and Reckhow [66], since we do not know how to verify semantic implications in polynomial time. In any semantic system we can download all axioms in the formula and then derive contradiction in a single inference step, and efficiently verifying such an inference means solving SAT in polynomial time. However, most results on (clause) space in the proof complexity literature actually hold in the stronger semantic setting. For maximal space this is not so surprising, since the semantic and syntactic space measures are within a constant factor of each other [2], but even for trade-offs one tends to get results in the semantic setting for free (with the notable exceptions of [19, 23]).

Syntactic sequential resolution is the standard definition discussed at the beginning of this section (and note that for this version of resolution the length and size measures are essentially the same). A somewhat unsatisfactory feature of this model is that (analogously to what is the case for pebbling) a maximal space lower bound s immediately implies a cumulative space lower bound $\Omega(s^2)$. The reason is completely analogous: since we can only infer one new clause per time step, during the $s/2$ time steps before reaching space s we must have had at least $s/2$ clauses in memory. It turns out, however, that we can actually beat this lower bound in certain settings, and we also remark that cumulative length-space trade-offs do not necessarily follow from such trivial arguments and so make sense even for syntactic sequential resolution.

By allowing parallel application of inference steps we want to try to get away from cumulative space lower bounds that hold only for the trivial reason just discussed. In

[5]For standard resolution as defined in the literature it is most often the case that the length and size definitions coincide, and we could have achieved this here also by counting only axiom download and inference steps when measuring length. Including also erasure steps seems slightly more natural in the current context, however, and also changes the measure by at most a constant factor 2, which is completely immaterial for our purposes.

syntactic inference-parallel resolution we therefore allow clauses to be derived in parallel. As it turns out, anything we are currently able to prove for this model we can also establish for the stronger *semantic inference-parallel resolution* system.

We can also go in the other direction from the syntactic sequential model and introduce a parallelism of sorts by studying *semantic sequential resolution*. As already alluded to, this is a very powerful system since any formula can be refuted in linear size and space by downloading all its axioms in a linear number of steps and then deriving contradiction in just one semantic inference step, but nevertheless the space lower bounds and length-space trade-offs in [27, 28] hold in this model, and can in fact be verified to hold even for semantic inference-parallel resolution.

The most challenging models in terms of lower bounds are the fully parallel ones. *Syntactic parallel resolution* could be viewed as a potentially interesting model for proving lower bounds on parallel SAT solvers using conflict-driven clause learning, where one could imagine an arbitrarily large number of solvers producing resolvents in parallel and having perfect access to shared memory. It is not hard to see that if a standard resolution proof is represented as a DAG in the natural way, then syntactic parallel length, which would be a proxy for execution time, is just the depth of this DAG.

In the semantic model, adding also parallel axiom downloads makes the proof system exceptionally powerful, since now any formula can be refuted in constant length 2, linear size, and linear cumulative space. This seems a bit too strong to be really interesting (and can be viewed as a reason for preferring the inference-parallel version described previously). However, we shall see that even for semantic fully parallel resolution it is still possible to obtain nontrivial trade-off results if the maximal (non-cumulative) space is bounded.

Moving on from this philosophical discourse to a more concrete discussion of results, we note that most of the proof complexity consequences we derive from the pebbling results in Section C.2 are for semantic inference-parallel resolution, and thus hold for all models above except the fully parallel ones. We start by reporting a disappointing fact, however: even in semantic inference-parallel resolution we have the problem that cumulative space is at least maximal space squared.

Lemma C.3.2. *If F requires maximal space s in semantic inference-parallel resolution, then any semantic inference-parallel refutation of F has cumulative space $\Omega(s^2)$.*

Proof. For simplicity let us think of each step in a semantic inference-parallel resolution refutation as being either an inference-plus-erasure step or a download step. Clearly, this can only affect the clause space measure by a factor 2.

An inference-plus-erasure step can be seen as a compression operation. Since the proof system is semantic, we only care about the information contained in a configuration, and since an inference step cannot increase the information but only add explicitly clauses that are already implied by the configuration, there is no need to add any extra clauses on top of the minimum amount needed to encode the semantic information we want the proof to maintain at this point. Therefore, without loss of generality the number of clauses only increases at download steps, and since these are sequential we can conclude that the number of clauses increases by at most 1 at every step.

But this means that we can apply the same argument as for syntactic sequential resolution above: during the $s/2$ time steps preceding a space-s configuration we must have at least $s/2$ clauses in memory, and hence a cumulative lower bound $\Omega(s^2)$ follows. □

It is important to note, though, that Lemma C.3.2 has no implications for cumulative space trade-offs for formulas where the maximal space complexity is at most $O(\sqrt{N})$ measured in the formula size N, since in this setting the max-space-squared argument only implies a trivial $\Omega(N)$ cumulative space lower bound, and we present such trade-off results below that do not follow from Lemma C.3.2. We also report results that asymptotically beat the maximal-space-squared lower bound for cumulative space.

In order to obtain these results, we need to review how our cumulative pebbling results in Section C.2 can be translated to claims about resolution refutations of so-called *XORified pebbling formulas*. We will be very brief here, since all that needs to be done is to read the pebbling-to-resolution reductions in [28] and verify that the proofs work not only for semantic sequential resolution but also for semantic inference-parallel resolution. We just state the reduction that we need below, since we can use it in a completely black-box fashion without knowing any details about what these formulas are. The interested reader is referred to [28] for the missing details.[6]

Theorem C.3.3 (by the proof of Theorem 2.1 in [28]). *Let π be a semantic inference-parallel resolution refutation of a XORified pebbling formula $Peb_G[\oplus]$ in length L, maximal space s, and cumulative clause space c. Then there is a sequential black-white pebbling of the underlying DAG G in time L, space s, and cumulative space c.*

Analogously to what is the case in [28], the generic reduction in Theorem C.3.3 can now be applied to a multitude of different graph families with different pebbling properties to yield CNF formulas with the same properties in resolution. Below we just give a sample of such results that we find particularly interesting.

For maximal space it is known that formulas refutable in linear size $O(N)$ never require space more than $O(N/\log N)$. For cumulative space the lower bound can be truly quadratic, however, beating the max-space-squared bound in Lemma C.3.2 by a factor $\log^2 N$.

Theorem C.3.4. *There is a family of 6-CNF formulas $\{F_N\}_{N\in\mathbb{N}^+}$ of size $\Theta(N)$ that have syntactic sequential resolution refutations in size $O(N)$, and hence also in maximal clause space*

[6] It might be worth noting, though, that just as in [28] our results hold not only for pebbling formulas substituted with exclusive or—substitution with any so-called *non-authoritarian* (or *robust*) function that can never be fixed by restricting any single variable to some value works fine. Binary exclusive or is just the simplest example of such a function, whereas standard or is a simple non-example since setting a single variable to true fixes the value of the function to true.

$\mathrm{O}(N/\log N)$*, but for which any semantic inference-parallel refutations require cumulative clause space* $\Omega(N^2)$.

This theorem follows from studying pebbling formulas defined in terms of *grate graphs* as in [169] and using that the high predecessor-robustness of these graphs imply strong lower bounds on cumulative space as stated in Section C.2.

A natural question is what cumulative space tells us about maximal space, and in particular whether high cumulative space complexity implies that the maximal space complexity must also be large. This might sound intuitively plausible, but turns out to be false in a very strong sense.

Theorem C.3.5. *There is a family of* 6*-CNF formulas* $\{F_N\}_{N\in\mathbb{N}^+}$ *of size* $\Theta(N)$ *that can be refuted in syntactic sequential resolution in size* $\mathrm{O}(N)$ *and also in maximal clause space* $\mathrm{O}(\log N)$*, but for which any semantic inference-parallel refutations require cumulative clause space* $\Omega(N^2/\log N)$.

Here the graphs we need are surprisingly simple, namely butterfly graphs. They again have high predecessor-robustness, but since they are shallow the pebbling formulas generated from them have refutations in small maximal space.

Finally, we turn to the question of length-space trade-offs. We remark that in a cumulative space setting formulas for which small-space proofs require superpolynomial length, as in the strongest results in [28, 23, 19], are not too interesting, since length is trivially a lower bound on cumulative space. Rather, we focus on formulas for which small-space proofs incur only a polynomial blow-up in proof length. Can we find such formulas for which it holds not only that short proofs must have large maximal space s, but where such short proofs must be memory-intensive in that this amount of space s must be used essentially throughout the whole proof? The answer to this question is yes, and one example are pebbling formulas over the bitreversal permutation graphs studied in [130]. The next theorem follows by combining the reduction in Theorem C.3.3 with the fact that bitreversal graphs are dispersed as stated in Section C.2.

Theorem C.3.6. *There is a family of* 6*-CNF formulas* $\{F_N\}_{N\in\mathbb{N}^+}$ *of size* $\Theta(N)$ *such that for any* $s = \mathrm{O}\big(\sqrt{N}\big)$ *the formula* F_N *has a syntactic sequential resolution refutation in size* $\mathrm{O}\big(N^2/s^2\big)$ *and maximal clause space* $\mathrm{O}(s)$*, but any semantic inference-parallel refutation of* F_N *in maximal space* s *requires cumulative clause space* $\Omega(N^2/s)$.

In particular, a proof in maximal space s has length $\Omega(N^2/s^2)$, and if furthermore the proof has length $\mathrm{O}(N^2/s^2)$, then $\Omega(N^2/s^2)$ of the configurations have space $\Omega(s)$. Hence, these formulas have syntactic sequential resolution refutations in simultaneous length $\mathrm{O}(N)$ and space $\mathrm{O}\big(\sqrt{N}\big)$, but any semantic inference-parallel refutation with the same parameters has $\Omega(N)$ configurations with space $\Omega(\sqrt{N})$. We remark that this result makes sense even in the weaker syntactic sequential model, since maximal space $\Omega(\sqrt{N})$ only implies a trivial $\Omega(N)$ cumulative space lower bound.

As already noted, semantic fully parallel resolution is an extremely powerful model, since we can refute any formula with just one (parallel) axiom download step followed by one (semantic) inference step, but if we limit the available space then the usefulness of parallelism is restricted. In fact, the speed-up from parallelism is proportional to the space.

Observation C.3.7. *Let π be a semantic parallel resolution refutation of a formula F in length L, maximal clause space s, and cumulative clause space c. Then there is a semantic sequential refutation of F in length Ls, maximal clause space s, and cumulative clause space cs.*

Proof. Each parallel axiom download or inference adds at most s new clauses, therefore we can simulate it by s sequential axiom downloads or inferences respectively. $\square$

Using Observation C.3.7 we can transfer the trade-offs above from inference-parallel to fully parallel semantic resolution by sacrificing a factor s.

Lemma C.3.8. *Let π be a syntactic sequential resolution refutation of a formula F in length L, maximal space s, and cumulative space c, and let $\ell \in \mathbb{N}^+$ be a positive integer. Then there is a semantic parallel resolution refutation of F in length $3\lceil L/\ell \rceil$, maximal space $s + \lceil \ell/2 \rceil$, and cumulative space $3\lceil c/\ell \rceil + L$.*

Proof sketch. Analogously to the proof of Lemma C.2.3, we divide π into L/ℓ intervals of ℓ steps each. We reorder derivation steps within every interval so that we do all axiom downloads first, inferences next, and removals at the end of the interval. We then collapse each sequence into one axiom download, one inference, and one removal step. $\square$

Let us finally just observe that although proving strong lower bounds for the fully parallel versions of resolution looks like a formidable challenge, which we leave as future work, we can obtain a simple separation between semantic and syntactic fully parallel resolution.

Proposition C.3.9. *Every syntactic, fully parallel resolution refutation of a minimally unsatisfiable CNF formula in space $s \le N$ requires length $N/s + \log s - 2$.*

Proof Sketch. Since the inference rule is binary, the number of useful clauses in the second-to-last-last configuration, namely those used to infer contradiction, is at most 2. Analogously, the number of useful clauses in the ith last configuration is at most 2^i. Hence, in the last $\log s$ steps we see at most $2s$ useful clauses in total. Since we need to see each axiom at least once, we still need at least $N/s - 2$ more steps. $\square$

In particular, any syntactic refutation requires length $\log N$, and a refutation in this length requires space $\Omega(N)$. This is in contrast to semantic refutations, which have proofs in length 2, and no space lower bound other than the trivial N/L.

By way of example, consider a (plain) pebbling formula on a path graph of length N. A syntactic refutation in length $\log N$ requires space $\Omega(N)$, while there exists semantic refutation in length $\log N$ and space $2N/\log N + O(1)$: just download the axioms corresponding to $2N/\log N$ consecutive vertices at a time and infer one new clause.

While this is technically a separation, it is also very brittle. For any integer k, it is possible to find a syntactic proof in length $(1+1/k)\log N$ and space $2kN/\log N + O(1)$. First, download the axioms corresponding to evenly spaced vertices at distance $\log N/k$. Then for $\frac{2}{k}\log N$ steps download the clauses corresponding to the previous and next vertex and do a parallel inference step. Another inference step leaves us with a path of length $N/k\log N$, which we can trivially refute in length $\log N - \log k - \log\log N$ and space $N/k\log N$.

C.4 Pebbling Cumulative Space Lower Bounds and Trade-offs

We denote with $\mathcal{P}_G$, $\mathcal{P}_G^{||}$, $\mathcal{P}_G^{bw}$, $\mathcal{P}_G^{pbsw}$, $\mathcal{P}_G^{||bw}$ the set of all legal sequential black, parallel black, sequential black-white, parallel-black sequential-white, and parallel black-white pebblings of G respectively.

Definition C.4.1 (Pebbling Complexity Notions). We define $|\mathbb{P}| = |B|+|W|$. The space, time, space-time and cumulative space complexities of a pebbling $\mathcal{P} = \{\mathbb{P}_0, \ldots, \mathbb{P}_\tau\} \in \mathcal{P}_G^{||bw}$ are defined as

$$\Pi_t(\mathcal{P}) = \tau\ , \quad \Pi_s(\mathcal{P}) = \max_{i\in[\tau]}|\mathbb{P}_i|\ , \quad \Pi_{st}(\mathcal{P}) = \Pi_t(\mathcal{P})\cdot\Pi_s(\mathcal{P})\ , \quad \text{and} \quad \Pi_{cc}(\mathcal{P}) = \sum_{i\in[\tau]}|\mathbb{P}_i|\ .$$

For a measure $\mu \in \{s,t,st,cc\}$, and a pebbling class $\mathcal{C} \in \{\cdot, ||, bw, pbsw, ||bw\}$, the pebbling complexity measures of G are defined as

$$\Pi_\mu^{\mathcal{C}}(G) = \min_{\mathcal{P}\in\mathcal{P}_G^{\mathcal{C}}} \Pi_\mu(\mathcal{P})\ .$$

C.4.1 Robustness Implies Cumulative Space Lower Bounds

In this section we draw connections between the $\mathcal{G}$-robustness of a graph and its cumulative space complexity for several types of pebbling games and sets $\mathcal{G}$. To do this we first generalize a similar result in [9] to the case of $\mathcal{G}$-robustness. Using this we show the results for two of the black-white pebbling variants considered in this work.

Theorem C.4.2. *Let $\mathcal{G}$ be a class of graphs each with a single sink. For any $d \in \mathbb{N}$ let*

$$\tau_d = \min\{\Pi_t^{\mathcal{C}}(L)\ :\ L \in \mathcal{G},\ \mathsf{size}(L) \geq d\}.$$

If G is an (e,d)-$\mathcal{G}$-robust DAG then for any pebbling class $\mathcal{C} \in \{\cdot, ||, bw, pbsw, ||bw\}$ it holds that $\Pi_{cc}^{\mathcal{C}}(G) > e\tau_d$.

Proof. Fix a pebbling class $\mathcal{C} \in \{\cdot, \|, bw, pbsw, \|bw\}$. For any $\alpha \in \mathbb{N}$ we define the following helpful value:

$$c_\alpha := \Pi^{\mathcal{C}}_{cc}(G)/\tau_\alpha.$$

We will show that for any d there exists a node set B of size $|B| \le c_d$ such that $G - B$ contains no subgraph $L \in \mathcal{G}$ of size at least d, or put differently, that G is not (c_d, d)-$\mathcal{G}$-robust. Note that this implies the theorem.

Fix positive integer d and let $\mathcal{P} = (\mathbb{P}_1, \ldots, \mathbb{P}_m)$ be a (class $\mathcal{C}$) pebbling of G with minimal cumulative space complexity. That is $\sum_{i=1}^{m} |\mathbb{P}_i| = \Pi^{\mathcal{C}}_{cc}(G)$. For each $\mathbb{P}_i$ we define the set $\overline{\mathbb{P}}_i := \{(v, i) : v \in \mathbb{P}_i\}$. For $i \in [d]$ define

$$B_i := \overline{\mathbb{P}}_i \cup \mathbb{P}_{i+\tau_d} \cup \mathbb{P}_{i+2\tau_d} \cdots$$

We observe that by construction $\sum_{i=0}^{d-1} |B_i| = \Pi^{\mathcal{C}}_{cc}(G)$, so the size of the B_i's is c_d on average. Let B be the smallest B_i. Then of course $|B| \le c_d$.

It remains to show that $G - B$ contains no subgraph $L \in \mathcal{G}$ of size (at least) d. Let $L \in \mathcal{G}$ be (any one of) the largest such subgraphs of $G - B$ and let v be its sink. Let $j \in \mathbb{N}$ be such that $(v, j) \in B$ and $\forall j' < j\ (v, j') \notin B$. Intuitively j is index of the first interval (of length $\tau_d - 1$) when v is pebbled in B.

Claim C.4.3. *For any v in graph L such a $j \in \mathbb{N}$ always exists.*

Proof. As L is a subgraph of $G - B$ node v is never pebbled in a time step of the form $i + jt_d$ for some $j \in \mathbb{N}$. However v must be pebbled at some point by $\mathcal{P}$. To see this for the (parallel) black pebbling classes notice that to pebble a node its parents, and so (recursively) all of its predecessors must first be pebbled. Since v is either a sink of G or a predecessor of a sink of G, to legally pebble G node v must also be pebbled. Similarly, for the (parallel) black-white pebbling classes, if $\mathcal{P}$ is legal then it must also pebble node v at some point. To see why this is we first argue, for the sake of contradiction, that if a node u is not pebbled by $\mathcal{P}$ then nor are any of its children. Thus if v is not pebbled then by repeating this argument we see that no sink having v as a predecessor can be pebbled by $\mathcal{P}$ which contradicts $\mathcal{P}$ being legal.

Suppose v is never pebbled. Then clearly its child u can never be pebbled with a black pebble as this would violate rule 1 of Definition C.2.1. Moreover u can not be pebbled with a white pebble as this white pebble can not be removed without violating rule 4 of Definition C.2.1. Yet if it such a white pebble were not removed then the final configuration would still contain that white pebble which would contradict the legality of $\mathcal{P}$. □

Let $\mathcal{P}'$ be that subpebbling for that interval but restricted to the pebbles on L. More formally $\mathcal{P}' = (\mathbb{P}'_{i+j\tau_d+1}, \ldots, \mathbb{P}'_{i+(j+1)\tau_d-1})$ where $\mathbb{P}'_i = \mathbb{P}_i \cap L$.

Claim C.4.4. *$\mathcal{P}'$ is a legal pebbling (for class $\mathcal{C}$) of the graph L.*

Proof. The previous claim established that v, the sink of L, is indeed pebbled by $\mathcal{P}'$.

We first show the claim for the sequential and parallel variants of the black pebbling game. That is we show $\mathcal{P}'$ also satisfies the pebbling rules. Suppose for a second that this were not the case, then this would mean that $\mathcal{P}$ would also violate the rule in the same step. Indeed, when constructing $\mathcal{P}'$ from $\mathcal{P}$ we did not remove any pebbles from nodes in L. So if one is missing in $\mathcal{P}'$ (thereby violating rule 2) then it is also missing for $\mathcal{P}$. This concludes the parallel black pebbling case. For the sequential case it remains only to observe that we added no new pebbles when constructing $\mathcal{P}'$. So if $\mathcal{P}$ places no more than 1 pebble per step then so does $\mathcal{P}'$.

Next we show the claim for both of the black-white pebbling variants. That is we show that if $\mathcal{P}'$ violates rule 2 3 and 3′ then so does $\mathcal{P}$. Placing a black pebble illegally in $\mathcal{P}'$ implies that it was already placed illegally in $\mathcal{P}$ for the same reason as just described above. Similarly, removing a white pebble illegally in $\mathcal{P}'$ occurs only when one of its parents wasn't currently pebbled. However then that parent would also not be pebbled at this point in $\mathcal{P}$. Finally clearly no more pebbles are being placed in $\mathcal{P}'$ than in $\mathcal{P}$ since the former is obtained from the later by removing pebbles. This completes the proof of the claim. □

To complete the proof of the theorem it remains only to observe that $\mathcal{P}'$ is a legal pebbling of L lasting for less than τ_d steps. Thus by definition of τ_d it must be that L has size less than d. □

As a first corollary to this theorem we obtain the following relations between depth-robustness and cumulative space complexity for each of the considered pebbling games. Together with the constructions of [150, 7, 169] they imply high space complexity graphs for each of the pebbling game.

Corollary C.4.5. *Let G be (e,d)-depth-robust. Then the following holds:*

$$\Pi^{\cdot}_{cc}(G) > ed \qquad \Pi^{\|}_{cc}(G) > ed \qquad \Pi^{bw}_{cc}(G) > ed \qquad \Pi^{pbsw}_{cc}(G) = \Omega(e\sqrt{d}).$$

The first inequality is implied by the second (as any legal sequential black pebbling of G is also a legal parallel black pebbling). The second inequality is proven in [9]. We now show the remaining two.

Proof. Let $\mathcal{G}$ contain only simple directed paths as in the definition of depth-robustness.

To see the third inequality we must show that in the sequential black-white pebbling game $\tau_d \geq d$ (where τ_d is defined as in Theorem C.4.2). Put simply, we must show that no path of length d can be pebbled in less than d steps. But by Definition C.2.1, at no step can more than one pebble be placed on G. Moreover, just as argued already in the proof of Theorem C.4.2 legally pebbling a path (or any DAG for that matter) requires placing a pebble at least once on each node. So clearly a path of length d requires at least d time.

The fourth inequality is implied by the following claim together with Theorem C.4.2 implies the fourth and final inequality of the corollary.

Claim C.4.6. *In the parallel-black sequential-white pebbling game a path of length d requires $\Omega(\sqrt{d})$ time to pebble.*

Proof. Let L_d be the path of length d. To see why $\Pi_t^{pbsw}(L_d) = \Omega(\sqrt{d})$ let there be a legal pebbling of L_d and suppose it runs for at most $t = o(\sqrt{d})$ steps. As argued already in the proof of Theorem C.4.2 above, for the pebbling to be legal, each node in L_d must have been pebbled at some point. However at most $o(\sqrt{d})$ white pebbles can be placed on L_d as they must be placed one step at a time. Therefore there must be (at least) one sub-path in L_d of length $d/t = \Omega(\sqrt{d})$ to be pebbled only with (parallel) black pebbles. However a node can not be pebbled with a black pebble before its parents are pebbled. Thus such the sub-path requires $\Omega(\sqrt{d})$ steps before it can be pebbled using only black pebbles. This is a contradiction to $t = o(\sqrt{d})$. □

□

As a second corollary to Theorem C.4.2 we get the following relation between predecessor-robustness and black-white cumulative space complexity. In particular the linear sized grates of [169] have $\Theta(n^2)$ black-white cumulative space complexity.

Corollary C.4.7. *Let G be (e,d)-predecessor-robust. Then $\Pi_{cc}^{bw}(G) > ed$.*

Proof. It suffices to show that $\min\{\Pi_t^{bw}(G) \,:\, G \in \mathcal{G},\ \mathsf{size}(G) \geq d\} = d$. But this follows trivially since in the black-white game at most one pebble may be placed per turn. Yet, as shown in the proof of Theorem C.4.2, all nodes of a graph must be pebbled at least once in a legal pebbling. Thus $\Pi_t^{bw}(G) \geq \mathsf{size}(G)$ which completes the proof. □

C.4.2 Dispersion and Cumulative Space Trade-offs

In [9] it is proven that dispersion implies cumulative space lower bounds for parallel black pebbling. In this section we show that subgraph-dispersion implies both cumulative space lower bounds and cumulative space trade-offs for black-white pebbling. We also prove that random permutations are path-dispersed and therefore exhibit such trade-offs.

Dispersion Implies Cumulative Space Trade-offs

Intuitively, A DAG is (k,z,g)-path-dispersed if it contains a path that can be partitioned into k segments such that each segment has at least z disjoint incoming paths of length at least g. The following definitions make this concept precise.

Definition C.4.8. *A DAG is (k,z,g)-path-dispersed if it contains a path $\mathcal{L}$ that can be partitioned into k paths, $L^1, L^2, \ldots, L^k$, such that for every L^i there are z paths, $\phi_1^i, \phi_2^i, \ldots, \phi_z^i$, each with at least g vertices that satisfy the following:*

- *for every $i \in [k]$ and every $j, j' \in [z]$ such that $j \neq j'$, then $\phi_j^i \cap \phi_{j'}^i = \emptyset$;*

- *for every $i, i' \in [k]$ and every $j \in [z]$, then $L^i \cap \phi_j^{i'} = \emptyset$; and*
- *for every $i \in [k]$ and every $j \in [z]$ there is a vertex $v_j^i \in L^i$ such that there is an edge from the sink of ϕ_j^i to v_j^i and moreover, if $j' \in [z]$ and $j \neq j'$, then $v_j^i \neq v_{j'}^i$.*

Note that the paths coming into one segment have to be disjoint among themselves, but not necessarily among paths coming into other segments.

Definition C.4.9. *A DAG is (k,z,g)-subgraph-dispersed if it contains k disjoint subgraphs, $L^1, L^2, \dots, L^k$, such that for every L^i there are z subgraphs, $\phi_1^i, \phi_2^i, \dots, \phi_z^i$, each with at least g vertices that satisfy the following:*

- *for every $i \in [k]$ and every $j, j' \in [z]$ such that $j \neq j'$, then $\phi_j^i \cap \phi_{j'}^i = \emptyset$;*
- *for every $i, i' \in [k]$ and every $j \in [z]$, then $L^i \cap \phi_j^{i'} = \emptyset$; and*
- *for every $i \in [k]$ and every $j \in [z]$ there is a vertex $v_j^i \in L^i$ such that v_j^i is the only sink of the subgraph induced by $\{v_j^i\} \cup V(\phi_j^i)$ and moreover, if $j' \in [z]$ and $j \neq j'$, then $v_j^i \neq v_{j'}^i$.*

Note that a (k,z,g)-path-dispersed graph is also (k,z,g)-subgraph-dispersed. We prove the following lemma for the latter definition.

Lemma C.4.10. *If G is a (k,z,g)-subgraph-dispersed graph, then*

$$\Pi_{cc}^{bw}(G) \geq k \min\{gz/2, z^2/4\} \ .$$

Furthermore, if $\mathcal{P} \in \mathcal{P}^{bw}$ is a sequential black-white pebbling of G in space $\Pi_s(\mathcal{P}) = s \leq z/2$, then

$$\Pi_{cc}(\mathcal{P}) \geq k \min\{g(z-s), g(z-2s)+s^2\} \ .$$

We prove that two subclasses of permutation graphs are path-dispersed and use Lemma C.4.10 to prove cumulative space trade-offs.

Proof. Consider k disjoint subgraphs, $L^1, L^2, \dots, L^k$, and the corresponding subgraphs ϕ_j^i for $i \in [k]$ and $j \in [z]$ that witness the fact that G is (k,z,g)-subgraph-dispersed graph. We refer to the subgraphs ϕ_j^i as *incoming-subgraphs*.

We keep an account for each L^i. For each pebbling move, we charge some of the pebbles in the configuration to each account, ensuring that we do not charge the same pebble at the same configuration to more than one account. It is enough to prove that $\Pi_{cc}(L^i) \geq \min\{gz, g(z-2s)+s^2\}$ for each subgraph L^i.

We charge pebbles according to the following rules:

1. When a black pebble is placed or a white pebble is removed from L^i, charge all pebbles on incoming-subgraphs to the account of L^i.

2. When a black is placed or a white is removed from some incoming-subgraph, charge, for all $i' \in [k]$, all pebbles on $L^{i'}$ to the account of $L^{i'}$.

Observe that at every configuration a pebble is never charged to two different segments L^i. This is so because if a black pebble is placed or a white pebble is removed from L^i then it is not being placed or removed from any ϕ nor from any other $L^{i'}$, $i \neq i'$, because these subgraphs are disjoint and hence any pebble charged at this configuration is only being charged to L^i. Moreover, if a black is placed or a white is removed from some ϕ^i_j, then the pebbles charged to each L^i are only the pebbles on L^i, and since theses subgraphs are disjoint, no pebble is being charged twice.

We now focus on a given subgraph L^i, so we drop the superscript and denote it L. We consider a time interval during which L is pebbled, i.e., there is no pebble on L before the the interval, every vertex in L is pebbled at some point, there is no pebble on L after the interval, and there is some pebble on L at all times during the interval. This interval exists because L has a unique sink.

We label incoming-subgraphs according to the time j they are first used, i.e., a black pebble is placed or a white pebble is removed from the vertex in L which is the sink of the incoming-subgraph. Let $\phi_1, \phi_2, \ldots, \phi_z$ be the incoming-subgraphs according to this labelling.

For each path ϕ_j we distinguish four cases depending on when the path is empty.

Case 1 (1): There is a pebble on ϕ_j during all of the interval. We charge at least $|L| \geq z$ pebbles according to rule 1, since there is always a pebble on ϕ_j during the interval.

Case 2 (010): The path ϕ_j is completely pebbled and unpebbled within the interval. We charge at least $|\phi_j| \geq g$ pebbles according to rule 2, since there is always a pebble on L while ϕ_j is pebbled and unpebbled.

Case 3 (01): There is no pebble on ϕ_j at the beginning of the interval, and there is some pebble on ϕ_j at all times since the first pebble is placed on ϕ_j. We charge at least $|L| - j \geq z - j$ pebbles according to rule 1, since there is always a pebble on ϕ_j after time j.

Case 4 (10): There is no pebble on ϕ_j at the end of the interval, and there is some pebble on ϕ_j at all times before the last pebble is removed from ϕ_j. We charge at least j pebbles according to rule 1, since there is always a pebble on ϕ_j before time j.

If we group the incoming-subgraphs by cases, we have that $\Pi_{cc}(L) \geq z|\Phi_1| + g|\Phi_{010}| + \sum_{j \in \Phi_{01}}(z-j) + \sum_{j \in \Phi_{10}} j$. The minimum is attained when $\{z - j : \phi_j \in \Phi_{01}\} = \{j : \phi_j \in \Phi_{10}\} = [|\Phi_{01}|]$, in which case we have $\Pi_{cc}(L) \geq z|\Phi_1| + g|\Phi_{010}| + |\Phi_{01}|(|\Phi_{01}|+1)$. Without any more restrictions, there are two possible minima: if $g \geq z/2$, then the minimum is attained when $|\Phi_{01}| = |\Phi_{10}| = z/2$, in which case we get $\Pi_{cc}(L) \geq z^2/4$; and if $g < z/2$ then the minimum is attained when $|\Phi_{01}| = |\Phi_{10}| = g$ and $|\Phi_{010}| = z - 2g$, in which case we get $\Pi_{cc}(L) \geq g(z-g) \geq gz/2$. If we enforce that the space is at most $s \leq z/2$, then $|\Phi_1 \cup \Phi_{01}| \leq s$ and $|\Phi_1 \cup \Phi_{10}| \leq s$. Therefore, if $g \geq s$ the first minimum is attained when $|\Phi_{010}| = z - 2s$ and $|\Phi_{01}| = |\Phi_{10}| = s$, giving $\Pi_{cc}(L) \geq g(z-2s)+s^2$; and if $g < s$ then the second minimum yields $\Pi_{cc}(L) \geq g(z-g) \geq g(z-s)$. □

Permutation Graphs are Disperse

Definition C.4.11. *Given a permutation $\sigma \in \mathfrak{S}_n$, the* permutation graph $G(\sigma)$ *is the graph $G = (W, E)$ such that $W = \{x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n\}$, and $E = \{(x_i, x_{i+1}), (y_i, y_{i+1}), (x_i, y_{\sigma(i)}) \mid 1 \leq i \leq n\}$.*

Recall that for $n = 2^\chi$ the bit-reversal permutation reverses the binary representation of a number. This is, if $j = (b_1 \ldots b_\chi)_{(2)}$ then $\sigma(j) = (b_\chi \ldots b_1)_{(2)}$. From Lemma C.4.10 we get the following result.

Corollary C.4.12. *If G be a bit-reversal permutation graph on $2n$ vertices, then*

$$\Pi_{cc}^{bw}(\mathcal{P}) \geq \frac{n^{3/2}}{4} \ .$$

Furthermore, if $\mathcal{P} \in \mathcal{P}^{bw}$ is a sequential black-white pebbling of G in space $s = \Pi_s(\mathcal{P})$, then

$$\Pi_{cc}(\mathcal{P}) \geq \frac{n^2}{9s} + \frac{n}{6} \ .$$

Proof. For every $z \leq n$ that is a power of 2, by partitioning the path $(y_1, y_2, \ldots, y_n)$ in G in n/z paths of length z, it is easy to see that G is $(n/z, z, n/z)$-path-dispersed. From Lemma C.4.10 we get that $\Pi_{cc}^{bw}(\mathcal{P}) \geq n^{3/2}/4$ by setting $z = 2^{\lceil \chi/2 \rceil}$.

Moreover, we also get that if $\mathcal{P}$ is a sequential black-white pebbling of G in space $s = \Pi_s(\mathcal{P})$, then

$$\Pi_{cc}(\mathcal{P}) \geq \frac{n^2(z-2s)}{z^2} + \frac{ns}{z} \min\left\{\frac{n}{z}, s\right\} \ . \tag{C.4.1}$$

If $s \geq n/3$, $\Pi_{cc}(\mathcal{P}) \geq \frac{n^2}{9s} + \frac{n}{6}$ holds trivially. So let us assume that $s \leq n/3$, and hence setting $z = 2^{\lceil \log 3s \rceil}$, we have $z \leq n$. Note that $3s \leq z \leq 6s$, and that $\min_{3s \leq z \leq 6s}(z-2s)/z^2 = 1/9s$. The result then follows by observing that $\min\{n/z, s\} \geq 1$. □

We now focus on the study of random permutation graphs. We will define a distribution which is equivalent to choosing a permutation uniformly at random.

Let $j_1, j_2 \ldots, j_n, k_1, k_2 \ldots, k_n$ be integers in $[n]$ chosen independently uniformly at random. Let $G' = (X \cup Y, A)$ be a digraph such that

- $X = \{x_1, x_2, \ldots, x_n\}$,
- $Y = \{y_1, y_2, \ldots, y_n\}$, and

- $A = \{(x_i, x_{i+1}), (y_i, y_{i+1}), (x_{j_i}, y_{k_i}) | 1 \leq i \leq n\}$.

We perform two operations on G' to obtain the graph G: edge contraction and degree reduction. Edge contraction consists of contracting one by one any edge (x_i, x_{i+1}) or (y_i, y_{i+1}) such that one of the endpoints is not adjacent to at least one vertex in the opposite partition. Degree reduction consists of substituting vertices of high degree by paths and distributing the edges adjacent to that vertex to the vertices on the path. Formally, for all vertices $x \in X$ that are adjacent to $d \geq 2$ vertices in Y, say $\{y_{i_1}, \dots, y_{i_d}\}$, substitute x by a path on d vertices, say $\{x_{i_1}, \dots, x_{i_d}\}$, and for $i \in [d]$ include the edge $(x_{i_1}, y_{i_{\sigma(1)}})$, where σ is a random permutation in $\mathfrak{S}_d$. The analogous operation is done for vertices $y \in Y$ of high degree.

We are now ready to prove the following theorem.

Theorem C.4.13. *A random permutation graph is whp an $(\alpha k, (\beta - 1/2)z, \alpha k)$-path-dispersed graph, for $\alpha, \beta < 1$, $\alpha + \beta - \alpha\beta \ll (1 - e^{-n/kz})$ and $kz = \mathrm{O}(n)$.*

Proof. We define a bipartite graph H from G' in the following manner. Let $H = ((U, V), F)$ with $U = \{u_1, u_2, \dots, u_k\}$ and $V = \{v_1, v_2, \dots, v_z\}$. Each node $u_i \in U$ is a super-node that corresponds to $\{x_{k(i-1)+1}, x_{k(i-1)+2}, \dots, x_{k(i-1)+n/k}\}$ and similarly each node $v_i \in V$ is a super-node that corresponds to $\{y_{z(i-1)+1}, y_{z(i-1)+2}, \dots, y_{z(i-1)+n/z}\}$. For $w \in U \cup V$ let $S(w) \subset X \cup Y$ be the set of vertices that w represents. There is an edge from $u \in U$ to $v \in V$ if there is at least one edge from some vertex in $S(u)$ to some vertex in $S(v)$ (see Figure C.2).

Theorem C.4.13 follows from the following claim which we will prove later on.

Claim C.4.14. *Let Z be an integer random variable in $[1, kz]$ that represents the number of edges in H. Let $p = (1 - 1/kz)^n$ and $q = (1 - 1/(kz - 1))^n$. Then*

$$\Pr[Z \leq \delta kz] \leq \frac{p(q-p)}{(1-p-\delta)^2}, \text{ for } \delta \leq (1-p) \ .$$

Note that $\lim_{n\to\infty} q - p = 0$ and that $\lim_{n\to\infty} p \ll 1$ if $kz = \mathrm{O}(n)$. Hence for any $\delta \ll (1-p)$ the probability that Z is less than δkz goes to 0 as n increases. In other words, with high probability the number of edges in the bipartite graph is extremely close to its expected value.

Claim C.4.14 implies that for any α, β that satisfy $\alpha + \beta - \alpha\beta \leq \delta \leq (1-p)$, then with probability at least $1 - \frac{p(q-p)}{(1-p-\delta)^2}$ there are αk nodes in U with outdegree at least βz, and βz nodes in V with indegree at least αk. This is indeed the case, because if there were not αk nodes in U with outdegree at least βz, then there would be strictly less than $(1-\alpha)\beta kz + \alpha kz = (\alpha + \beta - \alpha\beta)kz \leq \delta kz$ edges in H, and the same holds if there were not βz nodes in V with indegree at least αk.

Consider a node $u \in U$ with outdegree at least βz. We say an edge adjacent to u is *significant* if its other endpoint is a vertex in V with indegree at least αk. Observe that, since there are at least βz nodes in V with indegree at least αk, there must be are at least

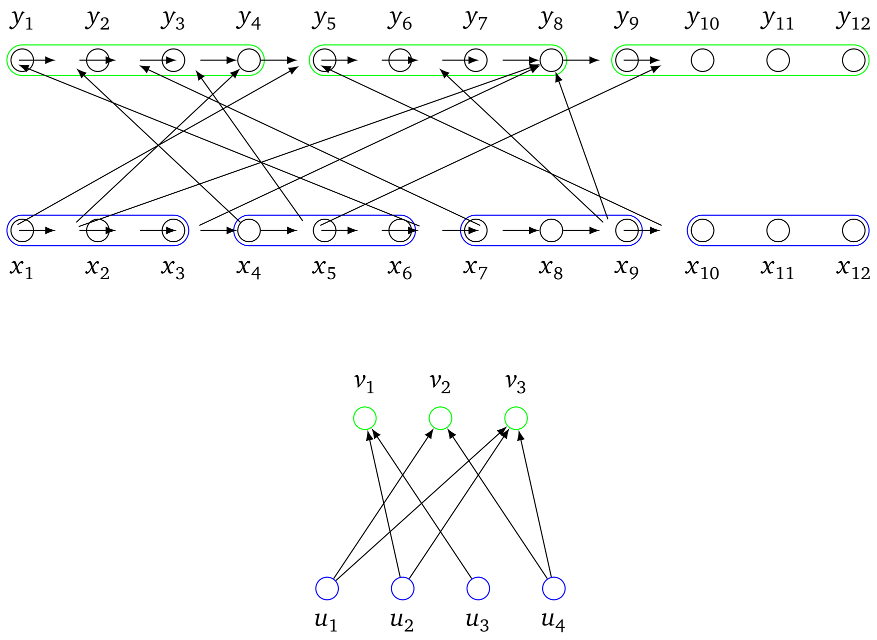

y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10} y_{11} y_{12}
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}
v_1 v_2 v_3
u_1 u_2 u_3 u_4

Figure C.2: An example of a random graph G' and its corresponding bipartite graph H, for $k = 4$ and $z = 3$

$(2\beta-1)z$ significant edges adjacent to u. By choosing every other significant edge, we can guarantee that there is at least an αk-distance in G between the corresponding endpoint nodes. Finally, since there are at least αk nodes in U with outdegree at least βz, we conclude that G is $(\alpha k, (\beta - 1/2)z, \alpha k)$-path-dispersed with probability $1 - \frac{p(q-p)}{(1-p-\delta)^2}$, for $\alpha + \beta - \alpha\beta \leq \delta \leq (1-p)$. Choosing $\delta \ll (1-e^{-n/kz}) \leq (1-p)$ yields the theorem. □

Proof of Claim C.4.14. Let us consider the probability of a given edge not being present in H. Let $u, u' \in U$ and $v, v' \in V$ be nodes chosen independently at random conditioned on $u \neq u'$ or $v \neq v'$. We define $p = \Pr[uv \notin E] = (1-1/kz)^n$ and $q = \Pr[u'v' \notin E \mid uv \notin E] = (1-1/(kz-1))^n$, so that $\Pr[uv \notin E \text{ and } u'v' \notin E] = pq$.

From Chebyshev's inequality, we have that

$$\Pr[Z \leq \delta kz] \leq \Pr[|\mathrm{E}[Z] - Z| \geq |\mathrm{E}[Z] - \delta kz|] \leq \frac{\mathrm{Var}(Z)}{(\mathrm{E}[Z] - \delta kz)^2} .$$

Let X_v be an integer random variable in $[1, z]$ that represents the degree of $v \in V$. Note that $Z = \sum_{v \in V} X_v$. Let Y_{uv} be a binary random variable that indicate is $uv \in E$.

Therefore, we obtain

$$\mathrm{E}[Z^2] = \sum_{v,v' \in V, v \neq v'} \mathrm{E}[X_v X_{v'}] = k\,\mathrm{E}[X_v^2] + k(k-1)\,\mathrm{E}[X_v X_{v' \neq v}] \tag{C.4.2}$$

$$= (kz)^2(1-2p+pq) - kzp(q-p) \ , \tag{C.4.3}$$

which follows from

$$\mathrm{E}[X_v X_{v'}] = \sum_{u,u' \in U} \mathrm{E}[Y_{uv} Y_{u'v'}] = z^2 \Pr[Y_{uv} = 1 \text{ and } Y_{u'v'} = 1] = z^2(1-2p+pq) \ , \tag{C.4.4}$$

where $v' \neq v$, and from

$$\mathrm{E}[X_v^2] = \sum_{u,u' \in U} \mathrm{E}[Y_{uv} Y_{u'v}] = z(1-p) + z(z-1)(1-2p+pq) \ . \tag{C.4.5}$$

Moreover, from linearity of expectation we get that $\mathrm{E}[Z] = kz(1-p)$. Hence,

$$\Pr[Z \leq \delta kz] \leq \frac{\mathrm{Var}(Z)}{(\mathrm{E}[Z] - \delta kz)^2} \leq \frac{(kz)^2 p(q-p)}{(kz(1-p) - \delta kz)^2} \leq \frac{p(q-p)}{(1-p-\delta)^2} \ . \qquad \square$$

Corollary C.4.15. *If G is a random permutation graph, then whp $\Pi_{cc}^{Pbw}(G) \geq n^{3/2}/125$. and any black-white pebbling $\mathcal{P}$ of G in space s, is such that $\Pi_{cc}(\mathcal{P}) \geq n^2/1250s$.*

Proof. Taking $k = n/4z$, $\alpha = 4/5$ and $\beta = 9/10$, we get $\alpha + \beta - \alpha\beta = 98/100$ and $(1 - e^{-n/kz}) = (1 - e^{-4}) > 0.9816$, and hence the conditions of Theorem C.4.13 are satisfied. Theorem C.4.13 therefore implies that with high probability G is $(n/5z, 2z/5, n/5z)$-path-dispersed. By setting $z = \sqrt{n}$, Lemma C.4.10 implies that $\Pi_{cc}^{Pbw}(G) \geq n^{3/2}/5^3$. and by setting $z = 10s$, Lemma C.4.10 implies that any black-white pebbling $\mathcal{P}$ of G in space s, is such that $\Pi_{cc}(\mathcal{P}) \geq n^2/1250s$. $\square$

C.4.3 Pebbling in Small Space Can Require Maximum Length

In this section we prove Theorems C.2.9 and C.2.10. We work with the following graphs.

Definition C.4.11 (Restated). *Given a permutation $\sigma \in \mathfrak{S}_n$, the* permutation graph $G(\sigma)$ *is the graph $G = (W, E)$ such that $W = \{x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n\}$, and $E = \{(x_i, x_{i+1}), (y_i, y_{i+1}), (x_i, y_{\sigma(i)}) \mid 1 \leq i \leq n\}$.*

For $n = 2^\chi$, the bit-reversal permutation reverses the binary representation of a number. This is, if $j = (b_1 \ldots b_\chi)_{(2)}$ then $\sigma(j) = (b_\chi \ldots b_1)_{(2)}$. Observe that the bit-reversal permutation is an involution, this is $\sigma^2 = 1$. For simplicity we shall assume that χ is even.

As a warm-up before proving general upper bounds, we begin by proving upper bounds for space $\sqrt{n}$. The main ideas we need are already in Lemma C.4.16.

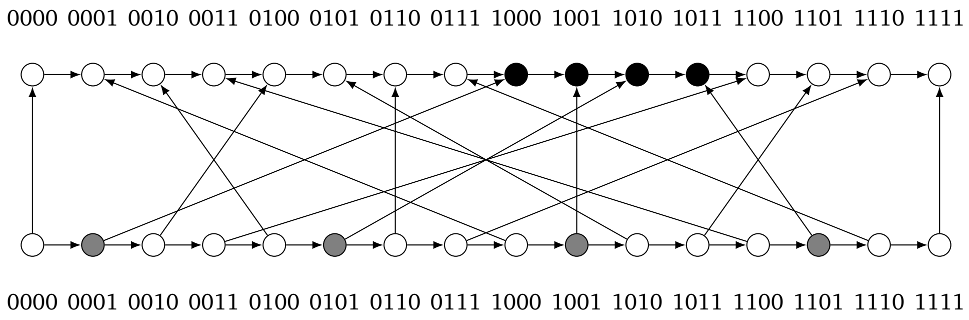


Figure C.3: The predecessors of a block are evenly distributed

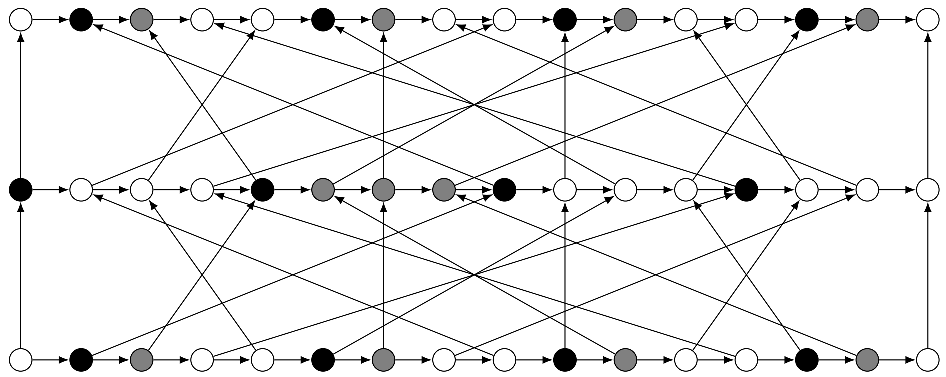


Figure C.4: An operation on layer i requires the same operation on layer $i-2$

Lemma C.4.16. *Let G be a k-layer bit-reversal graph. There is a sequential black pebbling $\mathcal{P} \in \mathcal{P}_G$ in space $\Pi_s(\mathcal{P}) = k\sqrt{n} + O(1)$ and time $\Pi_t(\mathcal{P}) \leq k^2 n^{3/2}$.*

Proof. We divide each path in $\sqrt{n}$ blocks of length $\sqrt{n}$. Observe that the image of a block are $\sqrt{n}$ evenly separated vertices at distance $\sqrt{n}$ (see Figure C.3, where the predecessors of the black block marked in black are marked in grey).

We pebble the graph layer by layer, keeping the invariant that when we begin to pebble a layer i there are $\sqrt{n}$ pebbles in each layer below, one at the beginning of each block. We call layers even or odd depending on the distance to layer i. We have two types of operations, each of which consists of several moves:

1. advance a (new) pebble through a block on an odd layer, and

2. advance each pebble on an even layer to the next position.

To pebble a layer we add a pebble at the beginning and then we proceed in rounds. In a round we apply $\sqrt{n}$ times operation 2, and then we add a pebble at the beginning. At each round the number of pebbles in the layer increases by 1, until after round $\sqrt{n}-1$ we have $\sqrt{n}$ pebbles on positions that are multiples of $\sqrt{n}$.

We maintain the following invariant between operations: there are $\sqrt{n}$ pebbles on each layer below, and while pebbles on odd layers are on positions that are multiples of $\sqrt{n}$, after the j-th operation on layer i pebbles on even layers are on positions congruent with j module $\sqrt{n}$. Observe that after a round, this is $\sqrt{n}$ operations, all pebbles are on positions that are multiples of $\sqrt{n}$.

We claim that if the operation invariant holds, then the cost of an operation is $\sqrt{n}$ plus the cost of the opposite operation in the layer below, this is time $i\sqrt{n}$ on layer i.

Indeed, if want to advance all pebbles on an even layer ℓ to the next position, we just need to advance one pebble along the preimage of the destination vertices, which is the $\sigma^{-1}(j)$-th block in layer $\ell-1$, and advance the corresponding pebble in layer ℓ after each advance in layer $\ell-1$. Since there is a pebble at the beginning of each block, we do not need more operations in layer $\ell-1$ (but we need operations in layer $\ell-2$).

If instead we want to advance a pebble through block $\sigma^{-1}(j)$ on an odd layer ℓ, then we need to have pebbles in the preimage of the block, which is the set of vertices in layer $\ell-1$ congruent with $\sigma^{-2}(j)=j$ modulo $\sqrt{n}$. By the operation invariant, the set of vertices in layer $\ell-1$ congruent with $j-1$ modulo $\sqrt{n}$ all have pebbles, so it is enough to advance each pebble on layer $\ell-1$ to the next position (see Figure C.4, where the configuration after the 2-nd operation is marked in black and the nodes needed for the 3-rd operation are marked in grey).

The total time to pebble the graph is the time to pebble each layer, this is $\sum_{i=1}^{k} in^{3/2} \leq k^2 n^{3/2}$, plus one additional round on layer k in order to reach the sink, which needs kn steps. □

Lemma C.4.17. *Let G be a k-layer bit-reversal graph. There is a sequential black-white pebbling $\mathcal{P} \in \mathcal{P}_G^{bw}$ in space $\Pi_s(\mathcal{P}) = 2k\sqrt{n} + O(1)$ and time $\Pi_t(\mathcal{P}) \leq kn$.*

Proof. We use the same approach as the black-only case, except that now we can initially setup each layer with white pebbles, and we convert these white pebbles into black whenever we have the opportunity.

More precisely, we initially place $k\sqrt{n}$ white pebbles, each at the beginning of one block. Then we do $\sqrt{n}$ even layer operations on layer k. Observe that at the end of each operation on each odd layer we replace a white pebble with a black pebble, and at the very end we replace all white pebbles with black pebbles on each even layer.

The final cost is the cost to do $\sqrt{n}$ advance operations on the last layer, this is time kn. □

The general upper bounds use the same approach, except that some times the blocks we want to pebble through do not have a pebble at the beginning, so we have to so a

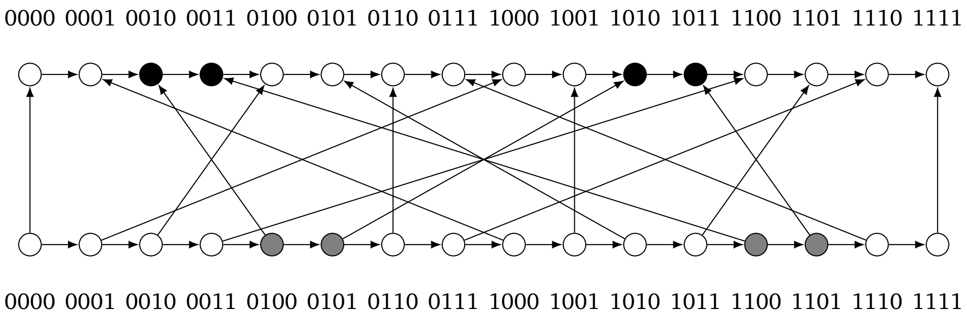


Figure C.5: The predecessors of evenly distributed blocks are evenly distributed blocks

setup phase. By choosing the order in which we pebble blocks, however, we are able to reuse the same setup for pebbling s consecutive blocks, thus saving on time.

Lemma C.4.18. *Let G be a k-layer bit-reversal graph. There is a sequential black pebbling $\mathcal{P} \in \mathcal{P}_G^{bw}$ in space $\Pi_s(\mathcal{P}) = 2k^2 s + O(1)$ and time $\Pi_t(\mathcal{P}) \le n^k/s^{2k-3} + O(n^{k-1})$.*

Proof. We divide each path in n/s blocks of length s. Observe that the preimage of s blocks at distance n/s are s (different) blocks at distance n/s (see Figure C.5).

Our approach is again to pebble the graph layer by layer, but now we only keep s pebbles in each layer. More precisely, when we begin to pebble a layer i there are s pebbles on positions that are multiples of n/s.

We have one type of operation: advance every pebble on layer i through a block. This operation combines the two operations on Lemma C.4.16, and in fact it uses them internally.

In order to do an operation on layer i, we setup the pebbles on each layer below so that layer ℓ is the preimage of layer $\ell+1$ for $\ell \in [i-1]$. Then, for s rounds, we do the following, in ascending layer order. On an even layer, we advance each pebble to the next position. On an odd layer, we advance a pebble through whichever block happens to be the image of the previous layer.

Let $T(i)$ be the time of doing one operation on layer i and $\Sigma(i) = \sum_{\ell=1}^{i} T(i)$. The time to setup layer ℓ is at most the time of doing n/s^2 operations on layer ℓ, which gives a cost of $\sum_{\ell \in [i-1]} n/s^2 T(\ell)$ for the setup phase. Then for each round we only need to do s moves in each layer, which gives a cost of is^2 for s rounds.

We get the recurrence

$$\begin{aligned} T(1) &= s^2 & T(i) &= (n/s^2)\Sigma(i-1) + is^2 \\ \Sigma(1) &= T(1) & \Sigma(i) &= \Sigma(i-1) + T(i) \ , \end{aligned}$$

which we can solve to get

$$T(i) = \sum_{\ell \in [i]} \binom{i}{\ell - 1} n^{i-\ell}/s^{2(i-\ell)}$$
$$\Sigma(i) = \sum_{\ell \in [i]} \binom{i+1}{\ell - 1} n^{i-\ell}/s^{2(i-\ell)} \quad .$$

The total time to pebble the graph is the time of doing n/s operations on each layer from 1 to k, that is $(n/s)\Sigma(k) = n^k/s^{2k-3} + \mathrm{O}(n^{k-1})$. □

Lemma C.4.19. *Let G be a k-layer bit-reversal graph. There is a sequential black-white pebbling $\mathcal{P} \in \mathcal{P}_G^{bw}$ in space $\Pi_s(\mathcal{P}) = 2k^2 s + \mathrm{O}(1)$ and time $\Pi_t(\mathcal{P}) \leq n^k/s^{2k-2} + \mathrm{O}(n^{k-1})$.*

Proof. As in the proof of Lemma C.4.17, we begin by adding s white pebbles on each layer, so that we just need to do n/s^2 operations in each layer. □

We prove the lower bounds for space smaller than $\sqrt{n}$ since for space larger than $\sqrt{n}$ we can just observe that a k-layer permutation graph contains a 2-layer permutation graph and use the lower bounds in [130].

Lemma C.4.20. *Let G be a k-layer bit-reversal graph. Let $\mathcal{P} \in \mathcal{P}_G$ be a sequential black pebbling in space $\Pi_s(\mathcal{P}) = s \leq \sqrt{n}/4$. Then*

$$\Pi_t(\mathcal{P}) \geq n^k/2^{3k}s^{2k-3} \quad .$$

Proof. It is enough to prove that $T \geq n^k/2^{3k-1}s^{2k-3}$ for s a power of 2. Consider a pebbling in space s. We divide each path in $n/2s$ blocks of length $2s$.

We show that pebbling a block, this is starting with an empty block and placing a pebble on the sink, on the i-th layer requires time $n^{i-1}/2^{3i-2}s^{2i-4}$ for $i \geq 2$.

Observe that to pebble the graph we have to consecutively pebble $n/2s$ blocks on the k-th layer, for a total of $n^k/2^{3k-1}s^{2k-3}$ as we wanted to show.

For $i = 2$, consider the $2s$ predecessors in layer 1 of the block we want to pebble. By construction of the bit-reversal permutation, the distance between these predecessors is $n/2s$. Since there are at most $s-1$ pebbles in layer 1, there are at least $s+1$ such predecessors whose closest pebble is at distance $n/2s$. Therefore, at least $n/2$ steps on layer 1 are needed before each predecessor has a pebble.

For $i > 2$, by the same argument there is at least one predecessor in layer $i-1$ whose closest pebble is at distance $n/2s$, therefore we can find $n/4s^2 - 2 \geq n/8s^2$ empty blocks in layer $i-1$ that need to be consecutively pebbled during the time interval we are considering. By induction hypothesis, pebbling a block on the $i-1$-th layer requires time $n^{i-2}/2^{3i-5}s^{2i-6}$. Since the blocks are pebbled in disjoint time intervals we can add the costs, so the total time is $n^{i-1}/2^{3i-2}s^{2i-4}$. □

Observe that in the induction step, even if we try to use the fact that there are $s+1$ blocks in layer $i-1$ that need to be pebbled, we cannot simply sum the individual cost of each block: their pebbling times can overlap, so a pebble placed in layer $i-1$ or below and used for one block in layer i may also be used for another block in layer i.

Lemma C.4.21. *Let G be a k-layer bit-reversal graph. Let $\mathcal{P} \in \mathcal{P}_G^{bw}$ be a sequential black-white pebbling in space $\Pi_s(\mathcal{P}) = s \leq \sqrt{n}/4$. Then*

$$\Pi_t(\mathcal{P}) \geq n^k / 2^{3k-3} s^{2k-2} \quad .$$

Proof. For black-white pebbling we say that a block is pebbled during some time interval if it is empty at the beginning, there is a pebble at the end of the block at some point in the interval, and there are only black pebbles left at the end of the interval.

In contrast to the proof of Lemma C.4.20, we cannot assume that blocks need to be pebbled consecutively, but we can still show that in any time interval of length $n^{i-1}/2^{3i-4}s^{2i-4}$ at most s blocks are pebbled.

This is true for $i = 1$.

For $i > 1$, consider a set of s blocks and the first time interval in which a block is completely pebbled. Arguing analogously to the proof of Lemma C.4.20, at the beginning of the interval there are s vertices in layer $i-1$ whose closest pebble is at distance $n/2s$, so we can find $s(n/4s^2 - 2) \geq n/8s$ blocks in layer $i-1$ that are also empty at the beginning of the interval. By induction hypothesis, we can find $n/8s^2$ sets of blocks that are being pebbled in consecutive time intervals, and each such interval lasts for at least $n^{i-2}/2^{3i-7}s^{2i-6}$ steps. Now, it is possible that some of these moves could be reused to make progress on some other block in layer i than the one we are considering, but since the pebbling uses only s pebbles and we assumed that this is the first block to be pebbled, there are at most $s-1$ other blocks on layer i that may also finish being pebbled, those that already had a pebble on them. At the end of the time interval, we discard these blocks from the set, we look at the next time interval in which a block is completely pebbled, and repeat until we finish handling all blocks.

Observe that to pebble the graph we have to pebble $n/2s$ blocks on the k-th layer, and thus we have $n/2s^2$ time intervals for a total of $n^k/2^{3k-3}s^{2k-2}$ steps, as stated in the Lemma.

□

C.5 Reduction from Pebbling to Resolution

C.6 Concluding Remarks

In this paper, we study space complexity with a focus not on peak memory usage but on aggregated memory consumption over the whole computation. We consider two computational models, namely pebble games on DAGs and the resolution proof system in proof complexity.

For black-white pebbling, which is a model of nondeterministic computation, we prove optimal cumulative space lower bounds and also time-space trade-offs where in order to achieve optimal time the space needs to be large not only at a single point in time but throughout essentially the whole computation. We do so by studying the concepts of *depth-robustness* and *dispersion* of graphs, drawing on and extending work in [7, 8, 9] and other papers, and proving that different graph families of interest possess these properties.

In the context of proof complexity we are not aware of the cumulative space measure having been studied before, and so our first contribution here is to give a suitable formal definition, and also to consider different, more or less parallel, versions of the resolution proof system in which it makes sense to study cumulative space. We then use, and slightly extend, the reductions between pebbling and resolution in [27, 28] to transfer our lower bounds and trade-off results for pebbling also to resolution.

Since, to the best of our knowledge, ours is the first paper to study cumulative space both for black-white pebbling and for proof complexity, it is perhaps not so surprising that there is a wealth of open problems that this paper does not resolve. Below, we briefly discuss some possible directions for future research.

One set of questions on which we make progress but which we do not answer completely concern the relation between maximal space and cumulative space. For sequential black-white pebblings of n-vertex DAGs we prove an optimal $\Omega\big(n^2\big)$ cumulative space lower bound for a particular family of DAGs, but for graphs that can be pebbled in maximal space $O(\log n)$ we only obtain a $\Omega\big(n^2/\log n\big)$ cumulative space lower bound and for graphs pebblable in space $O(1)$ the best cumulative bound we can get is $\Omega\big(n^{3/2}\big)$. Could it be the case that there are graphs that can be pebbled in maximal space $O(1)$ but nevertheless require cumulative space $\Omega\big(n^2\big)$? Or do strong enough cumulative space lower bounds by necessity imply also nontrivial maximal space lower bounds?

We briefly mentioned a pebble game between sequential black-white and parallel black-white pebbling, the parallel-black sequential-white game, as an example of how to apply the depth-robustness lemma to other pebbling models. Does this pebble game have other interesting properties or applications?

It has been shown for parallel black pebbling that extremal depth-robustness is both necessary and sufficient for a graph to have high cumulative space complexity. We prove that for black-white pebbling predecessor-robustness is sufficient to imply high cumulative space, but leave open whether this condition is necessary or not.

For standard time-space trade-offs in sequential pebbling, it was shown in [130] that bit-reversal DAGs have a black pebbling trade-off of the form $t = \Theta\big(n^2/s\big)$ whereas for black-white pebbling the trade-off is a slightly weaker $t = \Theta\big(n^2/s^2\big)$. It was conjectured in [130] that there are other permutation graphs for which the black-white pebbling trade-off could also be shown to be an optimal $t = \Theta\big(n^2/s\big)$. One natural candidate class of graphs to consider in this context are graphs obtained from random permutations, and this is the original reason why we were interested to study them in this paper. So far we were only able to obtain trade-offs with the same parameters as for bit-reversal

DAGs, but it is an interesting question whether our tools could be sharpened to prove even stronger trade-offs results for random permutation graphs.

Turning to our proof complexity results, they can be seen to be yet another contribution to the sequence of papers [142, 147, 27, 144, 28] obtaining space bounds and time-space trade-offs in proof complexity by instead studying pebble games and reductions between pebblings of DAGs and resolution refutations of so-called pebbling formulas defined in terms of these DAGs. While these connections have turned out to be very fruitful, it would also be interesting to go beyond pebbling formulas and explore whether cumulative space results could be obtained for, e.g., Tseitin formulas on long and narrow rectangular grids as studied in [19, 23] or for other formulas.

One motivation behind our models of parallel resolution was the connection to parallel SAT solving, but our models do not take into account practical limitations such as the number of computing nodes or the communication between nodes. Could there be natural ways to incorporate such limitations, and could this also provide a better understanding of parallel resolution?

Another, somewhat related, question is whether formulas possessing strong cumulative space lower bounds are hard also in practice for (sequential or parallel) SAT solvers. Just maximal space lower bounds do not seem to be sufficient to imply practical hardness, as shown, e.g., in the fairly extensive empirical experiments on pebbling formulas in [116], but perhaps cumulative space could be a more relevant concept in this context.

Finally, it can be noted that our study of cumulative space in proof complexity as initiated in this paper is limited to the resolution proof system. This is mostly because resolution is the proof system where space complexity is best understood, and where the toolbox for studying these questions is most well developed. However, different concepts of maximal space and time-space trade-offs have been studied also for other proof systems such as polynomial calculus [2, 23, 39, 78, 79] and cutting planes [71, 87, 95, 106], and it would be interesting to extend the study of cumulative space to these proof systems.

Acknowledgements

The authors wish to thank Yuval Filmus for making us aware of each other's existence and thus providing the stimulus for the joint work that led up to this paper. Different subsets of the authors are grateful to Ilario Bonacina, with whom we had stimulating discussions during various stages of this project and who, in particular, provided valuable insights on dispersion, and to Adam Schill Collberg and Jan Elffers for helpful discussions on random permutation graphs.

The second, third and fourth authors were funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 279611. The third author was also supported by Swedish Research Council grants 621-2010-4797 and 621-2012-5645.

Paper D

Trade-offs Between Time and Memory in a Tighter Model of CDCL SAT Solvers

JAN ELFFERS, JAN JOHANNSEN, MASSIMO LAURIA, THOMAS MAGNARD, JAKOB NORDSTRÖM, AND MARC VINYALS

Full length version of the article published in Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16), July 2016, pp. 160–176

Abstract

A long line of research has studied the power of conflict-driven clause learning (CDCL) and how it compares to the resolution proof system in which it searches for proofs. It has been shown that CDCL can polynomially simulate resolution even with an adversarially chosen learning scheme as long as it is asserting. However, the simulation only works under the assumption that no learned clauses are ever forgotten, and the polynomial blow-up is significant. Moreover, the simulation requires very frequent restarts, whereas the power of CDCL with less frequent or entirely without restarts remains poorly understood. With a view towards obtaining results with tighter relations between CDCL and resolution, we introduce a more fine-grained model of CDCL that captures not only time but also memory usage and number of restarts. We show how previously established strong size-space trade-offs for resolution can be transformed into equally strong trade-offs between time and memory usage for CDCL, where the upper bounds hold for CDCL without any restarts using the standard 1UIP clause learning scheme, and the (in some cases tightly matching) lower bounds hold for arbitrarily frequent restarts and arbitrary clause learning schemes.

D.1 Introduction

For two decades the dominant strategy for solving the Boolean satisfiability problem (SAT) in practice has been *conflict-driven clause learning (CDCL)* [18, 136, 139]. Although SAT is an NP-complete problem, and is hence widely believed to be intractable in the worst case, CDCL SAT solvers have turned out to be immensely successful over a wide range of application areas. An important problem is to understand how such SAT solvers can be so efficient and what theoretical limits exist on their performance.

D.1.1 Previous Work

At the core, CDCL searches for proofs in the proof system *resolution* [35]. While pre- and inprocessing techniques can, and sometimes do, go significantly beyond resolution (incorporating, e.g., solving of linear equations mod 2 and reasoning with cardinality constraints), understanding the power of even just the fundamental CDCL search algorithm seems like an interesting and challenging problem in its own right. Three crucial aspects of CDCL solvers, which are the focus of our work, are running time, memory usage, and restart policy.

In resolution, time is modelled by the size/length complexity measure, in that lower bounds on proof size yield lower bounds on the running time of CDCL solvers. Resolution proof size is a well-studied measure. It is not hard to show that it need never be larger than exponential in the formula size, and such exponential lower bounds were shown already in, e.g., [98, 177, 61].

Another more recently studied measure is *(clause) space*, measured as the number of clauses one needs to keep in memory while verifying the correctness of a resolution proof.[1] We remark that although the study of space was originally motivated by SAT solving concerns, it is not a priori clear to what extent this abstract space measure corresponds to CDCL memory usage. Space need never be more than linear in the worst case [77], even though such proofs might have exponential size, and optimal linear lower bounds on space were obtained in [2, 25, 77].

More interesting than such space bounds is perhaps what can be said regarding simultaneous optimization of time and space, which is the setting in which SAT solvers operate. There are strong trade-offs [28, 19, 23] showing that this is not possible in general. What this means is that one can find formulas for which (a) there are short proofs and (b) also space-efficient proofs but (c) no proof can get close to being simultaneously both size- and space-efficient.

Regarding restarts, such a concept does not quite make sense for resolution proofs and so has not been studied in that context as far as we are aware.

[1]We mention for completeness that there is also a *total space* measure counting the number of literals in memory, which has been studied in, e.g., [2, 40, 32, 38], but for our purposes clause space seems like a more relevant measure to focus on.

It is natural to ask to what extent upper and lower bounds for resolution apply to CDCL. By comparison, it is well understood that the *DPLL* method [70, 69] searches for proofs in *tree-like resolution*, which incurs an exponential loss in performance as compared to general resolution. There has been a long line of research investigating how CDCL compares to general resolution, e.g., [21, 180, 50, 101], culminating in the result by [152] that CDCL viewed as a proof system polynomially simulates resolution with respect to size/time. The nonconstructive part of this result is that variable decisions are not done according to some concrete heuristic but are provided as helpful advice to the solver. This limitation is probably inherent, since a fully algorithmic result would have unexpected implications in complexity theory [4]. It is worth noting, however, that in independent work [13] showed that for resolution proofs where all clauses have constant size, using a *random* variable selection heuristic will yield a constructive polynomial-time simulation.

One strength of the results in [13, 152] is that they hold regardless of the specific learning scheme used, as long as it is *asserting* (an assumption that anyway lies at the heart of the CDCL algorithm). The results also have a few less desirable aspects, however:

- The simulations require very frequent restarts. Only the first conflict after each restart is useful, and after that one has to wait for the next restart to make any further progress.
- There is also a large polynomial blow-up in the simulations, which means that for practical purposes these simulations are far too inefficient to yield really concrete insights into CDCL performance as compared to resolution.
- Finally, and most seriously, the results crucially rely on the assumption that no learned clause is ever forgotten. This is unrealistic, as typically around 90–95% of learned clauses are erased during CDCL search and this is absolutely essential for performance.

It would be desirable to obtain results relating CDCL and resolution that also take the above aspects into account.

Addressing one of these concerns, a more fine-grained study of the power of CDCL without restarts has been conducted in, e.g., [50, 45, 51, 22]. One problematic aspect here is that the models studied appear to be quite far from actual CDCL behaviour. Some papers assume non-standard and rather artificial preprocessing steps. Others study CDCL models that do not enforce that unit clauses are propagated or that do not trigger conflict analysis as soon as a clause is falsified. In the latter case, as a result one gets very limited restrictions on what the clause learning schemes are, and it is hard even to talk about what "conflict analysis" is supposed to mean in this context. This is not an issue for results establishing lower bounds limiting what CDCL can do—here a stronger model of CDCL only makes the results stronger—but for upper bounds the results become too optimistic, indicating that the theoretical CDCL model can do much better than what seems possible in practice. As a case in point, there are currently no known separations between general resolution and CDCL without restarts, but part of the reason for this appears to be that the models of CDCL without restarts are clearly too strong to be realistic.

We are not aware of any work on models measuring not only time but also memory consumption in a proof system formalizing CDCL. As discussed above, one can define a space measure for resolution proofs, but it is not clear what relation, if any, there is between this space measure and the size of the clause database during CDCL execution.

D.1.2 Our Contributions

In this work, we present a proof system that tightly models running time, memory usage, and restarts in CDCL. The model draws heavily on [13, 152], combined with ideas from [50] to capture memory and restarts. Indeed, we do not claim any key new technical insights for this part of our work, but rather it is more a matter of carefully studying previous models and painstakingly putting the pieces together to get as clean and simple a proof system as possible that is nevertheless significantly “closer to the metal” than in previous papers.

Our CDCL proof system enforces unit propagation and triggers conflict analysis directly at a conflict. It can incorporate any asserting learning scheme (as long as it is based on resolution derivations from the current conflict and reason clauses), and this scheme is specified explicitly as a parameter. Right from the definitions one obtains natural measures of time, memory usage, and restarts. Variable decisions are still provided externally, just as in [13, 152], but in principle one could also plug in, say, the most commonly used *VSIDS (variable state independent decaying sum)* decision scheme with *phase saving* and analyse what proofs can be generated using these heuristics (though this is not the focus of our current work). Since we are now managing the database of learned clauses explicitly, we also have to specify a clause database reduction policy. In this paper, the decisions about which clauses to delete are also provided to the solver, but the model allows to plug in a concrete reduction policy as well.

We argue that the proof system we present faithfully models possible execution traces during CDCL search. Some interesting questions to study in this model are as follows:

1. Do upper and lower bounds on resolution size and space transfer to this CDCL proof system?

2. How does CDCL compare to general resolution if we want efficient simulations with respect to *both* time and space, and in addition aim for at most constant-factor blow-ups rather than arbitrary polynomial blow-ups?

3. What is the power of CDCL without restarts compared to the subsystems of tree-like resolution or so-called *regular resolution*? (Briefly, regularity is the somewhat SAT solver-like restriction on resolution that along each path in the proof any variable is branched over only once.)

The worst-case upper bounds on size and space in resolution carry over to time and memory usage in CDCL, and it turns out that this can in fact be read off from [140], although that paper uses quite a different language (and so we present a self-contained proof in this paper for completeness). More interestingly, we show that there is a straightforward translation from CDCL to resolution that preserves both time and space, and so we obtain that all size and space lower bounds previously established for resolution apply also to CDCL (which, in particular, was not at all obvious for space).

This means that the lower bounds on time-space trade-offs in [28, 19, 23] also hold for CDCL. But this does not yet yield true trade-offs, since for such results we also want *upper bounds*. That is, we want to show that CDCL can find time- or space-efficient proofs optimizing just one of these measures in isolation. It is known how to construct such proofs in resolution, but these proofs are not obviously CDCL-like. Since SAT solving was mentioned as a motivation for [28, 19, 23] it is a relevant question whether the size-space trade-offs shown in these papers correspond to anything one could expect to see in practice, or whether the size- and space-efficient proofs have such peculiar structure that nothing similar can be found by CDCL proof search.

The main contribution of our work is to address the question of whether true time-space trade-offs can be established for CDCL. Finding an answer turns out to be surprisingly technically challenging, and we are not able to prove the known trade-offs for exactly the same formulas as in [28, 19, 23] However, for many of the formulas it is possible to modify them slightly to obtain CDCL trade-offs with essentially the same parameters. An additional feature of these trade-offs is that all our upper bounds hold for CDCL *without any restarts* using the standard *1UIP (first unique implication point)* learning scheme, while the (often tightly matching) lower bounds hold for arbitrarily frequent restarts and arbitrarily chosen clause learning schemes (even non-asserting ones).

We leave as open problems whether CDCL with 1UIP clause learning and with or without restarts can simulate or be separated from general or regular resolution, respectively. While those problems still look quite challenging, we hope and believe that it should be possible to make progress by investigating them in a model that more closely resembles what happens during CDCL proof search in practice, such as the model presented in this paper.

D.1.3 Organization of This Paper

In Section D.2 we define our model of CDCL and relate it to the resolution proof system, and an overview of our results is then given in Section D.3. A more detailed discussion of CDCL worst-case bounds is presented in Section D.4. We establish our time-space trade-off results for CDCL in Sections D.5 and D.6. Finally, Section D.7 contains some concluding remarks.

D.2 Modelling CDCL as a Proof System

We start by describing our model of CDCL and how it is formalized as a proof system. As already mentioned, this is very much inspired by [13, 152], but with ideas added from [50]. We want to remark right away that we describe the model at a level of detail that might seem excessive to SAT practitioners familiar with CDCL. We do so precisely because a serious issue with many contributions on the theoretical side has been that they fail to get crucial details of the model right, as discussed in the introduction.[2]

D.2.1 Preliminaries

Let us first fix some standard notation and terminology. A *literal* a over a Boolean variable x is either x itself or its negation $\overline{x}$ (a *positive* or *negative* literal, respectively). A *clause* $C = a_1 \vee \cdots \vee a_k$ is a disjunction of literals, where the clause is *unit* if it contains only one literal. A *CNF formula* F is a conjunction of clauses $F = C_1 \wedge \cdots \wedge C_m$. We think of clauses and CNF formulas as sets, so that the order of elements is irrelevant and there are no repetitions.

A *resolution derivation* of C from F is a sequence of clauses $(C_1, C_2, \ldots, C_\tau)$ such that $C_\tau = C$ and every C_i is either a clause in F (an *axiom*) or is derived from clauses C_j, C_k with $j, k < i$, by the *resolution rule*

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}\ , \tag{D.2.1}$$

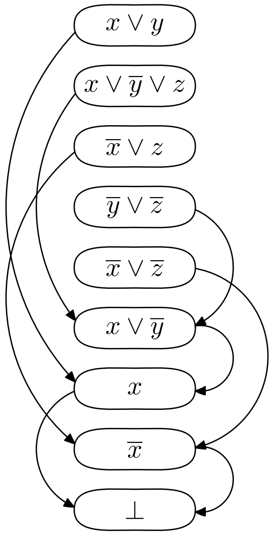
where we say that $C \vee x$ and $D \vee \overline{x}$ are *resolved* over x. A derivation is *trivial* if all variables resolved over are distinct and each C_i either is an axiom or is derived from a resolution rule application where one of the resolved clauses is an axiom. A *resolution refutation* of, or *resolution proof* for, an unsatisfiable formula F is a derivation of the empty clause $\perp$ (containing no literals) from the axioms in F. We can representent a refutation either as an annotated list of clauses as in Figure D.1a or as a directed acyclic graph (DAG) as in Figure D.1b. We say that a refutation is *tree-like* if this DAG is a tree (which is the case in Figure D.1b).

The *length* or *size* of a proof is the number of clauses in it counted with repetitions. The space of a proof at step t is the number of clauses at steps $\leq t$ that are used in applications of the resolution rule at steps $\geq t$. Looking at the example in Figure D.1, the space usage at step 7 is 5 (the clauses in memory at this point are clauses 1, 3, 5, 6, and 7). The space of a proof is obtained by measuring the space at each step and taking the maximum.

[2]Indeed there were issues with the model we presented in the conference version of this paper as well. Our description of the behaviour of the solver after a restart was not matching exactly what actual solvers seem to do in practice. Our trade-offs deal with CDCL proofs without restarts, therefore the correctness of the results was not compromised.

1.	$x \lor y$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$x \lor \overline{y}$	Res$(2, 4)$
7.	x	Res$(1, 6)$
8.	$\overline{x}$	Res$(3, 5)$
9.	$\perp$	Res$(7, 8)$

(a) Annotated list of clauses in refutation.


$x \lor y$
$x \lor \overline{y} \lor z$
$\overline{x} \lor z$
$\overline{y} \lor \overline{z}$
$\overline{x} \lor \overline{z}$
$x \lor \overline{y}$
x
$\overline{x}$
$\perp$

(b) DAG representation of refutation.

Figure D.1: Resolution refutation represented as list of clauses and directed acyclic graph.

D.2.2 A Formal Description of Conflict-Driven Clause Learning

A CDCL solver running on a formula F decides variable assignments and propagates values that follow from such assignments until a clause is falsified, at which point a learned clause is added to the clause database $\mathcal{D}$ (where we always have $F \subseteq \mathcal{D}$) and the search backtracks. A key concept is the current partial assignment maintained by the solver together with some book-keeping why variables were set this way, which we refer to as the *trail*. This is a sequence $s = (x_1 = b_1/*, x_2 = b_2/*, \ldots, x_\ell = b_\ell/*)$ where all variables are distinct and where $* = \mathsf{d}$ indicates that the assignment is a decision and $* = C$ that it was propagated by the clause C. We write $s_{\leq j}$ and $s_{<j}$ to denote the subsequences that are the prefixes of length j and $j-1$ of s, respectively. We denote the empty trail by ϵ.

The *decision level* of an assignment $x_j = b_j/*$ is the number of decision assignments in $s_{\leq j}$. The decision level of a (non-empty) trail is that of its last assignment. Identifying a trail s with the partial assignment it defines, we write $C\!\restriction_s$ to denote the clause C *restricted by* s, which is the trivially true clause if s satisfies C and otherwise C with all literals falsified by s removed, and this notation is extended to sets of clauses by taking unions. If a trail s falsifies a clause C, we say that C is *asserting* if it has a unique variable at the maximum decision level of s. If so, the second largest decision level represented in C is the *assertion level* of C.

A trail $s = (x_1 = b_1/*, \dots, x_\ell = b_\ell/*)$ is *legal* with respect to a formula F and clause database $\mathcal{D} \supseteq F$ if the following holds:

- $\mathcal{D}{\restriction}_{s_{<\ell}}$ does not contain the empty clause;
- if the jth element of s is $x_j = b_j/\mathsf{d}$, then $\mathcal{D}{\restriction}_{s_{<j}}$ does not contain a unit clause;
- if the jth element of s is $x_j = b_j/C$, then C is contained in $\mathcal{D}$ and has the property that $C{\restriction}_{s_{<j}}$ is unit and is satisfied by setting $x_j = b_j$.

This captures properties that must hold during CDCL search, and so in what follows trails are implicitly required to be legal unless otherwise specified.

At each point in time, the solver is in a *CDCL state* $(F, \mathcal{D}, s)$, where at the beginning $\mathcal{D} = F$ and $s = \epsilon$. It is convenient to describe the solver as being in one of the four modes **Default** (where it starts), **Unit**, **Conflict**, or **Decision**, where transitions are performed as described below (guided by plug-in components that specify the detailed behaviour; also to be discussed in what follows):

Default If s falsifies a clause in $\mathcal{D}$, the solver moves to **Conflict**, otherwise it checks that all variables in F have been assigned, and in that case the solver halts and outputs SAT together with the assignment s. Otherwise, if $\mathcal{D}{\restriction}_s$ contains a unit clause, the solver transits to **Unit** mode. If none of the previous cases applies, the solver uses its *restart policy* to decide whether to restart, i.e., to set $s = \epsilon$ and to move to **Default**. At last, if none of the others cases applies, solver uses its *clause database reduction policy* to decide whether to shrink $\mathcal{D}$ to $\mathcal{D}' \subsetneq \mathcal{D}$, where $\mathcal{D}'$ must still contain F and all clauses mentioned in the current trail s, after which it moves to **Decision**.

Conflict If s has decision level 0, the solver outputs UNSAT. Otherwise it applies the *learning scheme* to derive an asserting clause C and then *backjumps* by updating the state to $(F, \mathcal{D} \cup \{C\}, s')$ (where s' is the prefix of s that contains all assignments with decision level less than or equal to the assertion level of C), and shifts to **Unit** mode.

Unit The solver uses the *unit propagation scheme* to pick a clause C in $\mathcal{D}$ such that $C{\restriction}_s$ is unit, extends s with the assignment $x = b/C$ that satisfies $C{\restriction}_s$, and moves to **Default** mode.

Decision The solver uses the *decision scheme* to determine an assignment $x = b/\mathsf{d}$ with which to extend the trail and moves to **Default** mode.

We say that a CDCL state $(F, \mathcal{D}, s)$ is *stable* if, when solver is in **Default** mode, it causes neither a conflict, a unit propagation nor to output SAT. We say it is a *conflict state* if it causes a move from **Default** to **Conflict**. We remark that CDCL solvers typically apply restarts and database reductions only in the first stable state after a conflict. However, it is not hard to see that from a proof complexity point of view the solver does not get any stronger by allowing these steps to be performed at any stable state, and since this simplifies the description we have done so above.

In order to obtain a concrete CDCL implementation, one needs to instantiate the components referred to above. Let us briefly discuss how this can be done.

For the *clause learning scheme* the assumption is that the clause is derivable in resolution from the clause falsified (the *conflict clause*) and the clauses causing unit propagations (the *reason clauses*) and that the learned clause is always asserting. For our upper bounds we use the 1UIP learning scheme from [187], which is simply a trivial resolution derivation from the conflict clause and the reason clauses processed in reverse order up to the first point when there remains only one variable of maximal decision level in the clause.[3]

The *restart policy* determines when the solver should clear the trail and start over from the beginning (but keeping the clause database as it is). From a theoretical point of view adding more frequent restarts can only make the solver more powerful. Hence, in order to obtain the strongest possible result we want to prove our upper bounds on CDCL with a strict no-restarts policy and our lower bounds in a setting with no restrictions on restarts.

If there is more than one unit clause that can propagate in **Unit** mode, the *unit propagation scheme* determines in which order the clauses are chosen. Typically this will depend somewhat randomly on low-level implementation details, and therefore we try to prove our upper and lower bounds for the settings when the order chosen is maximally unhelpful and maximally helpful, respectively.

The *decision scheme* is used to choose the next variable to assign when there are no unit propagations. The dominant heuristic in practice is VSIDS [139], but for our theoretical analysis we follow [13, 152] by allowing the decisions to be chosen externally by a helpful oracle and fed to the solver.

The *database reduction policy*, finally, regulates when and how to forget learned clauses. Making this aspect explicit is the main difference between our work and [13, 152]—the latter papers crucially need the unrealistic assumption that no learned clauses may ever be erased. In principle, here one could plug in, say, the *literal block distance (LBD)* heuristic in [16] to decide which clauses to throw away or keep, but in this work we will let this, too, be part of the external input provided to construct a CDCL proof.

D.2.3 Formalizing CDCL as a Proof System

In order to construct a proof system corresponding to CDCL, we will simply let the proofs be execution traces that contain enough information to allow efficient verification that they are consistent with the detailed description of the CDCL model above. More

[3]In fact, our results hold for any UIP scheme, but for simplicity we focus on 1UIP, which is anyway dominant in practice.

formally, we say that a *CDCL trace* π is an ordered sequence of the following types of elements:

- decisions $x_i = b/\mathsf{d}$;
- unit propagations $x_i = b/C$ (with reason C);
- learned clauses $\mathsf{add}\, C/\sigma_C$ (with conflict analysis σ_C);
- deletions of clauses $\mathsf{del}\, C$;
- restarts R.

Given a CDCL model with components as above partially or fully specified, a trace π is *legal*, or is a *CDCL proof*, if it is consistent with an execution of the CDCL model as described in Section D.2.2.

To verify a trace we start the CDCL solver in state (F, F, ϵ) in **Default** mode and then traverse the sequence π, updating the state as we go along. If the next element is a decision $x_i = b/\mathsf{d}$, then the solver should be in **Decision** mode and the assignment should be consistent with the decision scheme, or should at least lead to a legal trail if the decision scheme is not specified. For a unit propagation $x_i = b/C$ the solver should be in **Unit** mode and $x_i = b$ should be propagated by C under the current trail, and should be the propagation chosen by the propagation scheme if specified. For $\mathsf{add}\, C/\sigma_C$ the solver should be in **Conflict** mode and C should be an asserting clause obtained by the resolution derivation σ_C from the conflict and reason clauses in the current trail, where σ_C complies with the requirements of the learning scheme if specified. Deletions of clauses $\mathsf{del}\, C$ and restarts R should only happen in **Default** mode and should be consistent with their respective policies if provided (where, at a minimum, no clause on the trail may be forgotten).

We say that a CDCL trace is a *CDCL proof of unsatisfiablity* or *CDCL refutation* of F if it is legal and makes the CDCL solver output UNSAT, and that it is a *CDCL proof of satisfiablity* if the output is SAT. It should be clear that if the components specified are efficiently computable, then CDCL traces are efficiently verifiable and constitute a proof system in the sense of [66] (and since all traces we construct will be legal, we will sometimes use the words "trace" and "proof" as synonyms).

The *time* of a CDCL proof π is the number of elements in the sequence plus the sum of the length of all conflict analysis resolution derivations σ_C, i.e., the total number of variable decisions, propagations, and steps in conflict analysis. The *space* of the proof at a given point in time is the number of learned clauses $|\mathcal{D} \setminus F|$, i.e., the number of statements $\mathsf{add}\, C/\sigma_C$ minus the number of statements $\mathsf{del}\, C$ up to that point, and the space of a proof is obtained by taking the maximum over all time steps in it.

These measures are intended to capture the execution time and memory usage of a CDCL solver execution described by the trace π, and in addition we want them to translate to length and space bounds for resolution. This is indeed the case, but in order to make a precise, formal statement we first need to discuss clause learning schemes.

D.2.4 Conflict Graph and Learning Schemes

To define the learning schemes that we will discuss in the following sections we need to introduce some concepts. The interested reader can find an extended discussion about conflict analysis in [21].

The *conflict graph* of a conflict state is defined iteratively as follows. We begin with two vertices labelled by the conflict literal and its negation. For each literal a in the graph that is unit propagated with reason C, we add an edge from each literal in $\overline{C \setminus \{a\}}$ (and possibly a new vertex) to a. For this construction we assume that the conflicting clause propagates the opposite of the conflict literal.

For each cut in this directed graph that separates all decision literals from the two conflict literals, we can collect the vertices from which the cut edges emanate and negate all these literals to obtain a clause that is falsified by the current assignment and is therefore a conflict clause that can be learned. The *reason side* contains the decisions and the *conflict side* contains the conflicts.

A *unique implication point* (UIP) is a vertex that belongs to all paths from the last decision literal to the two conflict literals. The clause induced by the cut whose conflict side are the successors of a UIP is asserting. The *1UIP* learning scheme learns the clause induced by the UIP closest to the conflict literals. It is well-defined because there is always at least one UIP: the last decision literal.

We will refer to a learning scheme that learns asserting clauses that come from a cut in the conflict graph as a *cutting* learning scheme. 1UIP is clearly a cutting scheme. The *decision* learning scheme, which learns the opposite of all decision literals in the current assignment, is asserting but not necessarily cutting.

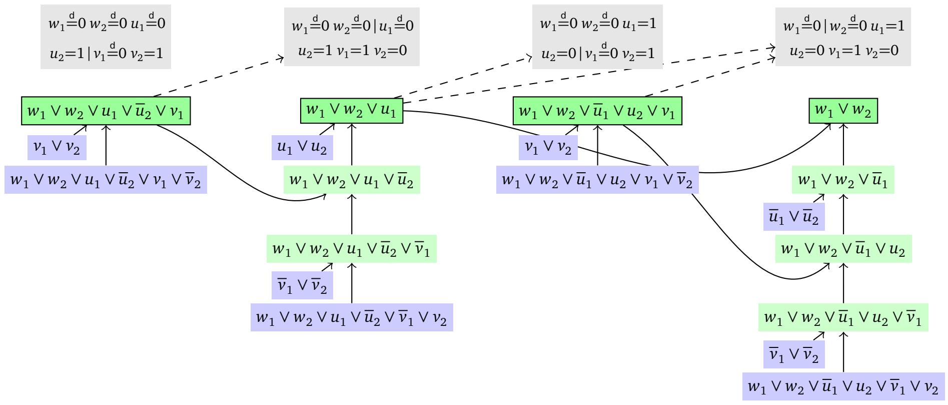
We say that a learning scheme is *trivial* if it produces clauses that can be derived from the clause database using trivial resolution. All cutting learning schemes are trivial [21].

D.2.5 From CDCL Proofs to Resolution Proofs

We now show that a CDCL refutation with a learning scheme that uses trivial resolution to derive conflict clauses can be translated to a resolution refutation in essentially the same time and space.

Theorem D.2.1. *If there is a CDCL proof with some trivial learning scheme refuting a CNF formula F in time τ and space s, then F has a resolution refutation of length at most τ and space at most $s + 3$.*

Proof. Given a legal CDCL trace refuting F, we construct a resolution proof in the following way. We list all the trivial resolution derivations of learned clauses in order, but we omit occurrences of learned clauses, so that we use the last occurrence of a learned clause at the end of a derivation as an input clause for the current derivation (see Figure D.2). At the end we add a derivation of the empty clause from the last conflict.



$w_1 \overset{d}{=} 0\ w_2 \overset{d}{=} 0\ u_1 \overset{d}{=} 0$
$u_2 = 1 \mid v_1 \overset{d}{=} 0\ v_2 = 1$
$w_1 \overset{d}{=} 0\ w_2 \overset{d}{=} 0 \mid u_1 \overset{d}{=} 0$
$u_2 = 1\ v_1 = 1\ v_2 = 0$
$w_1 \overset{d}{=} 0\ w_2 \overset{d}{=} 0\ u_1 = 1$
$u_2 = 0 \mid v_1 \overset{d}{=} 0\ v_2 = 1$
$w_1 \overset{d}{=} 0 \mid w_2 \overset{d}{=} 0\ u_1 = 1$
$u_2 = 0\ v_1 = 1\ v_2 = 0$
$w_1 \lor w_2 \lor u_1 \lor \overline{u}_2 \lor v_1$
$v_1 \lor v_2$
$w_1 \lor w_2 \lor u_1 \lor \overline{u}_2 \lor v_1 \lor \overline{v}_2$
$w_1 \lor w_2 \lor u_1$
$u_1 \lor u_2$
$w_1 \lor w_2 \lor u_1 \lor \overline{u}_2$
$w_1 \lor w_2 \lor u_1 \lor \overline{u}_2 \lor \overline{v}_1$
$\overline{v}_1 \lor \overline{v}_2$
$w_1 \lor w_2 \lor u_1 \lor \overline{u}_2 \lor \overline{v}_1 \lor v_2$
$w_1 \lor w_2 \lor \overline{u}_1 \lor u_2 \lor v_1$
$v_1 \lor v_2$
$w_1 \lor w_2 \lor \overline{u}_1 \lor u_2 \lor v_1 \lor \overline{v}_2$
$w_1 \lor w_2$
$w_1 \lor w_2 \lor \overline{u}_1$
$\overline{u}_1 \lor \overline{u}_2$
$w_1 \lor w_2 \lor \overline{u}_1 \lor u_2$
$w_1 \lor w_2 \lor \overline{u}_1 \lor u_2 \lor \overline{v}_1$
$\overline{v}_1 \lor \overline{v}_2$
$w_1 \lor w_2 \lor \overline{u}_1 \lor u_2 \lor \overline{v}_1 \lor v_2$

Figure D.2: Fragment of a CDCL proof translated into resolution. Dark blue clauses are axioms, dark green clauses with a border are learned and added to the clause database, and light green clauses are auxiliary. Upper boxes contain the trail at each conflict state. Edges show resolution steps, and dashed edges are reasons for unit propagation.

We can prove that it is a legal refutation by forward induction over the learned clauses in the derivation. Each partial derivation uses clauses in the database, all legally derived by induction hypothesis.

The length of the resolution refutation is the sum of lengths of each derivation, which we account for in the time of CDCL refutation, plus the length of the derivation of the empty clause. Since every derivation uses reason clauses, the length of the last derivation is at most the number of unit propagations.

To estimate the space of the refutation, we consider the partial derivation of the clause learned in a conflict state $(F, \mathcal{D}, s)$. At each resolution step, one of the clauses was in the clause database, and the other is either in the database or it was derived in the previous step and will not be used again. The same applies for later derivations. Therefore, for each resolution step t, all the clauses derived before step t that are used strictly later are in the clause database. Furthermore, axioms are always restated immediately before a derivation and we should not count them, so the upper bound is actually $|\mathcal{D} \setminus F|$. It only remains to count the clause derived at step t, and the two clauses used to derive it. Thus, the space at step t is at most $|\mathcal{D} \setminus F| + 3$. Taking the maximum over all steps gives precisely $s + 3$. ☐

D.3 Overview of Time-Space Trade-off Results

In this section we survey the kind of CDCL time-space trade-offs obtained in this paper, and discuss some of the challenges that have to be overcome when establishing such results.

D.3.1 Statement of Trade-off Theorems

Our first set of trade-off results are for formulas defined in terms of pebble games as described in [29]. Given a directed acyclic graph (DAG) G with source vertices S and a unique sink vertex z, and with all non-sources having fan-in 2, we let every vertex in G correspond to a variable and define the *pebbling formula* over G, denoted Peb_G, to consist of the following clauses:

- for all $s \in S$, the unit clause s (*source axioms*),
- for all non-sources w with immediate predecessors u, v, the clause $\overline{u} \vee \overline{v} \vee w$ (*pebbling axioms*),
- for the sink z, the unit clause $\overline{z}$ (*sink axiom*).

See the formula in Figure D.3b defined in terms of the pyramid graph in Figure D.3a for an illustration.

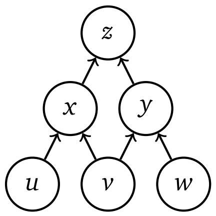
These formulas are not too interesting, since it is easy to see that they are solved immediately by unit propagation, but if we replace each variable by an exclusive or of two new, fresh variables, and then expand out to CNF we obtain a *XORified pebbling formula* $Peb_G^{\oplus}$ as in Figure D.3c. Given the right kind of graphs, [28] showed that such formulas have strong trade-offs between length and space in resolution, and we are able to lift most of these results to CDCL. We give two examples of such results below.

Theorem D.3.1 (Robust trade-offs (informal)). *There are XORified pebbling formulas F_n of size $\Theta(n)$ such that:*

- *CDCL with 1UIP learning and no restarts can refute F_n in time $O(n)$ and space $O(n/\log n)$ simultaneously.*
- *CDCL with 1UIP learning and no restarts can refute F_n in space $O\big((\log n)^2\big)$ and time $n^{O(\log n)}$ simultaneously.*
- *Any CDCL refutation of F_n in space $o(n/\log n)$ requires time at least $n^{\Omega(\log\log n)}$ regardless of learning scheme and restart policy.*

Theorem D.3.2 (Exponential trade-offs (informal)). *There are XORified pebbling formulas F_n of size $\Theta(n)$ such that:*

- *CDCL with 1UIP learning and no restarts can refute F_n in time $O(n)$ and space $O\big(\sqrt[4]{n}\big)$ simultaneously.*
- *CDCL with 1UIP learning and no restarts can refute F_n in space $O\big(\sqrt[8]{n}\big)$ and time $n^{O(\sqrt[8]{n})}$ simultaneously.*
- *Any CDCL refutation of F_n in space $O\big(n^{1/4-\epsilon}\big)$ for $\epsilon > 0$ requires exponential time regardless of learning scheme and restart policy.*



(a) Pyramid graph Π_2 of height 2.

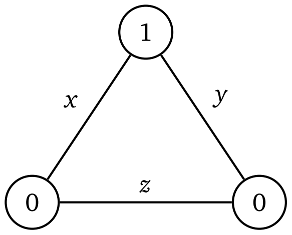
$$\begin{aligned}
& u \\
\land\ & v \\
\land\ & w \\
\land\ & (\overline{u} \lor \overline{v} \lor x) \\
\land\ & (\overline{v} \lor \overline{w} \lor y) \\
\land\ & (\overline{x} \lor \overline{y} \lor z) \\
\land\ & \overline{z}
\end{aligned}$$

(b) Pebbling contradiction Peb_{Π_2}.

$$\begin{aligned}
& (u_1 \lor u_2) \\
\land\ & (\overline{u}_1 \lor \overline{u}_2) \\
\land\ & (v_1 \lor v_2) \\
\land\ & (\overline{v}_1 \lor \overline{v}_2) \\
\land\ & (w_1 \lor w_2) \\
\land\ & (\overline{w}_1 \lor \overline{w}_2) \\
\land\ & (u_1 \lor \overline{u}_2 \lor v_1 \lor \overline{v}_2 \lor x_1 \lor x_2) \\
\land\ & (u_1 \lor \overline{u}_2 \lor v_1 \lor \overline{v}_2 \lor \overline{x}_1 \lor \overline{x}_2) \\
\land\ & (u_1 \lor \overline{u}_2 \lor \overline{v}_1 \lor v_2 \lor x_1 \lor x_2) \\
\land\ & (u_1 \lor \overline{u}_2 \lor \overline{v}_1 \lor v_2 \lor \overline{x}_1 \lor \overline{x}_2) \\
\land\ & (\overline{u}_1 \lor u_2 \lor v_1 \lor \overline{v}_2 \lor x_1 \lor x_2) \\
\land\ & (\overline{u}_1 \lor u_2 \lor v_1 \lor \overline{v}_2 \lor \overline{x}_1 \lor \overline{x}_2) \\
\land\ & (\overline{u}_1 \lor u_2 \lor \overline{v}_1 \lor v_2 \lor x_1 \lor x_2) \\
\land\ & (\overline{u}_1 \lor u_2 \lor \overline{v}_1 \lor v_2 \lor \overline{x}_1 \lor \overline{x}_2) \\
\land\ & (v_1 \lor \overline{v}_2 \lor w_1 \lor \overline{w}_2 \lor y_1 \lor y_2) \\
\land\ & (v_1 \lor \overline{v}_2 \lor w_1 \lor \overline{w}_2 \lor \overline{y}_1 \lor \overline{y}_2) \\
\land\ & (v_1 \lor \overline{v}_2 \lor \overline{w}_1 \lor w_2 \lor y_1 \lor y_2) \\
\land\ & (v_1 \lor \overline{v}_2 \lor \overline{w}_1 \lor w_2 \lor \overline{y}_1 \lor \overline{y}_2) \\
\land\ & (\overline{v}_1 \lor v_2 \lor w_1 \lor \overline{w}_2 \lor y_1 \lor y_2) \\
\land\ & (\overline{v}_1 \lor v_2 \lor w_1 \lor \overline{w}_2 \lor \overline{y}_1 \lor \overline{y}_2) \\
\land\ & (\overline{v}_1 \lor v_2 \lor \overline{w}_1 \lor w_2 \lor y_1 \lor y_2) \\
\land\ & (\overline{v}_1 \lor v_2 \lor \overline{w}_1 \lor w_2 \lor \overline{y}_1 \lor \overline{y}_2) \\
\land\ & (x_1 \lor \overline{x}_2 \lor y_1 \lor \overline{y}_2 \lor z_1 \lor z_2) \\
\land\ & (x_1 \lor \overline{x}_2 \lor y_1 \lor \overline{y}_2 \lor \overline{z}_1 \lor \overline{z}_2) \\
\land\ & (x_1 \lor \overline{x}_2 \lor \overline{y}_1 \lor y_2 \lor z_1 \lor z_2) \\
\land\ & (x_1 \lor \overline{x}_2 \lor \overline{y}_1 \lor y_2 \lor \overline{z}_1 \lor \overline{z}_2) \\
\land\ & (\overline{x}_1 \lor x_2 \lor y_1 \lor \overline{y}_2 \lor z_1 \lor z_2) \\
\land\ & (\overline{x}_1 \lor x_2 \lor y_1 \lor \overline{y}_2 \lor \overline{z}_1 \lor \overline{z}_2) \\
\land\ & (\overline{x}_1 \lor x_2 \lor \overline{y}_1 \lor y_2 \lor z_1 \lor z_2) \\
\land\ & (\overline{x}_1 \lor x_2 \lor \overline{y}_1 \lor y_2 \lor \overline{z}_1 \lor \overline{z}_2) \\
\land\ & (z_1 \lor \overline{z}_2) \\
\land\ & (\overline{z}_1 \lor z_2)
\end{aligned}$$

(c) XORified pebbling formula $Peb^{\oplus}_{\Pi_2}$.

Figure D.3: Example pebbling formula for the pyramid of height 2.



(a) Labelled triangle graph.

$$
\begin{aligned}
&(x \vee y) \\
\wedge\ &(\overline{x} \vee \overline{y}) \\
\wedge\ &(x \vee \overline{z}) \\
\wedge\ &(\overline{x} \vee z) \\
\wedge\ &(y \vee \overline{z}) \\
\wedge\ &(\overline{y} \vee z)
\end{aligned}
$$

(b) Corresponding Tseitin formula.

Figure D.4: Example Tseitin formula.

The other formula family considered in this paper are *Tseitin formulas*, which are defined in terms of undirected graphs with vertices labelled 0/1 in such a way that the total sum of all vertex labels is odd. The variables of the formula are the edges of the graph. For every vertex we add a constraint saying that the parity of the number of true edges incident to the vertex is equal to the vertex label. Summing over all vertices, each edge is counted exactly twice and hence the total number of true edges must be even. But this contradicts that the sum of the labels is odd, and thus the formulas are unsatisfiable. Figure D.4b gives an example Tseitin formula generated from the labelled graph in Figure D.4a.

Using Tseitin formulas over long, skinny grids, we can build on [23] to obtain the following trade-off, which applies even for superlinear space.

Theorem D.3.3 (Superlinear space trade-offs (informal)). *For a Tseitin formula $F_{w,\ell}$ over a grid graph with w rows and ℓ columns, $1 \leq w \leq \ell^{1/4}$, and with double edges between every two vertices at horizontal distance one or vertical distance one, it holds that*

- *CDCL with 1UIP learning and no restarts can refute $F_{w,\ell}$ in time $O(2^{5w}\ell)$ and space $O(2^{2w})$.*
- *CDCL with 1UIP learning and no restarts can refute $F_{w,\ell}$ in space $O(w\log(\ell))$ and time $O\big(\ell^{O(w)}\big)$.*
- *For any CDCL refutation in time τ and space s, regardless of learning scheme and restart policy, it holds that*

$$\tau = \left(\frac{2^{\Omega(w)}}{s}\right)^{\Omega\left(\frac{\log\log\ell}{\log\log\log\ell}\right)} .$$

D.3.2 Proof Techniques and Technical Challenges

All the trade-offs stated in Theorems D.3.1, D.3.2, and D.3.3 are known to hold for resolution, and so by Theorem D.2.1 we immediately obtain that the lower bounds carry over to CDCL. What we need to show in order to establish these theorems is that CDCL can find proofs that match the upper bounds in resolution.

The general idea how we would like to do this is clear: given a resolution proof $\pi = (C_1, C_2, \ldots, C_\tau)$, we should force the CDCL solver to efficiently learn the clauses C_i one by one, making sure at all times that the clause database size is comparable to the space complexity of the resolution proof. This seems hard to do, however, and somewhat ironically what causes trouble for us are the unit propagations that otherwise make CDCL so efficient. To illustrate the problem, suppose that we have learned $C \vee x$ and $D \vee \overline{x}$ and now want to learn their resolvent $C \vee D$. It would be nice to decide on all literals in $C \vee D$ being false, after which we could get a conflict on x. But there might be other clauses in the database that propagate literals to "wrong values" before we manage to falsify all of $C \vee D$, and if so the CDCL search will veer off in another direction and we will not be able to learn this resolvent.

This highlights two technical difficulties that we need to be able to deal with:

- Not only do we have to decide on variables in the right order, but we have to make sure that no other unexpected (and unwanted) propagations occur.
- In contrast to resolution, where having more clauses at your disposal never hurts, keeping too many learned clauses in the clause database can actually hinder the CDCL search. This is also a striking contrast to [13, 152], where a key technical lemma is precisely that having more clauses in the database can only be helpful.

We do not know how to simulate general resolution efficiently with respect to length and space simultaneously, even using ever so frequent restarts. And an additional problem is that we want to know—in order to better understand basic CDCL reasoning with unit propagation and conflict analysis—whether for the pebbling and Tseitin formulas presented above CDCL can find efficient proofs *even without restarts*. This makes our task substantially more complicated.

If we allow suitably frequent restarts, however, it is not too hard to show that CDCL can efficiently simulate the "canonical" resolution proofs for these formulas. To give at least some flavour of the technical arguments that will follow later in the paper when we reason about the CDCL proof system, we conclude this section with a description of how this result can be proven for pebbling formulas.

D.3.3 Pebbling Formula Upper Bound for CDCL with Restarts

A pebbling formula encodes the *black pebble game* played on a DAG G, where we start with G being empty and want to finish with a pebble on the sink z. A vertex can be pebbled if its predecessors have pebbles (vacuously true for sources), and pebbles can always be removed. The *time* of a pebbling is the number of moves before z is reached and the *space* is the maximum number of pebbles on vertices of G at any point.

Resolution can simulate such pebblings by deriving, whenever a vertex w is pebbled, the two *pebble clauses* $w_1 \vee w_2$ and $\overline{w}_1 \vee \overline{w}_2$ saying that the exclusive or $w_1 \oplus w_2$ is true,

```
  Input: a black pebbling $\mathcal{P}$
1 foreach (move,w) in $\mathcal{P}$ where w is not a source or the sink do
2     if move is Add then
3         HalfPebble ($w$, 0)
4         HalfPebble ($w$, 1)
5     else
6         del $w_1 \lor w_2$
7         del $\overline{w}_1 \lor \overline{w}_2$
8 PebbleSink
```

Figure D.5: Procedure Pebble ($\mathcal{P}$)

and by erasing these clauses whenever a pebble is removed. For a source vertex the pebble clauses are already available as source axioms in the formula (see Figure D.3c), and it is not hard to show that resolution can efficiently propagate exclusive ors from predecessors to successors. Once pebble clauses have been derived for the sink z, contradiction immediately follows from the sink axioms.

We want to mimic this in CDCL as described in the algorithm Pebble in Figure D.5 producing a CDCL trace. For a pebble placement, we want to learn first $w_1 \lor w_2$, corresponding to “half of the pebble” on w, and then $\overline{w}_1 \lor \overline{w}_2$. How to do this is described in the procedure HalfPebble in Figure D.6, where the notation x^b for $b \in \{0, 1\}$ is used as a compact way of denoting $x^1 = x$ and $x^0 = \overline{x}$.

When a pebble is placed on w in the pebbling, we let the CDCL solver make the decisions ($w_1 = 0/\mathsf{d}, w_2 = 0/\mathsf{d}$) with the goal of learning $w_1 \lor w_2$. Then we decide values for the variables of the predecessors u and v of w, and since there are clauses in memory encoding $u_1 \oplus u_2$ and $v_1 \oplus v_2$ this will provoke repeated conflicts until finally the clause $w_1 \lor w_2$ is learned. Since this only involves a constant number of variables, the time and space required for this is constant, and our goal can be achieved, e.g., as described in FindConflicts in Figure D.7.

But now we run into problems. At this point the CDCL solver will backjump to the decision $w_1 = 0$, where the learned clause $w_1 \lor w_2$ asserts $w_2 = 1$. As the next step, we want to generate conflicts that lead to the “second half” of the pebble $\overline{w}_1 \lor \overline{w}_2$ being learned, but there is no way this can happen since the decision $w_1 = 0$ is on the trail and the clause $\overline{w}_1 \lor \overline{w}_2$ is thus satisfied. Moreover, if the solver is not allowed to restart, then this satisfying assignment is fixed on the trail, and no new conflict could possibly cause a backjump to before this assignment. Therefore, the solver is forced to continue the proof search elsewhere. This turns out to be a major obstacle, which we are able to circumvent only by substantial extra work involving reordering the pebbling and using a different algorithm. This will be a large part of the work in the rest of this paper as the technical arguments are rather intricate.

```
   Input: A vertex $w$, a Boolean $b$
1  Decide $w_1 = b/\mathsf{d}$
2  Decide $w_2 = b/\mathsf{d}$
3  FindConflicts ($w$, $b$)
4  Learn $w_1^{1-b} \lor w_2^{1-b}$ and assert $w_2 = 1 - b/\mathsf{u}$
5  Restart R
6  foreach clause $C \in \mathcal{D} \setminus F$ such that $|C| > 2$ do
7      del $C$
```

Figure D.6: Procedure HalfPebble (w, b)

```
    Input: A vertex $w$ with predecessors $u$ and $v$, a Boolean $b$
1   Decide $u_1 = 0/\mathsf{d}$
2   Propagate $u_2 = 1/u_1 \lor u_2$
3   Decide $v_1 = 0/\mathsf{d}$
4   Learn $w_1^{1-b} \lor w_2^{1-b} \lor u_1 \lor \overline{u}_2 \lor v_1$
5   Assert $v_1 = 1/w_1^{1-b} \lor w_2^{1-b} \lor u_1 \lor \overline{u}_2 \lor v_1$
6   Learn $w_1^{1-b} \lor w_2^{1-b} \lor u_1$
7   Assert $u_1 = 1/w_1^{1-b} \lor w_2^{1-b} \lor u_1$
8   Propagate $u_2 = 0/\overline{u}_1 \lor \overline{u}_2$
9   Decide $v_1 = 0/\mathsf{d}$
10  Learn $w_1^{1-b} \lor w_2^{1-b} \lor \overline{u}_1 \lor u_2 \lor v_1$
11  Assert $v_1 = 1/w_1^{1-b} \lor w_2^{1-b} \lor \overline{u}_1 \lor u_2 \lor v_1$
```

Figure D.7: Procedure FindConflicts (w, b)

If we instead give the solver the option to restart at this point, it can clear the trail and also forget all unnecessary clauses. This means that the decisions $(w_1 = 1/\mathsf{d}, w_2 = 1/\mathsf{d})$ can be made, after which the clause $\overline{w}_1 \lor \overline{w}_2$ is learned in the same way as above. To conclude, we again trigger a restart and erase all auxiliary clauses that are no longer needed.

Pebble removals are very straightforward to simulate: the only condition that could stop us from erasing the clauses $w_1 \lor w_2$ and $\overline{w}_1 \lor \overline{w}_2$ is if they are reasons for propagated literals on the trail, but since we have just made a restart the trail is empty. Formalizing the arguments above, we obtain the following lemma.

Lemma D.3.4. *If $\mathcal{P}$ is a black pebbling of G in space s and time τ, then there is a CDCL proof of $Peb_G^{\oplus}$ with restarts using the 1UIP learning scheme and any unit propagation scheme in space at most $2s + 3$ and time $\mathrm{O}(\tau)$.*

Proof. Given a pebbling $\mathcal{P}$ we generate a trace as described by the procedure `Pebble`. Note that this procedure maintains the invariant that the pebble clauses for a non-source vertex w are in the clause database if and only if there is a pebble on w. No other clauses are in memory. The space bound follows from this invariant, and the time bound holds by construction.

It remains to check that the trace thus generated is legal. Observe that the clauses in memory only propagate if at least one variable from each vertex they mention is set. Since the decision sequence mentions at most three vertices at the same time, we only need to reason about clauses that mention these vertices.

The correctness of `FindConflicts` is straightforward to verify, since the order of unit propagations can be seen to be uniquely determined. At the end of `FindConflicts`, the assignments to w_1 and w_2 are decisions and all predecessor variables u_1, u_2, v_1, v_2 are set by unit propagation. Since one of the conflicting clauses, a pebbling axiom, contains the decision variables w_1 and w_2, they have to appear in any cut and therefore any conflict clause. The remaining variables in the conflict graph have maximal decision level, so they cannot appear in an asserting clause because w_2 already appears. Therefore, we learn the clause $w_1^{1-b} \lor w_2^{1-b}$ and assert $w_2 = 1 - b/\mathtt{u}$. We only erase clauses after a restart, so no erased clauses can be reasons for unit propagations. This concludes the proof.

□

Figure D.8 is the result of running `Pebble` on a specific pebbling strategy on the example formula from Figure D.3c, with resolution steps omitted.

D.4 Worst-case Upper Bound

The simulation by [152] shows that CDCL with unrestricted restarts can polynomially simulate resolution. The authors only analyze time and assume that no learnt clause is ever forgotten. However, it is not hard to come up with a clause deletion strategy that gives an $O(s \cdot \mathit{poly}(n))$ space bound, where s is the space of the resolution refutation we simulate and n is the number of variables. Because the proof length is always at least linear in n, the $O(\mathit{poly}(n))$ blowup is polynomial with respect to length. Space, however, can be $O(\log n)$ or even $O(1)$, so that the blowup is exponential. Therefore, the simulation does not resolve the question of space-efficient simulation of resolution with restarts.

To start acquainting ourselves with the CDCL proof system, let us demonstrate that the worst-case behaviour for CDCL is the same as for resolution: any unsatisfiable CNF formula can be refuted in exponential time and linear space simultaneously. We remark that this result was already shown in [140]. Since the language used there is quite different, however, we present a self-contained proof below for completeness.

1 $x_1=0/\mathrm{d}$
2 $x_2=0/\mathrm{d}$
3 $u_1=0/\mathrm{d}$
4 $u_2=1/u_1 \lor u_2$
5 $v_1=0/\mathrm{d}$
6 add $x_1 \lor x_2 \lor u_1 \lor \overline{u}_2 \lor v_1/\sigma$
7 $v_1=1/x_1 \lor x_2 \lor u_1 \lor \overline{u}_2 \lor v_1$
8 add $x_1 \lor x_2 \lor u_1/\sigma$
9 $u_1=1/x_1 \lor x_2 \lor u_1$
10 $u_2=0/\overline{u}_1 \lor \overline{u}_2$
11 $v_1=0/\mathrm{d}$
12 add $x_1 \lor x_2 \lor \overline{u}_1 \lor u_2 \lor v_1/\sigma$
13 $v_1=1/x_1 \lor x_2 \lor \overline{u}_1 \lor u_2 \lor v_1$
14 add $x_1 \lor x_2/\sigma$
15 $x_2=1/x_1 \lor x_2$
16 R
17 del $x_1 \lor x_2 \lor u_1 \lor \overline{u}_2 \lor v_1$
18 del $x_1 \lor x_2 \lor u_1$
19 del $x_1 \lor x_2 \lor \overline{u}_1 \lor u_2 \lor v_1$
20 $x_1=1/\mathrm{d}$
21 $x_2=1/\mathrm{d}$
22 $u_1=0/\mathrm{d}$
23 $u_2=1/u_1 \lor u_2$
24 $v_1=0/\mathrm{d}$
25 add $\overline{x}_1 \lor \overline{x}_2 \lor u_1 \lor \overline{u}_2 \lor v_1/\sigma$
26 $v_1=1/\overline{x}_1 \lor \overline{x}_2 \lor u_1 \lor \overline{u}_2 \lor v_1$
27 add $\overline{x}_1 \lor \overline{x}_2 \lor u_1/\sigma$
28 $u_1=1/\overline{x}_1 \lor \overline{x}_2 \lor u_1$
29 $u_2=0/\overline{u}_1 \lor \overline{u}_2$
30 $v_1=0/\mathrm{d}$
31 add $\overline{x}_1 \lor \overline{x}_2 \lor \overline{u}_1 \lor u_2 \lor v_1/\sigma$
32 $v_1=1/\overline{x}_1 \lor \overline{x}_2 \lor \overline{u}_1 \lor u_2 \lor v_1$
33 add $\overline{x}_1 \lor \overline{x}_2/\sigma$
34 $x_2=0/\overline{x}_1 \lor \overline{x}_2$
35 R
36 del $\overline{x}_1 \lor \overline{x}_2 \lor u_1 \lor \overline{u}_2 \lor v_1$
37 del $\overline{x}_1 \lor \overline{x}_2 \lor u_1$
38 del $\overline{x}_1 \lor \overline{x}_2 \lor \overline{u}_1 \lor u_2 \lor v_1$
39 $y_1=0/\mathrm{d}$
40 $y_2=0/\mathrm{d}$
41 $v_1=0/\mathrm{d}$
42 $v_2=1/v_1 \lor v_2$
43 $w_1=0/\mathrm{d}$
44 add $y_1 \lor y_2 \lor v_1 \lor \overline{v}_2 \lor w_1/\sigma$
45 $w_1=1/y_1 \lor y_2 \lor v_1 \lor \overline{v}_2 \lor w_1$
46 add $y_1 \lor y_2 \lor v_1/\sigma$
47 $v_1=1/y_1 \lor y_2 \lor v_1$
48 $v_2=0/\overline{v}_1 \lor \overline{v}_2$
49 $w_1=0/\mathrm{d}$
50 add $y_1 \lor y_2 \lor \overline{v}_1 \lor v_2 \lor w_1/\sigma$
51 $w_1=1/y_1 \lor y_2 \lor \overline{v}_1 \lor v_2 \lor w_1$
52 add $y_1 \lor y_2/\sigma$
53 $y_2=1/y_1 \lor y_2$
54 R
55 del $y_1 \lor y_2 \lor v_1 \lor \overline{v}_2 \lor w_1$
56 del $y_1 \lor y_2 \lor v_1$
57 del $y_1 \lor y_2 \lor \overline{v}_1 \lor v_2 \lor w_1$
58 $y_1=1/\mathrm{d}$
59 $y_2=1/\mathrm{d}$
60 $v_1=0/\mathrm{d}$
61 $v_2=1/v_1 \lor v_2$
62 $w_1=0/\mathrm{d}$
63 add $\overline{y}_1 \lor \overline{y}_2 \lor v_1 \lor \overline{v}_2 \lor w_1/\sigma$
64 $w_1=1/\overline{y}_1 \lor \overline{y}_2 \lor v_1 \lor \overline{v}_2 \lor w_1$
65 add $\overline{y}_1 \lor \overline{y}_2 \lor v_1/\sigma$
66 $v_1=1/\overline{y}_1 \lor \overline{y}_2 \lor v_1$
67 $v_2=0/\overline{v}_1 \lor \overline{v}_2$
68 $w_1=0/\mathrm{d}$
69 add $\overline{y}_1 \lor \overline{y}_2 \lor \overline{v}_1 \lor v_2 \lor w_1/\sigma$
70 $w_1=1/\overline{y}_1 \lor \overline{y}_2 \lor \overline{v}_1 \lor v_2 \lor w_1$
71 add $\overline{y}_1 \lor \overline{y}_2/\sigma$
72 $y_2=0/\overline{y}_1 \lor \overline{y}_2$
73 R
74 del $\overline{y}_1 \lor \overline{y}_2 \lor v_1 \lor \overline{v}_2 \lor w_1$
75 del $\overline{y}_1 \lor \overline{y}_2 \lor v_1$
76 del $\overline{y}_1 \lor \overline{y}_2 \lor \overline{v}_1 \lor v_2 \lor w_1$
77 $z_1=0/\mathrm{d}$
78 $z_2=0/z_1 \lor \overline{z}_2$
79 $x_1=0/\mathrm{d}$
80 $x_2=1/x_1 \lor x_2$
81 $y_1=0/\mathrm{d}$
82 add $z_1 \lor z_2 \lor x_1 \lor \overline{x}_2 \lor y_1/\sigma$
83 $y_1=1/z_1 \lor z_2 \lor x_1 \lor \overline{x}_2 \lor y_1$
84 add $z_1 \lor z_2 \lor x_1/\sigma$
85 $x_1=1/z_1 \lor z_2 \lor x_1$
86 $x_2=0/\overline{x}_1 \lor \overline{x}_2$
87 $y_1=0/\mathrm{d}$
88 add $z_1 \lor z_2 \lor \overline{x}_1 \lor x_2 \lor y_1/\sigma$
89 $y_1=1/z_1 \lor z_2 \lor \overline{x}_1 \lor x_2 \lor y_1$
90 add z_1/σ
91 $z_1=1/z_1$
92 $z_2=1/\overline{z}_1 \lor z_2$
93 $x_1=0/\mathrm{d}$
94 $x_2=1/x_1 \lor x_2$
95 $y_1=0/\mathrm{d}$
96 add $\overline{z}_1 \lor \overline{z}_2 \lor x_1 \lor \overline{x}_2 \lor y_1/\sigma$
97 $y_1=1/\overline{z}_1 \lor \overline{z}_2 \lor x_1 \lor \overline{x}_2 \lor y_1$
98 add $\overline{z}_1 \lor \overline{z}_2 \lor x_1/\sigma$
99 $x_1=1/\overline{z}_1 \lor \overline{z}_2 \lor x_1$
100 $x_2=0/\overline{x}_1 \lor \overline{x}_2$
101 $y_1=0/\mathrm{d}$
102 add $\overline{z}_1 \lor \overline{z}_2 \lor \overline{x}_1 \lor x_2 \lor y_1/\sigma$
103 $y_1=1/\overline{z}_1 \lor \overline{z}_2 \lor \overline{x}_1 \lor x_2 \lor y_1$
104 UNSAT

Figure D.8: Proof of the pebbling formula $Peb^{\oplus}_{\Pi_2}$

Theorem D.4.1. *Let F be an unsatisfiable CNF formula on n variables. There is a CDCL refutation of F without restarts and with any asserting learning scheme, unit propagation scheme, and decision scheme in time $\mathrm{O}(n2^n)$ and space at most $n-1$.*

Proof. We will let the CDCL solver use a very aggressive database reduction policy, namely to erase as many clauses as possible from the database. This means that when the database is reduced we delete all clauses except those that are reason clauses in the current trail, and so in formal notation the clause database after a reduction step in state $(F, \mathcal{D}, s)$ is

$$\mathcal{D}' = F \cup \{C \mid x = b/C \in s\} \ . \tag{D.4.1}$$

We first argue that the clause database contains at most $n-1$ learned clauses at every step. The computation begins with unit propagations at decision level 0. If there is a conflict then the trace ends, otherwise the solver reaches a stable state. In this phase we do not learn any clauses. Afterwards the clause database is pruned at each stable state and keeps growing until the solver reaches the next stable state (or terminates).

Let $(F, \mathcal{D}, s)$ be the state of the solver right after an erasure step, and let d be the decision level of s. By construction $\mathcal{D} \setminus F$ has at most one clause for each unit propagation in the sequence s, therefore $d + |\mathcal{D} \setminus F| \leq |s|$. Observe that if $|s| > n-2$ then $F{\restriction_s}$ would have either a unit or an empty clause. Since we are in a stable state, $|s| \leq n-2$.

Now consider an arbitrary state $(F, \mathcal{D}', s')$, and consider the state right after the most recent erasure. Call such state $(F, \mathcal{D}, s)$, and let d be the decision level of s. After an erasure, the solver does a decision step that increases the decision level to $d+1$. Each time the solver reaches **Conflict** mode the decision level decreases, and does not increase anymore until the next stable state. So there can be at most $d+1$ conflicts which result in learning a clause (a conflict at level zero would just end the computation, without changing the clause database). Thus $|\mathcal{D}' \setminus F| \leq d + 1 + |\mathcal{D} \setminus F| \leq n-1$ as we wanted.

Finally we prove that the solver terminates in time $\mathrm{O}(n2^n)$. Given a trail s we define the string $b(s) \in \{0,1\}^{|s|}$ by letting the coordinates be

$$b(s)_i = \begin{cases} 0 & \text{if } s_i \text{ is a decision} \\ 1 & \text{if } s_i \text{ is a unit propagation} \end{cases} \tag{D.4.2}$$

for $1 \leq i \leq |s|$.

We establish the upper bound on time by exhibiting a subsequence of states such that the corresponding subsequence $b(s)$ is strictly increasing with respect to the lexicographic order, and there are a constant number of states between any two states in the subsequence. The length of a string $b(s)$ is at most n, so the subsequence has length at most $2^{n+1}-1$ and the time upper bound follows by recalling that a trivial resolution has length at most n.

Consider a stable state $(F, \mathcal{D}, s)$. The solver applies an erasure and a decision, ending in some state $(F, \mathcal{D}', s')$ where $s' = s \cup (x = v/\mathsf{d})$. The string increases since $b(s') =$

$b(s)0 > b(s)$. If the state $(F, \mathcal{D}, s)$ causes a unit propagation, then the new state is $(F, \mathcal{D}, s')$ with $b(s') = b(s)1 > b(s)$. In the case of a conflict state $(F, \mathcal{D}, s)$ of decision level d, the solver learns a clause C of assertion level $d' < d$ (unless $d = 0$, in which case the trace ends). The solver gets to state $(F, \mathcal{D}', s')$ where $\mathcal{D}' = \mathcal{D} \cup \{C\}$ and s' is the prefix of s of decision level d', and then immediately unit propagates the asserting literal. Thus the trail of the next state s'' has $b(s'') = b(s')1$, while the prefix of length $|s'| + 1$ of $b(s)$ is equal to $b(s')0$ by construction, and so $b(s'') > b(s)$. □

Theorem D.4.1 shows that CDCL without restarts can solve any formula in exponential time and linear space. The simulation works with any decision heuristic and clause learning scheme. It doesn't give much insight in the power of CDCL compared to resolution, though. Could CDCL without restarts simulate tree-like resolution, or even regular resolution? Could it also simulate these with respect to space? These questions are much more difficult, as we need to adapt the strategy to concrete refutations, and show that the simulation is efficient compared to the refutation. Therefore, we consider formula families instead of arbitrary CNFs and show simulation results of CDCL without restarts for these families.

D.5 Trade-offs for Pebbling Formulas

In this section we discuss trade-offs between space and running time for CDCL refutations of the formulas based on pebble games that we introduced in Section D.3. Let us start with a brief review of those pebble games.

The *black-white pebble game* [67] is played by a single player on a DAG G with all vertices having indegree at most 2 and with a single sink (i.e., a vertex with no outgoing edges). At each step the player can either place or remove a pebble. A black pebble can be placed on a vertex if all its predecessors have pebbles, and it can be removed at any time. A white pebble can be placed at any time, but it can be removed from a vertex only if all its predecessors contain pebbles. The game starts with the DAG being empty, and the goal is to reach a position where there is a black pebble on the sink and the rest of the graph contains no pebbles. We call a sequence of moves that reaches the goal a *complete pebbling*. The *time* of a pebbling is the number of steps, and the *space* is the maximum number of pebbles in G at any point in time during the pebbling. The *black pebble game* [149] that we introduced in Section D.3 is a restricted version where only black pebbles are allowed. The interested reader can find more information about pebble games and a comparison of the various flavours of pebbling in the surveys [153, 146].

Recall that, given a DAG G, the XORified pebbling formula $Peb_G^{\oplus}$ is defined as follows. For every vertex $v \in V(G)$ there are two variables v_1 and v_2. There are clauses encoding the following: for each source vertex of G, the parity of its variable pair is odd, for each internal vertex, odd parity on all predecessors of the vertex implies odd parity on the vertex itself, and the parity on the sink is even.

More precisely, if we denote as $pred(v)$ the set of all predecessors of v (i.e., vertices that have an outgoing edge toward v), the pebbling formula consists of the set of constraints

$$v_1 \oplus v_2 = 1 \qquad \text{for each source } v, \qquad \text{(D.5.1a)}$$

$$\bigwedge_{u \in pred(v)} (u_1 \oplus u_2 = 1) \rightarrow v_1 \oplus v_2 = 1 \qquad \text{for each internal vertex } v, \qquad \text{(D.5.1b)}$$

$$z_1 \oplus z_2 = 0 \qquad \text{for the sink } z, \qquad \text{(D.5.1c)}$$

expressed in CNF form. We refer to (D.5.1a) as *source axioms*, to (D.5.1b) as *pebbling axioms*, and to (D.5.1c) as *sink axioms*.

It is straightforward to transform a black pebbling of G into a resolution proof for $Peb^{\oplus}_{G}$. For a vertex v, we call $v_1 \lor v_2$ and $\overline{v}_1 \lor \overline{v}_2$ the *pebble clauses* for v. We maintain the invariant that in the resolution proof, we have in memory the pebble clauses for all pebbled vertices. Each time a pebble is removed, we erase the corresponding pebble clauses. Correctness of the simulation follows because we can derive the pebble clauses for v from the pebble clauses for all its predecessors, and in a black pebbling, we can only pebble a vertex when all its predecessors are pebbled. This simulation yields the following lemma.

Lemma D.5.1 ([28]). *If G has a black pebbling of time τ and space s, then $Peb^{\oplus}_{G}$ has a resolution refutation of length* $O(\tau)$ *and clause space* $O(s)$*.*

In the other direction, it can be shown (although it is substantially more complicated) that any resolution refutation in length L and space s of $Peb^{\oplus}_{G}$ can be converted to a pebbling of G with asymptotically the same bounds on time and space, but this requires the full black-white pebble game.

Theorem D.5.2 ([28]). *From a resolution refutation of $Peb^{\oplus}_{G}$ of length L and clause space s we can extract a black-white pebbling for G with time* $O(L)$ *and space* $O(s)$*.*

D.5.1 Trade-offs with Restarts

As a warm-up, let us finish showing how to translate black pebblings into CDCL proofs with unrestricted restarts. Recall that our plan in Section D.3 was to proceed in the same way that we translate pebblings into resolution proofs. This is, for each pebbled vertex v we keep the pebble clauses in memory, except if v is a source, in which case these clauses are already source axioms of the pebbling formula.

Formally, we generate a CDCL trace using the following procedure. The trace is legal for the 1UIP learning scheme, and we discuss how to use a more general scheme later on.

`HalfPebble` sets the parity of a vertex to even and then learns the clause forbidding that assignment. The example in Figure D.2 is actually the translation into resolution of the trace generated by `HalfPebble` $(w, 0)$.

```
Procedure Pebble($\mathcal{P}$)
  Input: a black pebbling $\mathcal{P}$
1 foreach (move,v) in $\mathcal{P}$ where $v$ is not a source or the sink do
2     if move is Add then
3         HalfPebble ($v$, 0)
4         HalfPebble ($v$, 1)
5     else
6         del $v_1 \vee v_2$
7         del $\overline{v}_1 \vee \overline{v}_2$
8 PebbleSink
```

```
Procedure HalfPebble($v$, $b$)
  Input: A vertex $v$, a boolean $b$
1 Decide $v_1 = b/\mathsf{d}$
2 Decide $v_2 = b/\mathsf{d}$
3 FindConflicts ($v$, b)
4 Learn $v_1^{1-b} \vee v_2^{1-b}$ and assert $v_2 = 1 - b/\mathsf{u}$
5 Restart R
6 foreach clause $C \in \mathcal{D} \setminus F$ such that $|C| > 2$ do
7     del $C$
```

```
Procedure FindConflict(Case 1, $w$, $b$)
  Input: A vertex $w$ with predecessor $u$, a boolean $b$
1 Decide $u_1 = 0/\mathsf{d}$
2 Learn $w_1^{1-b} \vee w_2^{1-b} \vee u_1$ and assert $u_1 = 1/\mathsf{u}$
```

FindConflicts generates a conflict between an even vertex and its predecessors. We need to distinguish two cases depending on whether the vertex we are learning has one or two predecessors.

PebbleSink is a tweak of HalfPebble that accounts for propagations caused by sink axioms.

Using the decision and clause erasure strategies in this procedure, we can establish an analogue of Lemma D.5.1 as follows.

Lemma D.5.3. *If $\mathcal{P}$ is a black pebbling of G in space s and time τ, then there is a CDCL proof of $Peb_G^{\oplus}$ with restarts and any cutting learning scheme and unit propagation scheme in space at most $2s + 3$ and time $\mathrm{O}(\tau)$.*

Proof. The procedure Pebble maintains the following invariant. A non-source vertex v

Procedure FindConflict(Case 2, w, b)

```
Input: A vertex $w$ with predecessors $u$ and $v$, a boolean $b$
1 Decide $u_1 = 0/\mathsf{d}$
2 Propagate $u_2 = 1/u_1 \vee u_2$
3 Decide $v_1 = 0/\mathsf{d}$
4 Learn $w_1^{1-b} \vee w_2^{1-b} \vee u_1 \vee \overline{u}_2 \vee v_1$ and assert $v_1 = 1/\mathsf{u}$
5 Learn $w_1^{1-b} \vee w_2^{1-b} \vee u_1$ and assert $u_1 = 1/\mathsf{u}$
6 Propagate $u_2 = 0/\overline{u}_1 \vee \overline{u}_2$
7 Decide $v_1 = 0/\mathsf{d}$
8 Learn $w_1^{1-b} \vee w_2^{1-b} \vee \overline{u}_1 \vee u_2 \vee v_1$ and assert $v_1 = 1/\mathsf{u}$
```

Procedure PebbleSink

```
1 Decide $z_1 = 0/\mathsf{d}$
2 Propagate $y_z = 0/z_1 \vee \overline{z}_2$
3 FindConflicts ($z$, 0)
4 Learn $z_1$ and assert $z_1 = 1/\mathsf{u}$
5 Propagate $y_z = 1/z_1 \vee \overline{z}_2$
6 FindConflicts ($z$, 1)
```

has a pebble if and only if the pebbling clauses of v are in the clause database. No other clauses are in memory. The bound on space follows from this invariant, and the bound on time is by construction.

It remains to check that the trace is legal. Observe that the clauses we have in memory only propagate if at least one variable from each vertex they mention is set. Since the decision sequence mentions at most 3 vertices at the same time, we only need to reason about clauses that mention only these vertices.

The correctness of `FindConflicts` is easy to verify. There is no choice of which variable to propagate.

At the end of `FindConflicts` w_1 and w_2 are decisions, and all variables below are set by unit propagation. Since one of the conflicting clauses, a pebbling axiom, contains the decision variables w_1 and w_2, they appear in any cut and therefore any conflict clause. The remaining variables in the conflict graph are of last decision level, so they cannot appear in an asserting clause because w_2 already appears. Therefore we learn the clause $w_1^{1-b} \vee w_2^{1-b}$ and assert $w_2 = 1 - b/\mathsf{u}$.

We only erase clauses after a restart, so we are not erasing clauses involved in unit propagation.

Finally, observe that `PebbleSink` is doing essentially the same as two calls to `HalfPebble`, except that at line 4 there is exactly one decision, z_1, so this is the clause that we learn, and that at the end there is a conflict at level 0, so the proof ends.

To conclude the proof we need to argue that we can replace 1UIP by any cutting learning scheme. The argument to determine that at the end of `FindConflicts` we learn a pebble clause still works.

The remaining clauses that we learn are only used to unit propagate once, so learning any clause that asserts the same literal will serve the same purpose. Since the decision variable appears in a conflicting clause, it is the only possible asserting literal. Therefore we can replace all lines of the form "Learn C and assert $x = b/\mathsf{u}$" by "Learn $L(F, \mathcal{D}, s)$ and assert $x = b/\mathsf{u}$", where L is a given learning scheme and x is the same variable. □

D.5.2 Trade-offs without Restarts

In Section D.5.1, it was shown that CDCL with restarts can do a time and space efficient simulation of any black pebbling. The upper bounds we claim in Section D.3 are for the more challenging setting where CDCL is not allowed to restart, however. To establish such results, we need to modify the proofs substantially.

Recall the state at line 4 of `HalfPebble` after learning the first pebble clause of a vertex v. The vertex is set to odd, so there cannot be conflicts among v and its predecessors, and we cannot learn the second pebble clause. Since we are not allowed to restart, the search has to proceed elsewhere. It turns out that we can systematically control the process of learning a clause, moving to another part of the graph, and then learning the remaining clause. However, we have to settle for a restricted class of pebblings and we have to impose some additional structure on them.

We do the conversion from standard pebblings to CDCL refutations in three steps. First we convert the pebbling into a recursive form that suits our arguments better. Unfortunately this conversion needs to be tailored for every graph. The next step is to modify the pebbling to make it easier to simulate. This step is universal, but we need to modify the underlying graph, therefore the formulas that we prove trade-offs for differ slightly from those in [28]. The last step is to actually generate a CDCL trace from the pebbling.

We are going to show the specific form of black pebbling that has an efficient translation in the CDCL proof system in Section D.5.3. In Section D.5.4 we are going to explain the simulation itself. Finally in Section D.5.5 we will revisit graphs with known time/space trade-offs and recast the time- and space-efficient pebblings of these graphs into our new framework.

D.5.3 Binary Tree Pebbling

We can express our conditions more easily if we think of a pebbling recursively. Therefore we first introduce the notion of binary tree pebbling as a useful language to represent black pebbling strategies with a recursive structure.

In this section we denote by $\mathcal{T}$ a rooted binary tree with children ordered left and right. We use the convention that a single child is the left child. We also assume that $G = (V, E)$ is a DAG with exactly one sink and such that each vertex has at most two incoming edges. To avoid confusion between a DAG G and a tree $\mathcal{T}$ describing a pebbling of G, we say that G has *vertices*, which we usually denote by u, v, w, and $\mathcal{T}$ has *nodes*, which we denote by p, q, l, r.

The *left postfix order* of the nodes of a tree $\mathcal{T}$ is the total order $(p_1, p_2, \ldots, p_{|\mathcal{T}|})$ induced by a traversal of $\mathcal{T}$ that first visits the left subtree, then the right subtree (if any), and then the root. We say that p_i *comes at time* i in the traversal and we denote as $p < p'$ the fact that a node p comes at an earlier time than node p'.

Definition D.5.4 (Binary tree pebbling). A *binary tree pebbling* of G is a binary tree $\mathcal{T}$ whose nodes are labelled by vertices of G according to a label function ℓ and such that the following holds:

1. if p is an *internal* node of $\mathcal{T}$ the label function ℓ forms a bijection between the children of p and the predecessors of $\ell(p)$ in G;
2. if p is a leaf of $\mathcal{T}$, then either $\ell(p)$ is a source vertex of G or there is an internal node $q < p$ such that $\ell(p) = \ell(q)$; we call the latest such q the *cache node* for p.

The time of a binary tree pebbling is its order.

A binary tree pebbling corresponds to a generalized depth first search of G that starts at the sink and that can mark a vertex as “visited” only if all its predecessors are maked as “visited” and that, unlike standard DFS, can remove the “visited” mark at any stage of the process. If the process reaches any such vertex again, it has to visit it again, as if it had never reached it before.

The converse also holds: a standard depth first traversal corresponds to a pebbling strategy where we remember all “visited” nodes. Indeed, consider the spanning tree produced by a depth first traversal of a graph, except that instead of discarding a forward edge, we add a new leaf labelled as the already visited node the edge is pointing to. It is easy to verify that such binary tree is a pebbling and that its order is at most $|V| + |E|$.

Proposition D.5.5. *Every DAG has a binary tree pebbling of time at most* $|V| + |E|$.

Although we are more interested in the reverse direction, let us first show that that binary tree pebblings are essentially black pebblings. We extract a pebbling from a binary tree pebbling by traversing the tree in postorder and keeping a pebble on a vertex as long as it is useful.

Definition D.5.6 (Node lifespan). The *lifespan* of a non-root internal node p_i is the maximal interval $[i, j]$ such that p_j is the parent of either p_i or a non-source leaf whose cache node is p_i.

The lifespan of a leaf node p_i is only interesting when $\ell(p_i)$ is a source of G, and it is $[i, j]$ where p_j is the parent node of p_i. If a leaf node p_i is labelled by a non source vertex of G we say by convention that its lifespan is the empty interval. The lifespan of the root node is simply $[|\mathcal{T}|, |\mathcal{T}|]$.

We say that some node p_i is *alive at time* t if it has lifespan $[i, j]$ with $i \leq t \leq j$.

Definition D.5.7. The space of a binary pebbling tree $\mathcal{T}$, denoted as *space*$(\mathcal{T})$, is the maximum number of nodes alive at time t for $1 \leq t \leq |\mathcal{T}|$.

Proposition D.5.8. *If a DAG G has a binary tree pebbling $\mathcal{T}$, then G has a black pebbling with space space$(\mathcal{T})$ and time at most $2|\mathcal{T}|$.*

Proof. Consider a binary tree pebbling $\mathcal{T}$ and the corresponding left postfix order $(p_1, p_2, \ldots p_{|\mathcal{T}|})$. While we scan this sequence we build a pebbling of G as follows. At time t we consider whether node p_t has a lifespan $[t, j]$ for some $j \geq t$. In this case we place a pebble on $\ell(p_t)$. Then, for each node p_i with lifespan $[i, t]$, we remove the pebble on $\ell(p_i)$.

This pebbling has space exactly *space*$(\mathcal{T})$ and achieves the goal to place a pebble on the sink of G. At each node we place at most one pebble and we remove as many pebbles as we place, so the number of moves is at most $2|\mathcal{T}|$. Now we show that it is legal, i.e., for every node p_t with parent $p_{t'}$ there is a pebble on $\ell(p_t)$ at least until time t'. This guarantees that every pebble placement is legal. If p_t has a lifespan $[t, s']$ then by definition $s' \geq t'$. If p_t has no lifespan it means that there is a previous internal node p_s with $\ell(p_t) = \ell(p_s)$ with lifespan $[s, s']$ and $s' \geq t'$. □

Finally we discuss the strategy opposite to that in Proposition D.5.5: discard pebbles as soon as possible.

Proposition D.5.9. *Every DAG with depth d has a binary tree pebbling of time at most $2^{d+1} - 1$ and space $d + 2$, in which all leaves are labelled by source vertices.*

Proof. If the graph has depth 0 it consists of a single vertex and we are done. Otherwise consider the sink vertex u and assume without loss of generality that it has two children, v and w. The binary tree pebbling has the root node labelled by u, and two subtrees of order $2^d - 1$ and space $d + 1$ each, obtained by considering the binary tree pebbling of the subgraph of the ancestors of v and w, respectively. The total order of the tree is $2^{d+1} - 1$ and the total space is $d + 2$ since the node at the root of the left subtree is the only node in the left subtree which is alive during the whole visit of the right subtree. When the root is visited, both its left and right children are alive, so the space must be at least 3, but $d + 2 \geq 3$ if $d \geq 1$. □

In order to do the translation from a pebbling strategy to a CDCL refutation we need the binary tree pebbling to fulfill some further properties.

Definition D.5.10 (Robustness). We say that a binary tree pebbling $\mathcal{T}$ of graph G is *robust* if satisfies the following additional properties.

1. Each internal node has exactly two children;

2. if p is an internal node of $\mathcal{T}$, and l and r its left and right children, then for every node $q \in \mathcal{T}_r$ it holds that $\ell(l) \notin \{\ell(q)\} \cup pred_G(\ell(q))$; and

3. if an internal node q is the right child of l, then q is not the cache node of any non-source leaf in the subtree rooted in r.

Enforcing the robustness property is necessary but luckily not very problematic. We will argue that is is possible to modify any graph G into a new graph $r(G)$ so that any binary tree pebbling $\mathcal{T}$ for G can be turned into a binary tree pebbling $r(\mathcal{T})$ for $r(G)$ with negligible penalty in space and time.

We split every edge of the original graph in two, by adding a new vertex, and then we add a new source vertex as a new predecessors of all vertices of incoming degree one (this includes the vertices we just added to split the edges). Formally, the robust graph is the following.

Definition D.5.11. We define $r(G) = (V', E' \cup E'')$ where $V' = V \cup E \cup \{*\}$ and

$$\begin{aligned} E' &= \{(v,e),(e,v') \mid e \in E \text{ and } e = (v,v')\} \\ E'' &= \{(*,u) \mid u \in V \text{ and } |pred(u)| = 1 \text{ or } u \in E\} \ . \end{aligned}$$

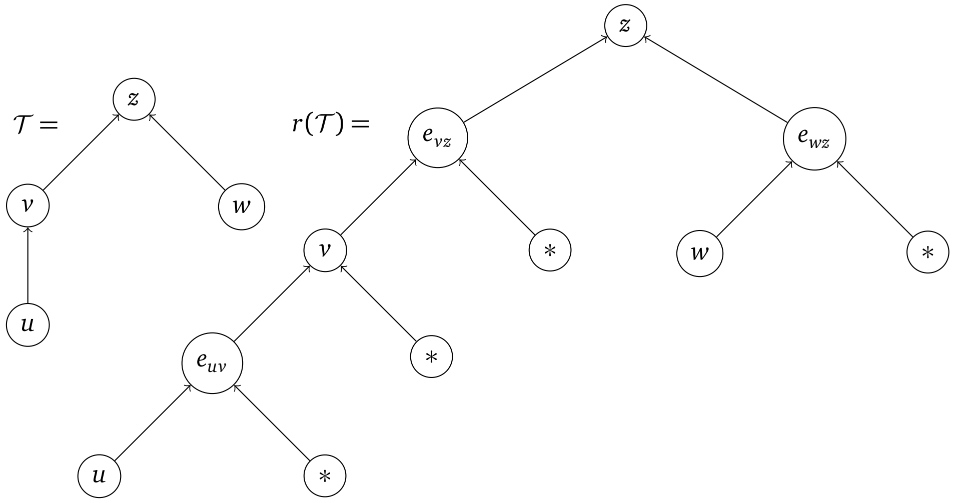


Figure D.9: Robust transform of a binary pebbling

Proposition D.5.12. *For every black (resp. black-white) pebbling of G of time τ and space s there is a black (resp. black-white) pebbling of $r(G)$ of time $O(\tau)$ and space $s + 2$. For every black (resp. black-white) pebbling of $r(G)$ of time τ and space s there is a black (resp. black-white) pebbling of G of time $O(\tau)$ and space s.*

Proof. The transformation of a pebbling for G into a pebbling for $r(G)$ is trivial. For the opposite direction consider a pebbling for $r(G)$, either black or black-white. We will build a pebbling of G of the claimed length and space, translating pebbling moves in $r(G)$ into moves for G. First we remove vertex $*$ from $r(G)$ to get a new graph $r(G)'$. A vertex removal can only decrease pebbling length and space.

Now we transform the pebbling of $r(G)'$ into a pebbling of G, and we maintain the invariants that: (a) whenever some vertex $e = (v, v')$ has a pebble in the original pebbling, then v must be pebbled in the new pebbling; (b) if a vertex v, coming from the original graph G, has a pebble, then it has a pebble also in the new pebbling; and (c) the pebble on v is white only if there is a white pebble on v in the original pebbling.

When a white pebble is placed on vertex e we instead place a pebble on v, unless it is already pebbled. When a black pebble is placed on vertex e we don't do anything since v must have been pebbled at that point. If a white pebble is placed on v in $r(G)$ we do the same in G unless there is already a black pebble there. If a black pebble is placed on v in $r(G)$ then in G we have a pebble on each predecessor of v. We can then black pebble v in G as well (after removing any white pebble on the same vertex). If a pebble is removed from v, we remove it from G only if in $r(G)$ the vertices representing all the outgoing edges of v in G are empty. If the pebble was white then it means that predecessors of v in $r(G)$ (and then in G) are pebbled. So we turn the pebble black. If e is unpebbled and neither v nor any other e' successors of v are pebbled in the original pebbling, then in our pebbling we remove the pebble over v, which is black because v does not have a pebble in the original pebbling.

This pebbling of G has no more pebbles than the one of $r(G)$, and we always turn one move over $r(G)$ into $\mathrm{O}(1)$ moves on G. □

Lemma D.5.13. *For any binary tree pebbling $\mathcal{T}$ of G it is possible to efficiently build a robust binary tree pebbling $r(\mathcal{T})$ of $r(G)$ with space* $\mathrm{O}(\mathit{space}(\mathcal{T}))$ *and time* $\mathrm{O}(|\mathcal{T}|)$.

Proof. We build a robust binary tree pebbling $r(\mathcal{T})$ from $\mathcal{T}$ by first splitting edges in two and then adding a node labelled by $*$ as the right child of every node of incoming degree 1, in a way that matches what we did for G in Definition D.5.11.

More specifically each edge $e = (s, t)$ of $\mathcal{T}$ is substituted by the path of three nodes s, p_e, t, where s and t are the original nodes of $\mathcal{T}$ and p_e is a new one, labelled by $(\ell(s), \ell(t))$. If s was the left (resp. right) child of t then p_e is the new left (resp. right) child of t. In any case s is always the left child of p_e. To complete the construction we add a new right child node $p_{v,*}$, labelled by $*$, to any internal node v of $\mathcal{T}$ without a right child. The order of $r(\mathcal{T})$ is at most $\mathrm{O}(|\mathcal{T}|)$ by construction.

To analyze the space of $r(\mathcal{T})$ we observe a few simple facts from the construction: (a) each of the labels of the new nodes of type p_e occur only once; (b) the left postfix order of $\mathcal{T}$ is the projection of $r(\mathcal{T})$ over the nodes in $V(\mathcal{T})$. These facts, together with Definition D.5.6, imply that the lifespan of each $p \in V(\mathcal{T})$ in the visit of $r(\mathcal{T})$ contains the projection of the lifespan of the same node p in the visit of $\mathcal{T}$. Therefore no more than $\mathit{space}(\mathcal{T})$ nodes in $V(\mathcal{T})$ can be simultaneously alive.

The new nodes $p_{v,*}$ are alive just until the next step. The lifespan of a node p_e with $e = (p, q)$ ends no later than when q is visited. For comparison, in the left postfix order

Procedure Cleanup(p_j)

```
1 foreach clause $C \in \mathcal{D} \setminus F \setminus \bigcup_{p \text{ alive at time } j} \mathit{scaffolding}(p, j)$ do
2     del $C$
```

of $\mathcal{T}$ the lifespan of p ends no earlier than at the visit of q. Thus we can charge each p_e to the lifespan of p in the left postfix order of $\mathcal{T}$. Since the mapping from p_e to p is injective, the total contribution of these nodes is $\mathit{space}(\mathcal{T})$.

In the end we get that $\mathit{space}(r(\mathcal{T})) = 2\mathit{space}(\mathcal{T}) + \mathrm{O}(1)$.

To conclude we need to show that $r(\mathcal{T})$ is indeed robust. It is easy to see that the new tree respects all conditions of Definition D.5.4 and that every internal node has two children, but the other conditions in D.5.10 require an explicit proof.

Condition 2 is verified immediately for nodes for types p_e and $p_{u,*}$ that have trivial (or null) right subtrees. For an arbitrary node p labelled by a vertex u, its left child is labelled by some vertex e which corresponds to an edge in G, and such label occur only once in $r(\mathcal{T})$, and in particular cannot occur in the right subtree of p. Furthermore, the only node that has e as a predecessor is u, so e does not appear as a predecessor of a node in the right subtree of p either.

With respect to condition 3, the only nodes that appear as right children are labelled by $*$, which is not a cache node because it is a source, and by edges of G, which are not cache nodes either because leaves are labelled after vertices in G. □

D.5.4 CDCL Trace from a Tree Pebbling

In this section we consider a DAG G and a robust binary tree pebbling $\mathcal{T}$ for it. We want to describe a CDCL proof that refutes $\mathit{Peb}_G^{\oplus}$ in time and space proportional to the time and space of $\mathcal{T}$, respectively.

We build the proof recursively by traversing the binary tree pebbling according to the left postfix order. Ideally we would like to learn the pebble clauses for the label of each visited node and keep such clauses for the lifespan of the node itself. As we argued, this is not possible without restarts, but we roughly achieve this goal by learning the clauses corresponding to some ancestors of the visited node within a fixed constant distance. This results in a constant number of binary clauses per visited node.

More precisely, we define the *scaffolding* of a node p alive at time j as the set of descendants of p that have a pebble at time j and are reachable from p by a path of nodes that do not have a pebble. For instance, the scaffolding of a pebbled node is the node itself, and the scaffolding of an unpebbled node whose two predecessors are pebbled are its predecessors. We will ensure that the size of any scaffolding never exceeds 3.

Our deletion strategy is, after processing p_j, to keep the scaffolding of each node alive at time j and to erase every other clause.

```
Procedure QuickVisit(Case 1, $b$, $\mathcal{T}$)
  Input: a robust binary tree pebbling $\mathcal{T}$ with root $p$.
1 Decide $\ell(l)_1 = 0/\mathsf{d}$
2 Propagate $\ell(l)_2 = 1/\ell(l)_1 \vee \ell(l)_2$
3 Decide $\ell(r)_1 = 0/\mathsf{d}$
4 Learn $\ell(p)_1^{1-b} \vee \ell(p)_2^{1-b} \vee \ell(l)_1 \vee \overline{\ell(l)}_2 \vee \ell(r)_1$ and assert $\ell(r)_1 = 1/\mathsf{u}$
5 Learn $\ell(p)_1^{1-b} \vee \ell(p)_2^{1-b} \vee \ell(l)_1$ and assert $\ell(l)_1 = 1/\mathsf{u}$
6 Propagate $\ell(l)_2 = 0/\overline{\ell(l)}_1 \vee \overline{\ell(l)}_2$
7 Decide $\ell(r)_1 = 0/\mathsf{d}$
8 Learn $\ell(p)_1^{1-b} \vee \ell(p)_2^{1-b} \vee \ell(l)_1 \vee \overline{\ell(l)}_2 \vee \ell(r)_1$ and assert $\ell(r)_1 = 1/\mathsf{u}$
```

It is important to delete clauses in a timely manner, not only for keeping the size of the clause database in shape but to avoid spurious unit propagations.

In the following text, we assume that $\mathcal{T}'$ is a subtree of $\mathcal{T}$, with root p and children l and q. We say that a node p is pebbled if the pebble clauses of $\ell(p)$ are in the clause database. We also assume the learning scheme to be 1UIP, and we appeal to the argument in Section D.5.1 to show that this is without loss of generality.

To learn pebble clauses we will use two auxiliary procedures, `DeepVisit` and `QuickVisit`. The former is defined recursively and it is the main component of our CDCL refutation. The latter is the base case and only works for some specific trees of constant size. A visit procedure starts with p set to even, sets some variables in $\mathcal{T}'$, and ends with a conflict at the level of $\ell(p)_2$ or earlier. It is the responsibility of the caller to learn the appropriate clause.

We use `QuickVisit` in the following three cases, where we assume that p is not pebbled.

1. l and r have a pebble.
2. r has a pebble, l does not, but it has two children, l_1 and l_2, that are pebbled.
3. l has a pebble, r does not, but it has two children, r_1 and r_2, that are pebbled.

Case 1 of `QuickVisit` is essentially `FindConflicts` from Section D.5.1 with the notation adapted to binary pebbling.

In Case 2 we also generate a conflict with the predecessors of a node and we pebble the left node as a side-effect. First we learn the clause $\ell(l)_1 \vee \ell(l)_2$ using Case 1 `QuickVisit`, analogously to `HalfPebble`. Then instead of restarting we generate a conflict earlier in the trail, with the same goal of clearing the left subtree. Now we can apply Case 1 again and learn $\overline{\ell(l)}_1 \vee \overline{\ell(l)}_2$. Finally we generate another conflict and backjump.

Finally, Case 3 is essentially the same as Case 2, flipping the left and the right subtrees.

Procedure QuickVisit(Case 2, b, $\mathcal{T}$)

```
Input: a robust binary tree pebbling $\mathcal{T}$ with root $p$.
1  Decide $\ell(l)_1 = 0/d$
2  Decide $\ell(l)_2 = 0/d$
3  QuickVisit (Case 1, $b = 0$, $\mathcal{T}_l$)
4  Learn $\ell(l)_1 \vee \ell(l)_2$ and assert $\ell(l)_2 = 1/u$
5  Decide $\ell(r)_1 = 0/d$
6  Learn $\ell(p)_1 \vee \ell(p)_2 \vee \ell(l)_1 \vee \overline{\ell(l)}_2 \vee \ell(r)_1$ and assert $\ell(r)_1 = 1/u$
7  Learn $\ell(p)_1 \vee \ell(p)_2 \vee \ell(l)_1$ and assert $\ell(l)_1 = 1/u$
8  Decide $\ell(l)_2 = 1/d$
9  QuickVisit (Case 1, $b = 1$, $\mathcal{T}_l$)
10 Learn $\overline{\ell(l)}_1 \vee \overline{\ell(l)}_2$ and assert $\ell(l)_2 = 0/u$
11 Decide $\ell(r)_1 = 0/d$
12 Learn $\ell(p)_1 \vee \ell(p)_2 \vee \ell(l)_1 \vee \overline{\ell(l)}_2 \vee \ell(r)_1$ and assert $\ell(r)_1 = 1/u$
```

The recursive procedure for large trees is `DeepVisit`. In order to learn the pebble needed at some node p of the binary tree pebbling, we need to visit both subtrees multiple times. The first time we visit each subtree we use `DeepVisit` which, as a side effect, learns the pebbles of the nodes at a constant distance from p. Afterwards, since a pebbled node behaves as a source, the subtrees become effectively of constant size. We can use `QuickVisit` where appropriate to generate local conflicts and end with a backjump.

Finally we pebble the root and start the recursive procedure in `BinaryPebble`.

We can now establish the technical lemma that is the main goal of this section.

Lemma D.5.14 (CDCL refutation). *If $\mathcal{T}$ is a robust binary tree pebbling of a graph G, then* `BinaryPebble` *($\mathcal{T}$) is a CDCL proof of $Peb^{\oplus}_{G}$ without restarts and with any cutting learning scheme and unit propagation scheme in time* $O(|\mathcal{T}|)$ *and space at most* $O(space(\mathcal{T}))$

The number of steps in the trace is $O(|\mathcal{T}|)$ because `QuickVisit` always runs in constant time and `DeepVisit` calls itself recursively at most once on each subtree. Furthermore, all conflicts involve a constant number of variables, so all derivations are of constant size as well.

To measure the space we need the following invariant.

Claim D.5.15. *After we run* `DeepVisit` *on a subtree rooted in p, with left and right children l and r, we store a pebble on l and on either r or its two children.*

Claim D.5.16. *The scaffolding of every node alive at time j has size at most 3.*

Proof. Observe that the order in which `DeepVisit` processes nodes is the left postfix order of $\mathcal{T}$, except that some of the calls to `DeepVisit` actually result in calls to `QuickVisit` on some constant size subtrees. Therefore every node alive at time j has been visited.

Procedure DeepVisit($\mathcal{T}$)

```
Input: a robust binary tree pebbling $\mathcal{T}$ with root $p$.
If $\mathcal{T}$ meets the requirements use QuickVisit instead, otherwise go ahead.
Decide $\ell(l)_1 = 0/\mathsf{d}$
if $\ell(l)$ is pebbled then
    Propagate $\ell(l)_2 = 1/\ell(l)_1 \lor \ell(l)_2$
else
    Decide $\ell(l)_2 = 0/\mathsf{d}$
    DeepVisit ($\mathcal{T}_l$)
    Learn $\ell(l)_1 \lor \ell(l)_2$ and assert $\ell(l)_2 = 1/\mathsf{u}$
    Cleanup ($l$)
Decide $\ell(r)_1 = 0/\mathsf{d}$
if $\ell(r)$ is not pebbled then
    Propagate $\ell(r)_2 = 0/\ell(p)_1 \lor \ell(p)_2 \lor \ell(l)_1 \lor \overline{\ell(l)}_2 \lor \ell(r)_1 \lor \overline{\ell(r)}_2$
    DeepVisit ($\mathcal{T}_r$)
Learn $\ell(p)_1 \lor \ell(p)_2 \lor \ell(l)_1 \lor \overline{\ell(l)}_2 \lor \ell(r)_1$ and assert $\ell(r)_1 = 1/u$
if $\ell(r)$ is not pebbled then
    Propagate $\ell(r)_2 = 1/\ell(p)_1 \lor \ell(p)_2 \lor \ell(l)_1 \lor \overline{\ell(l)}_2 \lor \ell(r)_1 \lor \overline{\ell(r)}_2$
    Cleanup ($r$)
    QuickVisit ($\mathcal{T}_r$, $b = 1$)
Learn $\ell(p)_1 \lor \ell(p)_2 \lor \ell(l)_1$ and assert $\ell(l)_1 = 1/\mathsf{u}$
if $\ell(l)$ is pebbled then
    Propagate $\ell(l)_2 = 1/\overline{\ell(l)}_1 \lor \overline{\ell(l)}_2$
else
    Decide $\ell(l)_2 = 1/\mathsf{d}$
    QuickVisit ($\mathcal{T}_l$, $b = 1$)
    Learn $\overline{\ell(l)}_1 \lor \overline{\ell(l)}_2$ and assert $\ell(l)_2 = 0/u$
Decide $\ell(r)_1 = 0/\mathsf{d}$
if $\ell(r)$ is not pebbled then
    Propagate $\ell(r)_2 = 0/\ell(p)_1 \lor \ell(p)_2 \lor \overline{\ell(l)}_1 \lor \ell(l)_2 \lor \ell(r)_1 \lor \overline{\ell(r)}_2$
    QuickVisit ($\mathcal{T}_r$, $b = 0$)
Learn $\ell(p)_1 \lor \ell(p)_2 \lor \ell(l)_2 \lor \overline{\ell(l)}_1 \lor \ell(r)_1$ and assert $\ell(r)_1 = 1/\mathsf{u}$
if $\ell(r)$ is not pebbled then
    Propagate $\ell(r)_2 = 1/\ell(p)_1 \lor \ell(p)_2 \lor \overline{\ell(l)}_1 \lor \ell(l)_2 \lor \ell(r)_1 \lor \overline{\ell(r)}_2$
    QuickVisit ($\mathcal{T}_r$, $b = 1$)
```

```
Procedure BinaryPebble($\mathcal{T}$)
  Input: a robust binary tree pebbling $\mathcal{T}$ with root $p$.
1 Decide $\ell(p)_1 = 0/\mathsf{d}$
2 Propagate $\ell(p)_2 = 0/\ell(p)_1 \lor \overline{\ell(p)_2}$
3 DeepVisit ($\mathcal{T}$)
4 Learn $\ell(p)_1$ and assert $\ell(p)_1 = 1/\mathsf{u}$
5 QuickVisit ($\mathcal{T}$, $b = 1$)
```

□

This proves that there are at most 6 times more pebbling clauses than alive nodes in memory. The remaining learned clauses in the database are used for unit propagation. `QuickVisit` involves a constant number of clauses, and propagations in `DeepVisit` only occur because of pebble clauses, which we already accounted for, and pebbling axioms, which are not learned. Putting all these observations together we have that during the CDCL procedure we remember O(1) clauses per alive node, plus an additional O(1) clauses. This amounts to space O(*space*($\mathcal{T}$)).

To conclude the proof of Lemma D.5.14 we have to show that the trace is legal. Before a call to a visit procedure of $\mathcal{T}'$, the following invariants hold.

Claim D.5.17. *The trail assigns all nodes in the path from the root of $\mathcal{T}$ to p to even, and possibly their left siblings to odd. No other node is assigned.*

Claim D.5.18. *No other vertex labelled in $\mathcal{T}'$ has a variable assigned in the trail.*

Proof. The nodes that are assigned in the trail do not appear in $\mathcal{T}'$ by conditions 2 and 3 of Definition D.5.10. □

It is straightforward to check the propagations and conflicts listed in `QuickVisit`, and in Claim D.5.20 we will show that these are the only that happen.

At the last conflict all variables below $\ell(p)_2$ are set by unit propagation. If $\ell(p)_2 = b/\mathsf{d}$ is a decision, since the conflicting clause $\ell(p)_1^{1-b} \lor \ell(p)_2^{1-b} \lor \overline{\ell(l)_1} \lor \ell(l)_2 \lor \overline{\ell(r)_1} \lor \ell(r)_2$ contains the decision variables $\ell(p)_1$ and $\ell(p)_2$, they appear in any cut and therefore any conflict clause. The remaining variables in the conflict graph are of last decision level, so they cannot appear in an asserting clause because $\ell(p)_2$ already appears. Therefore we learn the clause $\ell(p)_1^{1-b} \lor \ell(p)_2^{1-b}$ and assert $\ell(p)_2 = 1 - b/\mathsf{u}$. If $\ell(p)_2$ is a propagation we backjump to an earlier point.

In Case 2 we also learn clauses $\overline{\ell(l)_1} \lor \overline{\ell(l)_2}$ and $\ell(l)_1 \lor \ell(l)_2$, so at the end of `QuickVisit` the vertex $\ell(l)$ is pebbled. Analogously, the vertex $\ell(r)$ is pebbled in Case 3.

It is also easy to verify that the propagations and conflicts in `DeepVisit` are correct. After line 14 both $\mathcal{T}_l$ and $\mathcal{T}_r$ have been visited, and both l and r are alive nodes when we visit p. This means that we store pebble clauses for their four immediate predecessors in the database, by induction hypothesis, therefore the subsequent calls to `QuickVisit` are legal. At the end of the procedure we have the pebble clauses that we claim in D.5.15 in the database. We learn the pebble clauses for $\ell(l)$ in lines 8 and 25, and we learn the pebble clauses for the children of $\ell(r)$ in the call to `QuickVisit` at line 18.

There is a detail that we have swept under the carpet up to this point. We claimed that the base case of `DeepVisit` is `QuickVisit`, but this is only the case if leaves are pebbled; otherwise `DeepVisit` would try to access an empty tree.

Claim D.5.19. *When we query whether a leaf is pebbled, the answer is yes.*

Proof. Leaves labelled by source vertices are always pebbled. For a non-source leaf, we only query its state when its cache node is alive, but there is some time during which a node is alive and not pebbled. Consider a cache node q. We need to distinguish three cases.

1. If q is the left child of a node p_j, then q will have a pebble at time j. During that time we visit the right subtree of p_j, but by condition 2 of Definition D.5.10 $\ell(q)$ does not appear at all in that subtree.

2. If q is the right child of a node l, which is the left child of a node p_j, then q will have a pebble at time j. During that time we visit the right subtree of p_j, but by condition 3 of Definition D.5.10 $\ell(q)$ does not appear as a leaf in that subtree.

3. If q is the right child of a node r, which is the right child of a node p_j, then q will have a pebble at time j and we will not visit any subtree during that time.

$\square$

Finally we show that no other unit propagation or conflict occurs.

Claim D.5.20. *All possible unit propagations and conflicts are listed in* `QuickVisit` *and* `DeepVisit`.

Proof. First we claim that all propagations and conflicts caused by pebble clauses, both learned and source axioms, are accounted for. Indeed, whenever a variable x_u is set, the next step depends on whether u has a pebble. The remaining learned clauses are erased as soon as they stop unit propagating, so they do not pose a problem either.

Regarding pebbling axioms, we show that when we visit a node p, only those axioms with support in p and its predecessors cause conflicts. Consider an axiom in $x_u \oplus y_u = 1 \wedge x_v \oplus y_v = 1 \rightarrow x_w \oplus y_w = 1$ such that two of its nodes are set and do not satisfy the axiom. We distinguish two cases.

1. u and v are set to odd.

2. u (or v) is set to odd and w to even.

In Case 1, by Claim D.5.17 and condition 2 of Definition D.5.10, u and v are not the predecessors of any node in the current subtree. Therefore, w is not set and no propagation happens.

In Case 2, since w is set to even, it is not pebbled, so by Claim D.5.19 it is an internal node. By Claim D.5.17, the only reason v is not set is that it is the node that we are currently visiting. Therefore, we account for propagations and conflicts. □

D.5.5 CDCL Trade-offs for Pebbling Formulas

We obtain trade-offs for CDCL refutations from trade-offs between pebbling time and pebbling space for DAGs. In particular our CDCL trade-offs will come from pebbling trade-offs for (variants of) Carlson-Savage graphs [52, 53], stacks of superconcentrators [130] and bit reversal graphs [130]. The upper bounds in this section apply to all CDCL systems regardless of the learning scheme, as long as it is cutting, and without restarts. The lower bounds apply to any CDCL system, regardless of the learning scheme and of the restart policy.

Lemma D.5.21. *Let G be some directed acyclic graph with indegree at most 2.*

1. *If G has a binary tree pebbling $\mathcal{T}$ then $Peb^{\oplus}_{r(G)}$ has a CDCL refutation without restarts, with any cutting learning scheme of time $\mathrm{O}(|\mathcal{T}|)$ and space $\mathrm{O}(space(\mathcal{T}))$*
2. *Given any CDCL refutation of $Peb^{\oplus}_{r(G)}$ of time τ and space s, we can extract a black-white pebbling for G of time $\mathrm{O}(\tau)$ and space $\mathrm{O}(s)$.*

Proof. To prove item 1 we use Lemma D.5.13 to get a robust binary tree pebbling $r(\mathcal{T})$ for $r(G)$ of time $\mathrm{O}(|\mathcal{T}|)$ and space $\mathrm{O}(space(\mathcal{T}))$. Then Lemma D.5.14 immediately gives the CDCL refutation $Peb^{\oplus}_{r(G)}$ of time $\mathrm{O}(|\mathcal{T}|)$ and space $\mathrm{O}(space(\mathcal{T}))$. To prove item 2 we observe that a CDCL refutation is indeed a resolution refutation, therefore we can use Theorem D.5.2 to get a black-white pebbling for graph $r(G)$ of time $\mathrm{O}(\tau)$ and space $\mathrm{O}(s)$. Using Proposition D.5.12 we obtain a black-white pebbling of G of time $\mathrm{O}(\tau)$ and space $\mathrm{O}(s)$ as well. □

The first trade-off comes from *stacks of superconcentrators*, a graph family originally defined in [130].

A DAG is a *superconcentrator* if it has m sources $S = \{s_1, \ldots, s_m\}$, m sinks $Z = \{z_1, \ldots, z_m\}$, and for any subsets $S' \subseteq S$ and $Z' \subseteq Z$ with $|S'| = |Z'|$ there are $|S'|$ vertex disjoint paths, each connecting one source in S' to some sink in Z'.

In literature there are efficient constructions of superconcentrators of indegree 2 with m sources and sinks, $\mathrm{O}(m)$ edges and depth $\mathrm{O}(\log m)$, for every $m = C \cdot 2^k$ where C is a fixed value depending of the construction and k is a free parameter (see for example [5]).

Let $\mathrm{SC}^{(1)}_m, \ldots, \mathrm{SC}^{(r)}_m$ denote r copies of an efficiently constructible superconcentrator with m sources and sinks, with $m = \Theta(2^k)$ for some $k > 0$, indegree 2, $\mathrm{O}(m)$ edges and depth $O(\log m)$. The graph $\Phi(m,r)$ is constructed by placing these r copies on top of one another, i.e., with the sinks of $z_1^j, z_2^j, \ldots, z_m^j$ of $\mathrm{SC}_m^{(j)}$ connected to the sources $s_1^{j+1}, s_2^{j+1}, \ldots, s_m^{j+1}$ of $\mathrm{SC}_m^{(j+1)}$ with edges (z_i^j, s_i^{j+1}) for $i = 1, \ldots, m$ and $j = 1, \ldots, r-1$.

Theorem D.5.22 (Trade-off for stack of superconcentrators). *There exists an efficiently constructible family of* 3*-CNF formulas F_n of size $\Theta(n)$ that have*

- *a CDCL refutation without restarts and with any cutting learning scheme in time* $O(n)$ *and space* $O(n/\log n)$ *simultaneously;*
- *a CDCL refutation without restarts and with any cutting learning scheme in space* $O((\log n)^2)$ *and time* $n^{O(\log n)}$ *simultaneously.*

Moreover there exists a constant $K > 0$ such that any CDCL refutation in space $s \le Kn/\log n$ satisfies that time τ is at least $n^{\Omega(\log\log n)}$, regardless of the learning scheme and of the restart policy.

Proof. The formula family F_n for which we prove this theorem is $Peb_{G_n}^{\oplus}$, where we build the graph G_n as follows. We first start with a stack of super concentrators $\Phi(m,r)$, where we set the parameters $m = \Theta(n/\log n)$ and $r = \Theta(\log n)$, and then we add an additional binary tree on top to make the graph have a single sink. More specifically we put on top a complete binary tree of depth $\log m$ where the sources are denoted $s_1^{r+1}, s_2^{r+1}, \ldots, s_m^{r+1}$ and the sink as z_1^{r+1}. The sinks of the topmost layer of the superconcentrator are connected to this tree through edges (z_i^r, s_i^{r+1}). Let us denote this graph as $\widehat{\Phi}(m,r)$ for the rest of the proof.

In [130] it is shown that any black-white pebbling for $\Phi(m,r)$ that has space $s \le m/20$ requires time $\tau \ge m \cdot \left(\frac{rm}{64s}\right)^r$. Since any black-white pebbling for $\widehat{\Phi}(m,r)$ induces a black-white pebbling for $\Phi(m,r)$ the result holds for $\widehat{\Phi}(m,r)$ too. In particular this means that there is a constant K such that any black-white pebbling with space at most $Kn/\log n$ takes time at least $n^{\Omega(\log\log n)}$.

We show two binary tree pebblings for $\widehat{\Phi}(m,r)$: a time efficient one, with simultaneous time $O(mr) = O(n)$ and space $O(m) = O(n/\log n)$, and a space efficient one, with simultaneous time $m^{O(r)} = n^{O(\log n)}$ and space $O(r\log m) = O((\log n)^2)$. After building these binary tree pebblings we get the theorem by setting G_n to be $r(\widehat{\Phi}(m,r))$ and by using Lemma D.5.21 on graph $\widehat{\Phi}(m,r)$.

The space efficient construction is immediate since the depth of $\widehat{\Phi}(m,r)$ is $O(r\log m)$ and therefore the graph has a binary tree pebbling of the same depth by Proposition D.5.9. The time efficient binary tree pebbling $\mathcal{T}$ is the depth-first pebbling from Proposition D.5.5, which has time $O(mr)$. The pebbling is induced by a depth first search, so there is at most one internal node labelled by each vertex. The rest of the proof consists of showing that this binary tree pebbling has space $O(m)$.

Consider, in the left postfix order of $\mathcal{T}$, the interval between the first occurrence of a sink of layer j, and a sink of layer $j+1$, say z_i^{j+1}. Since all sources of layer j have already

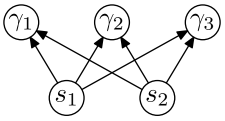


Figure D.10: Base case $\Gamma(3,1)$ for Carlson-Savage graph with 3 sinks.

been visited, no node with a label in layer $j-1$ is alive. Since z_i^{j+1} is the first node in layer $j+1$, no node with label in layer $j+2$ or above has been visited and therefore cannot be alive. Therefore the number of alive nodes is bounded by the number of nodes in the binary tree with labels in layers j and $j+1$, that is $O(m)$. □

Before moving to the next graph family we need to describe one of its components, the pyramid graph, for which we also need some properties.

Definition D.5.23. The pyramid graph of height h (denoted as Π_h) is a graph over $(h+1)(h+2)/2$ vertices, indexed as (i,j) for $0 \le i \le j \le h$. For $i > 0$ each vertex (i,j) has two incoming edges from vertices $(i-1,j-1)$ and $(i-1,j)$. The sink vertex is (h,h).

The black-white pebbling price of a pyramid of height h is $h/2 + \Theta(1)$ [123].

Proposition D.5.24. *There is a binary tree pebbling for the pyramid of height $h \ge 1$ of time $O(h^2)$ and space $h+2$.*

Proof. We build the tree by induction over h. The case $h = 0$ is immediate. For $h > 0$ consider the subgraph of the pyramid induced by the vertices (i,j) with $j \ne h$. This is a pyramid graph of height $h-1$ with sink $(h-1,h-1)$. Consider the tree $\mathcal{T}'$ of this pyramid. The tree $\mathcal{T}$ for Π_h has root node p_h labelled by (h,h), its left subtree is a copy of $\mathcal{T}'$ and its right subtree is made by nodes $p_{h-1},\dots,p_0$ and leaf nodes $q_{h-2},\dots,q_0$, where p_i is labelled by (i,h) and q_i is labelled by $(i,h-1)$, and for $i > 0$ the left and right child of p_i are, respectively, q_{i-1} and p_{i-1}.

There is a bijection between the edges of $\mathcal{T}$ and the edges of Π_h, therefore the size of $\mathcal{T}$ is $O(h^2)$. To see that the cost of the tree is $h+2$ observe that in the left postfix order visit of $\mathcal{T}$, whenever the (unique) internal node labelled by (i,j) is visited it will be a cache node only until the visit of $(i+1,j+1)$, therefore the worst case is at the visit of the (unique) node of $\mathcal{T}$ labelled by $(1,h)$. In that moment the alive nodes are internal nodes with labels $(1,h)$, $(i,h-1)$ for $0 \le i \le h-1$ and the leaf labelled by $(0,h)$. □

The next trade-off result is based on Carlson and Savage graphs [52, 53]. We use a slight adaptation of this construction, more suitable for proof complexity results, due to [144]. The definition of this family of graphs is rather intricate, so we suggest that the reader refers to the illustrations in Figures D.10 and D.11.

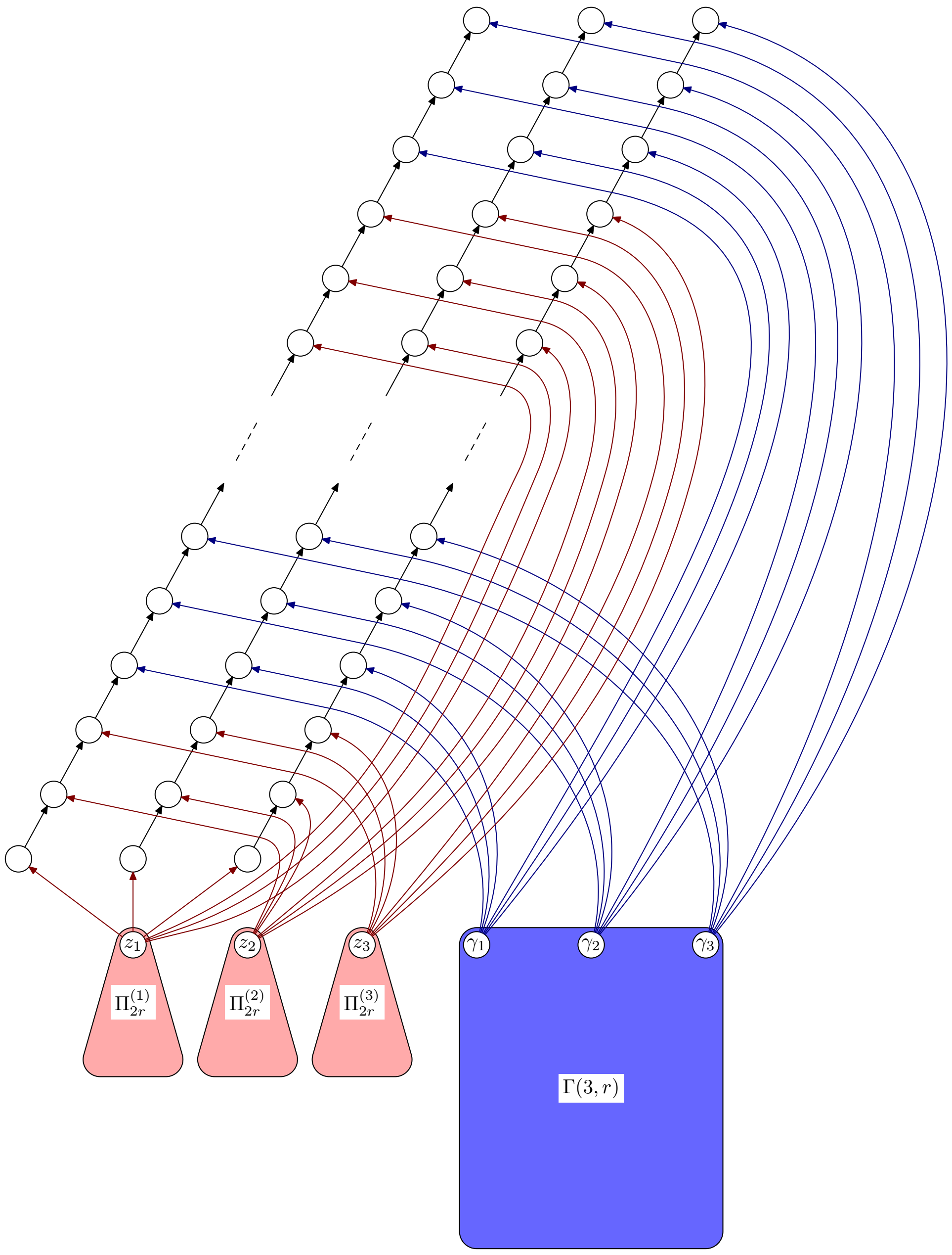


Figure D.11: Inductive definition of Carlson-Savage graph $\Gamma(3, r + 1)$ with 3 spines and sinks.

Definition D.5.25 (Carlson-Savage graphs [52, 53]). For positive integers c, r, the graph family $\Gamma(c, r)$, is defined by induction over r. The base case $\Gamma(c, 1)$ is a DAG consisting of two sources s_1, s_2 and c sinks $\gamma_1, \dots, \gamma_c$, with edges from both sources to all sinks. The graph $\Gamma(c, r+1)$ has c sinks and is built from the following components:

- c disjoint copies $\Pi^{(1)}_{2r}, \dots, \Pi^{(c)}_{2r}$ of a pyramid (Definition D.5.23) of height $2r$, where we let $z_1, \dots, z_c$ denote the pyramid sinks;
- one copy of $\Gamma(c, r)$, for which we denote the sinks by $\gamma_1, \dots, \gamma_c$;
- c disjoint and identical *spines* of length $\lambda = 2c^2 r$. Each spine is a sequence of λ vertices, denoted as $v_1, \dots, v_\lambda$, connected by edges $(v_\ell, v_{\ell+1})$ for $\ell = 1, \dots, \lambda - 1$.

These components are connected through edges as follows.

- There are edges $\left(z_j, v_\ell\right)$ to any vertex v_ℓ where $\ell = 2c\ell' + j$ for some integer ℓ'.
- There are edges $\left(\gamma_j, v_\ell\right)$ to any vertex v_ℓ where $\ell = 2c\ell' + c + j$ for some integer ℓ'.

Pebbling games on the Carlson-Savage graphs have strong time-space tradeoffs. Specifically, in [144], it was shown that for any black-white pebbling of $\Gamma(c, r)$ in space s and time τ, if $s \leq r + k$ for some $0 \leq k \leq c/8$, then

$$\tau \geq \left(\frac{c - 2k}{4k + 4}\right)^r r! \ . \qquad \text{(D.5.2)}$$

Theorem D.5.26 (Trade-off for arbitrarily slowly growing space). *Let $g(n) = \omega(1)$ be any arbitrarily slowly growing monotone function such that $g(n) = \mathrm{O}(n^{1/7})$ and fix any $\epsilon > 0$. There exists an efficiently constructible family of 3-CNF formulas F_n of size $\Theta(n)$ that have*

- *a CDCL refutation without restarts and with any cutting learning scheme in time $\mathrm{O}(n)$ and space $\mathrm{O}\big(\sqrt[3]{n/g^2(n)}\big)$ simultaneously;*
- *a CDCL refutation without restarts and with any cutting learning scheme in space $\mathrm{O}(g(n))$ and time $n^{\mathrm{O}(g(n))}$ simultaneously.*

Moreover any CDCL refutation of space $\mathrm{O}\big(\left(n/g^2(n)\right)^{1/3-\epsilon}\big)$ has superpolynomial time, regardless of the learning scheme and of the restart policy.

Picking an appropriate function g, in this case $g(n) = \sqrt[8]{n}$, the trade-off is even exponential.

Corollary D.5.27. *There exists an efficiently constructible family of 3-CNF formulas F_n of size $\Theta(n)$ that have*

- *a CDCL refutation without restarts and with any cutting learning scheme in time $\mathrm{O}(n)$ and space $\mathrm{O}\big(\sqrt[4]{n}\big)$ simultaneously;*

- *a CDCL refutation without restarts and with any cutting learning scheme in space* $\mathrm{O}\big(\sqrt[8]{n}\big)$ *and time* $n^{\mathrm{O}(\sqrt[8]{n})}$ *simultaneously.*

Moreover, for any $\epsilon > 0$*, any CDCL refutation of space* $\mathrm{O}\big(n^{1/4-\epsilon}\big)$ *has exponential time, regardless of the learning scheme and of the restart policy.*

Proof of Theorem D.5.26. The formula is based on the single sink version of the graph $\Gamma(c,r)$ from Definition D.5.25, which is the graph with a maximally unbalanced binary tree on top. This graph is denoted as $\widehat{\Gamma}(c,r)$ and it is formally defined as follows. Let $\gamma_1,\ldots,\gamma_c$ be the sinks of $\Gamma(c,r)$; we add new vertices $\eta_2,\ldots,\eta_c$, two edges (γ_1,η_2), (γ_2,η_2) and then we add edge pairs (η_{j-1},η_j), (γ_j,η_j) for j from 3 to c. Hence $\widehat{\Gamma}(c,r)$ has a unique sink vertex η_c, it contains $\Gamma(c,r)$ as an induced subgraph, and it has $\Theta(cr^3+c^3r^2)$ vertices. The main part of the proof is to show that

- there is a binary tree pebbling for $\widehat{\Gamma}(c,r)$ of space $2r+1+3c$ and time $\Theta(cr^3+c^3r^2)$;
- there is a binary tree pebbling for $\widehat{\Gamma}(c,r)$ of space $2r+2$;

and that the inequality claimed in [144] (see Equation (D.5.2)) holds for $\widehat{\Gamma}(c,r)$ as well. Regarding (D.5.2), we can just observe that any black-white pebbling of η_c induces a black-white pebbling of $\Gamma(c,r)$ within at most the same time and space.

To get the statement of the theorem we are going to apply Lemma D.5.21 to $\widehat{\Gamma}(c,r)$, where we set r as $r(n)=g(n)$ and c as $c(n)=\sqrt[3]{n/g^2(n)}$, and we are going to set G_n to be $r(\widehat{\Gamma}(c(n),r(n)))$. The size of the resulting formula $\mathit{Peb}^{\oplus}_{G_n}=\mathit{Peb}^{\oplus}_{r(\widehat{\Gamma}(c(n),r(n)))}$ is $\Theta(cr^3+c^3r^2)=\Theta(n)$, the space efficient CDCL refutation has space $\mathrm{O}(g(n))$ and the time efficient CDCL refutation has linear time and space $\mathrm{O}\big(\sqrt[3]{n/g^2(n)}\big)$, for $g(n)=\mathrm{O}(n^{1/7})$. Now let's fix $k=c^{1-\epsilon}$ in (D.5.2) for some $\epsilon>0$ so that the space of the black-white pebbling extracted by the refutation of $\mathit{Peb}^{\oplus}_{r(\widehat{\Gamma}(c(n),r(n)))}$, which is within a linear factor from k, is less than $c/8$ for n large enough. Equation (D.5.2) and Lemma D.5.21 imply that any CDCL refutation with space $\mathrm{O}\Big(\big(\sqrt[3]{n/g^2(n)}\big)^{1-\epsilon}\Big)$ has time $\big(n/g^2(n)\big)^{\Omega(g(n))}$, which is superpolynomial since $g(n)=\omega(1)$.

Now it remains to build the space efficient tree $\widehat{\mathcal{T}}_{(\mathrm{s},r)}$ and the time efficient tree $\widehat{\mathcal{T}}_{(\tau,r)}$ for $\widehat{\Gamma}(c,r)$. As intermediate step we describe, by induction on r, two binary tree pebblings for the subgraph of $\Gamma(c,r)$ constituted by all vertices from which a sink γ_j is reachable. The space efficient version of this intermediate construction is denoted as $\mathcal{T}_{(\mathrm{s},r)}$, and the time efficient one as $\mathcal{T}_{(\tau,r)}$. Since $\Gamma(c,r)$ is symmetric with respect to the permutation of its sink we will discuss the construction of $\mathcal{T}_{(\mathrm{s},r)}$ and $\mathcal{T}_{(\tau,r)}$ with respect of a generic sink.

The two base cases $\mathcal{T}_{(\mathrm{s},1)}$ and $\mathcal{T}_{(\tau,1)}$ are the same. They have the root labelled by γ' and its two children labelled by sources s_1 and s_2.

For the inductive case we consider an arbitrary sink γ' of $\Gamma(c,r+1)$, and we build the two trees $\mathcal{T}_{(\mathrm{s},r+1)}$ and $\mathcal{T}_{(\tau,r+1)}$ for the subgraph of all vertices of $\Gamma(c,r+1)$ that can reach γ'. We suggest to refer to Figure D.11 to follow the argument. We denote as

$(v_1,\dots,v_\lambda)$ the sequence of the vertices in the spine that ends with sink γ', so that z_1 is a predecessor of v_1 and v_λ is γ' itself.

We define the tree $\mathcal{T}_{(*,r+1)}$ so that its leftmost path backward from the root has length λ and is labelled by the sequence of vertices $(v_\lambda,\dots,v_1)$. The only child of the vertex labelled by v_1 is a leaf vertex labelled by z_1. For $1<\ell\leq\lambda$ the right child of the node labelled by v_ℓ is a leaf node labelled by the corresponding predecessor of v_ℓ among $\{z_1,\dots,z_c,\gamma_1,\dots,\gamma_c\}$. For the sake of uniformity we give the definition of $\mathcal{T}_{(*,1)}$ too, which is identical to $\mathcal{T}_{(\tau,1)}$ and to $\mathcal{T}_{(s,1)}$. For $1\leq j\leq c$ we denote as $\mathcal{T}^{\Pi}_j$ the binary tree pebbling for the pyramid subgraph $\Pi^{(j)}_{2r}$ given in Proposition D.5.24, which has time $O(r^2)$ and space $2r+2$. Both the $\mathcal{T}_{(s,r)}$ and $\mathcal{T}_{(\tau,r)}$ constructions have $\mathcal{T}_{(*,r)}$ as a starting point, and the difference in the two constructions is in the subtrees that we attach to their leaves.[4]

The space efficient construction $\mathcal{T}_{(s,r+1)}$ from $\mathcal{T}_{(s,r)}$. We start with $\mathcal{T}_{(*,r+1)}$ and we attach to each leaf labelled z_j a copy of $\mathcal{T}^{\Pi}_j$. For each leaf labelled γ_j we attach a copy of the space efficient binary tree pebbling $\mathcal{T}_{(s,r)}$ obtained by inductive hypothesis, namely the space efficient tree for the subgraph of $\Gamma(c,r)$ constituted by all vertices that can reach γ_j. We claim by induction on r that this construction has space at most $2r+1$. For $r=1$ the space is 3 so we are within the limits. For $r>1$ observe that when we visit the right subtree of a node labelled by some vertex v_ℓ, the only node alive outside of that subtree is the one labelled by $v_{\ell-1}$. The space used by the total construction then is one plus the maximum space among all subtrees attached to $\mathcal{T}_{(*,r+1)}$. The space for each copy of $\mathcal{T}^{\Pi}_j$ is $2r+2$, and by induction the space for each copy of $\mathcal{T}_{(s,r)}$ is $2r+1$. Therefore the space for $\mathcal{T}_{(s,r+1)}$ is $2r+3$, and the inductive claim is proved.

The space efficient construction $\widehat{\mathcal{T}}_{(s,r)}$ from $\mathcal{T}_{(s,r)}$. Now we can build the space efficient binary tree of $\widehat{\Gamma}(c,r)$. The top part of the tree is an isomorphic copy of the tree in $\widehat{\Gamma}(c,r)$ that connects the sinks $\gamma_1,\dots,\gamma_c$ of $\Gamma(c,r)$ to the new sink η_c, where nodes are labelled by the corresponding vertices. To each node labelled by γ_j we attach a new copy of the space efficient tree $\mathcal{T}_{(s,r)}$ for the corresponding sink. The final tree is a binary tree pebbling for $\widehat{\Gamma}(c,r)$, and it has space $2r+2$, since one space unit is used for the top part and each copy of $\mathcal{T}_{(s,r)}$ has space $2r+1$.

The time efficient construction $\mathcal{T}_{(\tau,r+1)}$ from $\mathcal{T}_{(\tau,r)}$. In the time efficient construction we only attach subtrees to just the first few leaves in $\mathcal{T}_{(*,r+1)}$ so that later leaves remain leaves in the final construction too, and use them as cache nodes. This makes the tree smaller, but increases the space of the construction.

We consider the sequence of nodes $(p'_1,\dots,p'_c,q_1,\dots,q_c,p_1,\dots,p_c)$ from $\mathcal{T}_{(*,r+1)}$ so that, for each $j\in[c]$, node p'_j is the child labelled by z_j of the node labelled by v_j, node q_j is the child labelled by γ_j of the node labelled by v_{c+j} and node p_j is the child labelled by z_j of the node labelled by v_{2c+j}.

[4]Here we use the verb *to attach* in a very specific way. Given a leaf node p labelled by some vertex v and a binary tree pebbling $\mathcal{T}$ with the root labelled by the same vertex v, we say that we attach $\mathcal{T}$ to the node p by substituting p with $\mathcal{T}$ itself.

We will attach a subtree only to those $3c$ nodes. Any later occurrence of a leaf node labelled by z_j will use p_j as cache node, and any later occurrence of a leaf labelled by γ_j will use q_j as cache node.

We need to stress that $p'_1, \ldots, p'_c$ are not going to be cache nodes. This choice is made because the left postorder needs to do a full visit of $\mathcal{T}_{(\tau,r)}$ between the visit of $p'_1, \ldots, p'_c$ and the visit of any later node labelled by some z_j. If such nodes were cached during the recursive descent, the space of the tree would explode.

For this reason we are going to visit each pyramid $\Pi^{(j)}_{2r}$ twice, namely for $j \in [c]$ we attach the tree $\mathcal{T}^{\Pi}_j$ to both p'_j and p_j. In this way p_j is the last internal node to have label z_j, and it is going to be the cache node for all later nodes with that label. To node q_1 instead we attach a copy of the tree $\mathcal{T}_{(\tau,r)}$ built by induction, and for $2 \leq j \leq c$ we attach to node q_j a copy of $\mathcal{T}_{(*,r)}$, built by induction for the sink γ_j.

The tree built so far is the final tree $\mathcal{T}_{(\tau,r+1)}$, and since it is more complex than the space efficient one we will explicitly show its correctness.

To do that we need notation to refer to vertices in the subgraph $\Gamma(c,r)$ contained in $\Gamma(c,r+1)$, and notation to refer to nodes in the subtree $\mathcal{T}_{(\tau,r)}$ inside $\mathcal{T}_{(\tau,r+1)}$. We recall that γ' denotes our target sink in the graph $\Gamma(c,r+1)$, and that we denoted as z_j the sinks of the pyramids and as γ_j the sinks of the copy of $\Gamma(c,r)$ contained in the recursive construction $\Gamma(c,r+1)$. When $r > 1$ we denote as $z^{\dagger}_1, \ldots, z^{\dagger}_c$ the sinks of the pyramids (of height $2r-2$) inside the subgraph of $\Gamma(c,r)$, and as $\gamma^{\dagger}_1, \ldots, \gamma^{\dagger}_c$ the sinks of its internal copy of $\Gamma(c,r-1)$. Furthermore we denote as $p^{\dagger}_j$ the (unique) cache node in $\mathcal{T}_{(\tau,r)}$ labelled by $z^{\dagger}_j$, and as $q^{\dagger}_j$ the (unique) cache node in $\mathcal{T}_{(\tau,r)}$ labelled by $\gamma^{\dagger}_j$.

Let us now argue correctness.

Claim D.5.28. *For each leaf node in $\mathcal{T}_{(\tau,r)}$ which is not labelled by a source vertex of $\Gamma(c,r)$ there is a corresponding cache node earlier in the tree.*

Proof. Our proof is by induction on r. For $r = 1$ it is immediate. Let us assume that it is true for $r \geq 1$, we want to prove that each leaf node in $\mathcal{T}_{(\tau,r+1)}$ which is not labelled by a source vertex of $\Gamma(c,r+1)$ there is a corresponding cache node earlier in the tree. Subtrees $\mathcal{T}^{\Pi}_j$ are correct by construction, and the correctness of subtree $\mathcal{T}_{(\tau,r)}$ holds by induction. The remaining non-source leaves in $\mathcal{T}_{(\tau,r+1)}$ are either the ones coming from $\mathcal{T}_{(*,r+1)}$ to which we did not attach a subtree, or they are non-source leaf nodes inside the $(c-1)$ copies of $\mathcal{T}_{(*,r)}$ attached to $p_2, \ldots, p_c$. In the first case the leaf is either labelled by γ_j or by z_j and it comes later in the left postfix order than nodes $q_1, \ldots, q_c, p_1, \ldots, p_c$, one of them being its cache node. For $r = 1$ the leaves of all attached copies of $\mathcal{T}_{(*,r)}$ are all labelled by sources s_1 and s_2. For $r > 1$ the leaves of all attached copies of $\mathcal{T}_{(*,r)}$ are all labelled by vertices among $\gamma^{\dagger}_1, \ldots, \gamma^{\dagger}_c, z^{\dagger}_1, \ldots, z^{\dagger}_c$, and earlier in the left postfix order, $\mathcal{T}_{(\tau,r+1)}$ contains the subtree $\mathcal{T}_{(\tau,r)}$, which contains nodes $q^{\dagger}_1, \ldots, q^{\dagger}_c, p^{\dagger}_1, \ldots, p^{\dagger}_c$ with the same labels. ☐

The size of the tree is estimated in the following claim.

Claim D.5.29. *The number of internal vertices in $\mathcal{T}_{(\tau,r)}$ plus $(c-1)$ copies of $\mathcal{T}_{(*,r)}$ is at most C times the number of edges in $\Gamma(c,r)$, for some universal constant $C>0$.*

Proof. Our proof is again by induction on r. It is immediate for $r=1$. Assume that the claim holds for $r \geq 1$, we prove it for $r+1$. Observe that $\mathcal{T}_{(\tau,r+1)}$ contains a copy of $\mathcal{T}_{(\tau,r)}$ ending in γ_1 plus $(c-1)$ copies of $\mathcal{T}_{(*,r)}$ ending in $\gamma_2,\ldots,\gamma_c$, respectively. By inductive hypothesis this accounts to at most C times the size of $\Gamma(c,r)$. Furthermore $\mathcal{T}_{(\tau,r+1)}$ contains one isomorphic copy of one spine from $\Gamma(c,r+1)$, while the other $(c-1)$ spines in $\Gamma(c,r+1)$ are accounted to the edges of the $(c-1)$ copies of $\mathcal{T}_{(*,r+1)}$ as in the claim. Tree $\mathcal{T}_{(\tau,r+1)}$ also contains $2c$ binary tree pebblings for pyramid graphs of height $2r$, each of size at most C' times the size of the pyramid graph itself, for a universal constant C' (see Proposition D.5.24). Let C be a universal constant larger than $\max\{2C',1\}$ and then the induction step follows. □

We claim that the space of $\mathcal{T}_{(\tau,r)}$ is at most $2r+1+3c$. The space is 3 for $r=1$ so we are within the bound. For the general claim we use the following inductive statement.

Claim D.5.30. *During each moment of the visit of $\mathcal{T}_{(\tau,r+1)}$, the set of alive nodes union the set of nodes in $\{q_1,\ldots,q_c,p_1,\ldots,p_c\}$ already visited at that moment has size at most $2r+3+3c$.*

Proof. We assume that either $r=1$ or that the claim holds for $\mathcal{T}_{(\tau,r)}$ if $r>1$.

Observe that the left postfix order visits the sequence of nodes labelled v_ℓ in order. We divide the sequence in four segments: the first three have length c each, and the last one is made by the rest of the sequence.

First segment. At the beginning we visit a copy of $\mathcal{T}^{\Pi}_1$ using space $2r+2$, then the node labelled by v_1. At this point no node in this copy of $\mathcal{T}^{\Pi}_1$ is alive anymore. The next step is to visit the copy of $\mathcal{T}^{\Pi}_2$ rooted in p'_2, and then the node labelled by v_2, and so on up to $\mathcal{T}^{\Pi}_c$. During the visit of some $\mathcal{T}^{\Pi}_j$, the only other alive node is the one labelled by v_{j-1}. At some point we reach the node labelled by v_c using the maximum space $2r+3$, and that is the only alive node at that moment.

Second segment (I). If $r=1$ then the visit of the subtree $\mathcal{T}_{(\tau,r)}$ is done in space 3. Otherwise we visit the subtree $\mathcal{T}_{(\tau,r)}$ in space $2r+1+3c$, passing through nodes $\{q^{\dagger}_1,\ldots,q^{\dagger}_c,p^{\dagger}_1,\ldots,p^{\dagger}_c\}$, that stay alive. The root of this tree is q_1, which is a cache node for later nodes. During the visit of the subtree $\mathcal{T}_{(\tau,r)}$ the node labelled by v_c is alive, therefore we have maximum space so far equal to $2r+2+3c$, and the alive nodes are: the node labelled by v_c, node q_1 and, when $r>1$, $\{q^{\dagger}_1,\ldots,q^{\dagger}_c,p^{\dagger}_1,\ldots,p^{\dagger}_c\}$.

Second segment (II). The visit now proceeds up to the node labelled by v_{2c}: we visit each of the $(c-1)$ copies of $\mathcal{T}_{(*,r)}$ in order. If $r=1$ then $\mathcal{T}_{(*,1)}$ is a node with two leaf children, so three additional alive nodes are necessary. If $r>1$ each of them is a maximally unbalanced tree with leaves labelled by $\{\gamma^{\dagger}_1,\ldots,\gamma^{\dagger}_c,z^{\dagger}_1,\ldots,z^{\dagger}_c\}$ for which we already have alive cache nodes, therefore only two additional alive nodes are necessary to complete this phase. During the visit of the copy of $\mathcal{T}_{(*,r)}$ rooted in q_j, the alives nodes are

$q_1, \ldots, q_{j-1}$ from previous copies; the nodes $\{q_1^\dagger, \ldots, q_c^\dagger, p_1^\dagger, \ldots, p_c^\dagger\}$ from recursion when $r > 1$; the node labelled by v_{c+j-1}; and at most two additional nodes during the visit itself (three if $r = 1$). The maximum space in this specific phase is at most $3 + 3c$, and after its conclusion the alive nodes are: the node labelled by v_{2c} and $q_1, \ldots, q_c$. In particular after this moment we will not visit any node for which any of $\{q_1^\dagger, \ldots, q_c^\dagger, p_1^\dagger, \ldots, p_c^\dagger\}$ is a cache node.

Third segment. In the next phase we visit another series of the binary tree pebbling for the pyramids, but this time their roots $p_1, \ldots, p_c$ are going to be cache nodes for later non-source leaf nodes. For each $j \in [c]$ we visit the copy of $\mathcal{T}_j^\Pi$ rooted in node p_j while nodes $q_1, \ldots, q_c$, nodes $p_1, \ldots, p_{j-1}$ and the node labelled by v_{2c+j-1} are alive. The total space in this phase is at most $2r + 3 + 2c$.

Fourth segment. For each $\ell > 3c$, we just visit v_ℓ in order, using cache nodes $\{q_1, \ldots, q_c, p_1, \ldots, p_c\}$, with two additional alive nodes simultaneously.

In total we have that the number of alive nodes is always at most $2r + 3 + 3c$, and furthermore the nodes $\{q_1, \ldots, q_c, p_1, \ldots, p_c\}$ are counted as alive from the moment they are visited, until the end of the visit, as requested by the claim. □

The time efficient construction $\widehat{\mathcal{T}}_{(\tau,r)}$ from $\mathcal{T}_{(\tau,r)}$. Now we can build the time efficient binary tree of $\widehat{\Gamma}(c, r)$. The top part of the tree is an isomorphic copy of the tree in $\widehat{\Gamma}(c, r)$ that connects the sinks $\gamma_1, \ldots, \gamma_c$ of $\Gamma(c, r)$ to the new sink η_c, where nodes are labelled by the corresponding vertices. To the node labelled by γ_1 we attach a copy of the time efficient tree $\mathcal{T}_{(\tau,r)}$, while to each other node labelled by γ_j with $j > 1$ we attach a copy of $\mathcal{T}_{(*,r)}$. By Claim D.5.29 the time of $\widehat{\mathcal{T}}_{(\tau,r)}$ is $O(c)$ plus a constant times the size of $\Gamma(c, r)$, which is $O(cr^3 + c^3r^2)$ as desired. The left postfix visit of $\widehat{\mathcal{T}}_{(\tau,r)}$ is the composition of the visit of $\mathcal{T}_{(\tau,r)}$ plus the visit of the rest of the tree, which uses the $2c$ cache nodes q_j and p_j from $\mathcal{T}_{(\tau,r)}$. The rest of the visit can be done with a constant number (i.e., 4) of additional alive nodes, so the space of $\widehat{\mathcal{T}}_{(\tau,r)}$ is dominated by the one for the visit of $\mathcal{T}_{(\tau,r)}$, and is $2r + 1 + 3c$. □

The last graph family we consider is the bit reversal graph from [130], which is made by two paths of vertices, where each vertex in the bottom path is connected to a vertex in the top path, according to a specific bijection (see Figure D.12).

Definition D.5.31 (Bit reversal graph). Fix $k > 0$ and let π be the permutation that maps each integer $0 \le i < 2^k$ written as its k bit binary representation into the number that corresponds to the reverse of that binary representation. The *bit-reversal graph* has $2 \cdot 2^k$ vertices, divided into two paths. The lower path is $u_0, \ldots, u_{2^k-1}$ where u_{i-1} is a predecessor of u_i for $1 \le i < 2^k$. The upper path is $v_0, \ldots, v_{2^k-1}$ where v_{i-1} is a predecessor of v_i for $1 \le i < 2^k$. For $i > 0$, each v_i has two predecessors, namely v_{i-1} and $u_{\pi(i)}$. The only predecessor of v_0 is u_0.

Theorem D.5.32 ([130]). *Fix $n = 2 \cdot 2^k$. Consider the bit reversal graph with n vertices and any black-white pebbling of time t and $s > 3$. Then it holds that $t \ge \frac{n^2}{18s^2} + 2n$.*

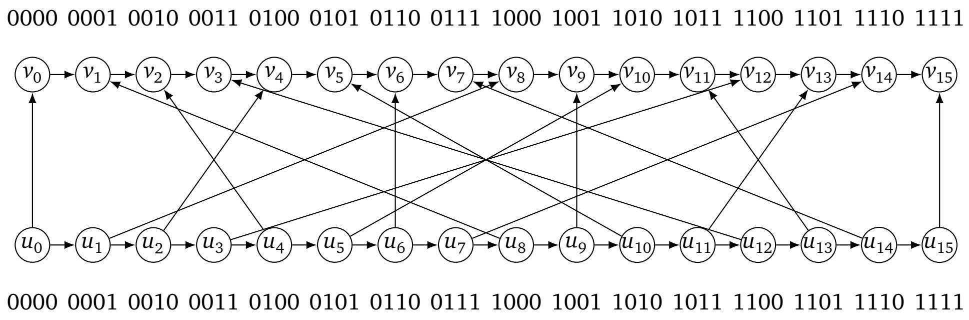

0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13} v_{14} v_{15}
u_0 u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9 u_{10} u_{11} u_{12} u_{13} u_{14} u_{15}
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

Figure D.12: Bit reversal graph with $k = 4$.

Theorem D.5.33 (Trade-off for pebbling formulas over bit reversal graphs). *There exists an efficiently constructible family of* 3*-CNF formulas F_k of size $n = \Theta(2^k)$ that have a CDCL refutation without restarts and with any cutting learning scheme in space* $O(s)$ *and time* $O(n^2/s)$ *simultaneously, for every $s = O(\sqrt{n})$. Moreover, any CDCL refutation simultaneously in space $s \leq \sqrt{n}$ and time τ satisfies*

$$\tau = \Omega\left(\left(\frac{n}{s}\right)^2\right), \tag{D.5.3}$$

regardless of the learning scheme and of the restart policy.

Proof. The formula family F_k for which we prove this theorem is $Peb^{\oplus}_{G_k}$, where the graph G_k is the robust version of the bit reversal graph in Definition D.5.31 with $2 \cdot 2^k$ vertices.

We will show that the bit reversal graph has a binary tree pebbling of space $O(s)$ and time $O(n^2/s)$. Since the appropriate trade-off for black-white pebbling also holds by Theorem D.5.32, the result then follows by Lemma D.5.21.

We start with a chain of nodes $p_0, \ldots, p_{2^k-1}$ labelled respectively $v_0, \ldots, v_{2^k-1}$, and connected so that for $0 < i < 2^k$, p_{i-1} is the left child of p_i. Each p_i has a right child q_i labelled by vertex $u_{\pi(i)}$. Let us further denote $\mathcal{T}_{i,j}$ for $0 \leq i \leq j < 2^k$ as a tree made by a single chain of $j - i + 1$ nodes labelled from the leaf to the root with vertices $u_i, \ldots, u_j$ respectively. Notice that $\mathcal{T}_{i,i}$ is a tree containing a single node.

A trivial construction would be to attach to each node q_i a tree $\mathcal{T}_{0,\pi(i)}$. This produces a binary tree pebbling for the bit reversal graph of time $O(n^2)$ and constant space. Another simple construction would be to attach at node q_i the tree $\mathcal{T}_{j,\pi(i)}$ where j is the index closest to $\pi(i)$ so that label u_j occur in one of the trees attached to some earlier $p_{i'}$ with $i' < i$. In this construction instead all internal nodes labelled by some u_j are indeed cache nodes. The space of this tree is $O(n)$ and its time is $O(n)$ as well. For our construction we do something in between of these two extremes: we divide the sequence $0, \ldots, 2^k - 1$ into s intervals of length $2^k/s$. Let L be the set of lower endpoints of the intervals. We use the nodes in L as cache nodes and go back to the closest cache node each time. Formally, we attach the tree $\mathcal{T}_{0,0}$ to p_0, and for each $0 < i < 2^k$, we attach $\mathcal{T}_{s_i,\pi(i)}$ to node p_i, where s_i is the closest cache node: if $t_i = \max_{j<i} \pi(j)$ is the rightmost vertex in the lower path visited so far, then $s_i = \max\{l \in L \mid l \le t_i\}$. Therefore, the total number of cache nodes is $O(s)$. For the time bound, observe that a length ℓ tree $\mathcal{T}_{i,i+\ell-1}$ adds $\ell/(2^k/s) - O(1)$ new cache nodes. Because each cache node is added only once, $(s/2^k)\sum \ell - O(2^k) \le s$, so the total path length is $O((2^k)^2/s + 2^k)$. Therefore the total time of the whole construction is $O(n^2/s + n)$, which is $O(n^2/s)$ when $s = O(\sqrt{n})$. □

D.6 Trade-offs for Tseitin formulas

In this section we show time-space trade-offs for CDCL refutations of Tseitin formulas, which are formulas encoding a particular form of unsatisfiable linear systems of equations mod 2. Let us give a formal definition of these formulas.

Definition D.6.1 (Tseitin formula). Let $G = (V, E)$ be an undirected graph and $\chi : V \to \{0, 1\}$ be a vertex *charge* function. Identify every edge $e \in E$ with a variable x_e that gets value in $\{0, 1\}$ and let $\mathit{PARITY}_{v,\chi}$ denote the CNF encoding of the parity constraint $\sum_{e \ni v} x_e = \chi(v) \pmod 2$ for a vertex $v \in V$. Then the *Tseitin formula* over G with respect to χ is $\mathit{Ts}(G, \chi) = \bigwedge_{v \in V} \mathit{PARITY}_{v,\chi}$.

When the degree of G is bounded by d, $\mathit{PARITY}_{v,\chi}$ has at most 2^{d-1} clauses, all of width at most d, and hence $\mathit{Ts}(G, \chi)$ is a CNF formula with at most $2^{d-1}|V|$ clauses. Again, we refer the reader to Figure D.4 for a small example of a Tseitin formula.

We say that a set of vertices U has *odd (even) charge* if $\sum_{u \in U} \chi(u)$ is odd (even). By a simple counting argument one sees that $\mathit{Ts}(G, \chi)$ is unsatisfiable if $V(G)$ has odd charge.

In this section we are interested in grid (multi-)graphs with double edges. A grid graph of size $w \times \ell$ has vertex set $\{(i, j) \mid 0 \le i < w, 0 \le j < \ell\}$. A grid graph wit double edges has the following set of edges: for each vertex (y, x), if $y < w - 1$, there are two vertical edges to $(y + 1, x)$ labelled $ev_{y,x}, ev'_{y,x}$, and if $x < \ell - 1$, there are two horizontal edges to $(y, x + 1)$ labelled $eh_{y,x}, eh'_{y,x}$.

For Tseitin formulas on grid graphs with double edges, trade-off results are know for resolution refutations.

Theorem D.6.2 ([23]). *Consider a Tseitin formula over a $w \times \ell$ grid graph with double edges, with odd charge function. It holds that*

- *the formula has a resolution refutation of length $O(2^{O(w)}\ell)$ and clause space $O(2^{O(w)})$,*
- *the formula has a tree-like resolution refutation of length $O(\ell^{O(w)})$ and clause space $O(w \log(\ell))$,*
- *if $1 \le w \le \ell^{1/4}$, then if a resolution refutation has length L and clause space s,*

$$L = \left(\frac{2^{\Omega(w)}}{s}\right)^{\Omega\left(\frac{\log\log \ell}{\log\log\log \ell}\right)}$$

The short resolution refutation follows a dynamic programming approach. During the proof we maintain a subset of vertices and we keep in memory the clauses that represent the sum of charges over that subset—equivalently, the sum over the border. The proof ends when we show that an empty border has odd charge. We update the subset by adding vertices ordered by column, so that the border has size at most $w+1$ and we can represent its sum in at most 2^w clauses. Adding a vertex means resolving a constant number of clauses with the clauses in memory, which gives a total length of $O(2^w w\ell)$.

The small resolution refutation is the tree-like proof induced by a search tree. The search tree performs a binary search along the columns, at each step querying the w variables in the middle column and then recursively exploring the subgraph of odd charge. The depth of this search tree is at most $w(\log(\ell)+1)$, which yields a proof of length $O(\ell^w)$ and space $O(w\log(\ell))$.

Our goal is to find CDCL equivalents of these resolution proofs. In the next two subsections, we show CDCL simulations of both proofs. We assume the 1UIP learning scheme.

Theorem D.6.3 (CDCL simulation results for Tseitin). *Consider the Tseitin formula over a $w \times \ell$ grid graph with double edges, with charge 1 in vertex $(0,0)$ and charge 0 in all other vertices. It holds that*

- *the formula has a time-efficient CDCL refutation with the 1UIP learning scheme of time $O(2^{5w}\ell)$ and space $O(2^{2w})$ simultaneously,*
- *the formula has a space-efficient CDCL refutation with the 1UIP learning scheme of space $O(w\log\ell)$ and time $O(\ell^{O(w)})$ simultaneously.*

Moreover, if $1 \le w \le \ell^{1/4}$, any CDCL refutation of space s has time

$$\tau = \left(\frac{2^{\Omega(w)}}{s}\right)^{\Omega\left(\frac{\log\log\ell}{\log\log\log\ell}\right)},$$

regardless of the learning scheme and of the restart policy.

In the proof of both simulations, we use the following notation. We denote the variable corresponding to edge e also by e. We denote the set of vertices between $x=l$ and $x=r$ by $V_{[l,r]}$ and the set of edges in the induced subgraph of $V_{[l,r]}$ by $E_{[l,r]}$. Let $EH_j = \bigcup_{0\le i<w}\{eh_{i,j}, eh'_{i,j}\}$ denote the set of $2w$ horizontal edges between columns j and $j+1$, for $0 \le j < w-1$. Let $EV_j = \bigcup_{0\le i<w-1}\{ev_{i,j}, ev'_{i,j}\}$ denote the set of $2w-2$ vertical edges in column j, for $0 \le j < w$. Let $\text{charge}(S) = \bigoplus_{e\in S} v(e)$ denote the parity of the set of variables S under the current assignment v, where $v(e)=1$ corresponds to assignment TRUE to variable e and $v(e)=0$ corresponds to FALSE.

In the dynamic programming proof, we will learn the set of clauses saying that $\text{charge}(EH_j) = 1$ for all $0 \le j < w-1$. This linear equation is encoded using 2^{2w-1}

clauses. Each of these clauses rules out a single even charge assignment to EH_j. Let L_j denote this set of clauses for EH_j.

The main lemma we use is the following.

Lemma D.6.4. *Consider a subgrid $V_{[l,r]}$ ($0 \leq l \leq r \leq w-1$). Suppose $E_{[l,r]}$ is unassigned and both EH_{l-1} and EH_r are assigned, unless $l = 0$ or $r = \ell - 1$. Let $(\ell_0, \ell_1, \dots, \ell_{2w-1})$ denote the last $2w$ assignments. Suppose $\{\mathrm{var}(\ell_i) \mid 0 \leq i < 2w\} \in \{EH_{l-1}, EH_r\}$, and let B be the set of literals assigned* TRUE *from $(EH_{l-1} \cup EH_r) \setminus \{\mathrm{var}(\ell_i) \mid 0 \leq i < 2w\}$ (B is empty if $l = 0$ or $r = \ell - 1$).*

Suppose that the only learnt clauses not being used as a reason are from the L_j sets, and that the restricted problem on $V_{[l,r]}$ would be satisfiable if any of the assignments to $EH_{l-1} \cup EH_r$ was inverted.

For each literal ℓ_i that is not a decision, denote its reason by C_i. Let $R_i = C_i \setminus \big(\{\neg\ell_j \mid 0 \leq j < i\} \cup \{\ell_i\}\big)$ and suppose $R_i \cap V_{[l,r]} = \emptyset$. Suppose at least one ℓ_i is a decision and let k be the largest index such that ℓ_k is a decision. Assume the only possible learnt clauses involving variables from $V_{[l,r]}$ are clauses from one of the L_j.

Consider any CDCL trace setting only variables within $E_{[l,r]}$ until the first conflict that causes the CDCL refutation to backtrack from ℓ_k (and possibly further). At this conflict, CDCL with 1UIP learns the clause $(\neg\ell_0 \vee \ldots \vee \neg\ell_k) \vee \bigvee_{i>k} R_i \vee \bigvee_{\ell \in B} \neg\ell$ with asserting literal $\neg\ell_k$.

Proof. Consider the resolution derivation of the conflict clause derived, where the clauses from the L_j that were learnt before the trace began are axioms, and we expand the derivation of clauses learnt during the trace. Observe that we never resolve over any of the variables in EH_{l-1} or EH_r, because these are part of the trail during the whole trace.

Suppose the asserting variable is one of the edges in $E_{[l,r]}$. Then ℓ_k was not used in the derivation; however, the conclusion does not hold if we invert ℓ_k: in that case, the restricted problem is satisfiable, and we can flip any two variables for each double edge in $E_{[l,r]}$ (note that this doesn't falsify the clauses from the L_j). But the derivation should also be true for the restricted formula with ℓ_k inverted, so we obtain a contradiction.

Therefore the asserting variable must be one of $\ell_k, \ell_{k+1}, \dots, \ell_{2w-1}$. Because the restricted formula $V_{[l,r]}$ would be satisfiable if any of the assignments to $EH_{l-1} \cup EH_r$ was inverted, for all border vertices of $V_{[l,r]}$ adjacent to the ℓ_i, we use at least one clause from the charge axioms for these vertices, involving each ℓ_i. So 1UIP resolves over $\ell_{k+1}, \dots, \ell_{2w-1}$ and ends up with asserting variable ℓ_k. □

D.6.1 Time-efficient proof

In this section, we show a CDCL simulation of the time-efficient proof. We learn all the L_j clause sets and try to do tree-like resolution otherwise. The simulation strategy is given as a recursive procedure `SolveDP`(j) which produces a sequence of decisions and forget commands. The initial call to generate this sequence is `SolveDP`($\ell - 2$). The clause deletion strategy will be to forget a learnt clause part of an L_j when a forget command for this clause is encountered in the sequence; all other learnt clauses are forgotten at the earliest stable state in which they do not occur as a reason anymore. We use a procedure $\mathtt{BF}(L, cb)$ that takes as input a list of variables L and a callback function cb, and tries all possibilities for the variables in L (generating a full binary tree of depth $|L|$; variables in L are decided on in the order they appear in L), calling function cb at each leaf in the tree (that is, for each assignment of L).

The strategy first assigns the variables in the order from $x = w$ to $x = 1$ and then learns the L_j from $x = 1$ to $x = w$. In order to learn that charge(EH_j) $= 1$, we try all possible assignments such that charge(EH_j) $= 0$ and derive UNSAT for the subproblem on $V_{[0,j]}$. For the first such assignment, we have to solve the whole subgrid $V_{[0,j]}$, but as a result we learn L_{j-1}, so that proving all other assignments UNSAT can be done without assignments past column $j - 1$. In order to learn a clause from L_j, the last assignment to EH_j must be a decision; if it is a propagation, variables from EH_{j+1} (used to refute an assignment to EH_j with odd charge) could be included in the conflict clause. Therefore, we process the assignments such that for assignments with even charge, the last assignment a decision. This is implemented by deciding the last variable such that the total charge is 0. For example, if $w = 3$, the bitstrings for EH_j are processed in the following order. The first bit denotes the top-level decision in the tree; bold numbers denote implications, the others decisions. 000, 00**1**, 0**1**1, 0**10**, **1**01, **1**0**0**, **11**0, **111**. Note that the last variable is a decision in every other step, and that in these steps, the parity of the bitstring is even. If the last variable is a decision, we learn a clause from L_j; otherwise, we use the assignment to $EH_j \cup EH_{j+1}$ to prove UNSAT in EV_{j+1}.

In the pseudocode, we denote by # trailingzeroes(i, n) the number of trailing zeroes of the binary representation of i ($0 \leq i < 2^n$; it returns n for $i = 0$).

Lemma D.6.5. *Suppose* $0 \leq j \leq \ell - 2$. *If certain preconditions hold, calling* $\mathtt{SolveDP}(j)$ *has time complexity (length of the trace)* $O(2^{5w} \cdot (j+1))$*, space complexity* $O(2^{2w})$ *and results in a set of postconditions.*

Preconditions:

- $E_{[0,j+1]}$ *is unassigned and there are no learnt clauses involving variables from* $E_{[0,j+1]}$.
- *If* $j = \ell - 2$, *the trail is empty; otherwise, the last assignments on the trail are the assignments to* EH_{j+1} *(all zeroes, and all decisions).*

Postconditions:

- $E_{[0,j+1]}$ *is unassigned.*
- *If* $j = \ell - 2$, *UNSAT is determined; otherwise, the clause* $\bigvee_{x \in EH_{j+1}} x$ *is learnt, and the only change to the trail is that the lowest decision is replaced by an implication of* $\bigvee_{x \in EH_{j+1}} x$.
- *All clauses in* L_j *are learnt and remembered.*

Proof. The Tseitin formula restricted to the current assignment is unsatisfiable on $V_{[0,j+1]}$ because, if $j < \ell - 1$, then charge(EH_{j+1}) $= 0$ so the restricted formula on $V_{[0,j+1]}$ has odd charge.

We bruteforce over EH_j. This means creating a full binary tree of depth $2w-1$ over the variables in EH_j, except for the last one $eh_{w-1,j}$. We set the variables in order of increasing y. At each leaf of the tree, we call `MainLoopDP`(j). Because we want to learn that charge(EH_j) $= 1$, as explained above, we decide the last variable $eh'_{w-1,j}$ such that charge(EH_j) $= 0$ and learn a clause from L_j by solving the inconsistent restricted formula on $V_{[0,j]}$. Then the last variable flips, charge(EH_j) $= 1$ and we solve the inconsistent restricted formula on $V_{[j+1,j+1]}$, that is, for EV_{j+1}.

If $j = 0$, in the first part of `MainLoopDP(0)` we bruteforce over EV_0. We state without proof that only at the end of this bruteforce, we get a conflict beyond the assignments to EV_0, and then by Lemma D.6.4 with $l = 0$ and $r = 0$, we learn each clause from L_0 (because B is empty and $k = 2w-1$, that is, the last variable set in EH_0 was a decision). In the second part, we bruteforce over EV_1. Again we state without proof that only at the end of this bruteforce, we get a conflict beyond the assignments to EV_1, and then by Lemma D.6.4 with $l = r = 1$, backtracking on the EH_0 set works as usual. At the end of the last call to `MainLoopDP(0)`, all literals in EH_0 were implied, so if $\ell = 1$ we have a top-level conflict and conclude UNSAT; otherwise, by Lemma D.6.4 with $l = 0$ and $r = 1$ we learn $\bigvee_{x \in EH_1} x$ and the lowest decision is negated.

Now assume $j > 0$. For the first leaf, `MainLoopDP`(j) recurses to `SolveDP`($j-1$). The result of this recursive call, by induction, is to learn L_{j-1} and to learn the first clause of L_j as well. For all other leafs, `MainLoopDP`(j) bruteforces over EH_{j-1} except the last edge, plus half the edges in EV_j, one edge per edge pair. This is enough to discover UNSAT because EH_{j-1} was learnt. Note that we skip the last edge of both EH_{j-1} and EV_j because these will be unit propagated (for EH_{j-1}, this is because L_{j-1} is learnt; for EV_j, it is because all other edges incident to the same vertex were already assigned). We state without proof that only at the end of this bruteforce, we get a relative top-level conflict. By Lemma D.6.4 with $l = 0$ and $r = j$, we learn a clause from L_j. In the second part of `MainLoopDP`(j), we have charge(EH_j) $= 1$, so EH_j and EH_{j+1} (or the empty set of edges adjacent to the last column if $j = \ell - 2$) have opposite parity and we discover the conflict by bruteforcing over EV_{j+1}. We state without proof that only at the end of this bruteforce, we get a conflict beyond these assignments. Except for the last leaf, by Lemma D.6.4, we backtrack in the intended way. For the last leaf, all assignments to EH_j are implications, by Lemma D.6.4 with $l = 0$ and $r = j+1$, we learn $\bigvee_{x \in EH_{j+1}} x$, and the lowest decision is negated, except if $j = \ell - 2$, in which case we have a top-level conflict and UNSAT is determined.

The time complexity: the non-recursive part of `SolveDP`(j) creates a search tree of depth $2w$, and calls `MainLoopDP`(j) at each leaf, generating another search tree of depth at most $3w$, so the time complexity is $O(2^{5w} \cdot (j+1))$. It can be verified that at most one edge propagates at each node in all “brute force” trees, so the total cost of the unit propagations is constant.

The space complexity: no clauses are learnt before the recursive call to `SolveDP`$(j-1)$. Except for the L_j, all clauses learnt during the "brute force" parts are used in a tree-like manner and removed when we are in a stable state and they are not used as a reason anymore. Except for the recursive call, we use $O(w)$ variables, so the additional space of other learnt clauses is $O(w)$. So the space complexity is $O(2^{2w})$. □

Procedure BF(L,`callback`)

```
Input: A list $L_0, L_1, \ldots, L_{n-1}$, and a callback function callback
for $i \leftarrow 0, 1, \ldots, 2^n - 1$ do
    for $j \leftarrow$ # trailingzeroes$(i, n) - 1, \ldots, 0$ do
        Decide $L_{n-1-j}$ =0/d
    callback()
```

Procedure MainLoopDP(j)

```
Decide $eh'_{w-1,j}$ =charge($EH_j \setminus \{eh'_{w-1,j}\}$)/d
if $j = 0$ then
    BF ([$ev_{0,0}, \ldots, ev_{w-2,0}$],None)
else
    if all variables in $EH_j$ are assigned zero then
        SolveDP ($j-1$)
    else
        BF ([$eh_{0,j-1}, eh'_{0,j-1}, \ldots, eh_{w-2,j-1}, eh'_{w-2,j-1}, eh_{w-1,j-1}, ev_{0,j}, \ldots, ev_{w-2,j}$],None)
Learn a clause from $L_j$
Assert $eh'_{w-1,j}$ =1 − charge($EH_j \setminus \{eh'_{w-1,j}\}$)/u
BF ([$ev_{0,j+1}, \ldots, ev_{w-2,j+1}$],None)
```

Procedure SolveDP(j)

```
BF ([$eh_{0,j}, eh'_{0,j}, \ldots, eh_{w-2,j}, eh'_{w-2,j}, eh_{w-1,j}$],MainLoopDP($j$))
if $j > 0$ then Forget all clauses in $L_{j-1}$
```

D.6.2 Space-efficient proof

In this section, we show a CDCL simulation of the space-efficient proof. The simulation strategy is given as a recursive procedure `SolveTreelike`(l, r) which produces a sequence of decisions.

The initial call to generate this sequence is `SolveTreelike`$(0, w-1)$. The clause deletion strategy will be to forget a learnt clause at the earliest stable state in which they do not occur as a reason anymore. We use the same procedure $\mathtt{BF}(L, cb)$ as described in the time-efficient proof.

The strategy bruteforces over the middle column $EH_{\lfloor (w-1)/2 \rfloor}$, and then recursively solves either $V_{[0,\lfloor (w-1)/2 \rfloor]}$ or $V_{[1+\lfloor (w-1)/2 \rfloor, w-1]}$ depending on which part has an odd total charge.

Lemma D.6.6. *Consider a subgrid $V_{[l,r]}$ ($0 \le l \le r \le w-1$). Suppose $V_{[l,r]}$ is unassigned and there are no learnt clauses containing variables within $E_{[l,r]}$. Suppose we assign EH_{l-1} (if $l > 0$) and EH_r (if $r < \ell - 1$) such that, in the resulting restricted Tseitin formula on $V_{[l,r]}$, the sum of charges is odd in column l, and even in columns $l+1, \dots, r$.*

Then $\mathtt{SolveTreelike}(l, r)$ *is a proof of UNSAT for the restricted problem and backtracks beyond assignments to $E_{[l,r]}$.*

Proof. If $l = r$, we state (without proof) that brute forcing over every other edge of EV_l constitutes an UNSAT proof with backtracking beyond assignments to EV_l after the last specified conflict. Otherwise, we first brute force over EH_m. At each iteration of $\mathtt{MainLoopTreelike}(l, r)$, we set the last edge in EH_m such that $\text{charge}(EH_m) = 0$. Then we recursively call $\mathtt{SolveTreelike}(l, m)$. After executing $\mathtt{SolveTreelike}(l, m)$, we get a conflict at the decision level of $eh'_{w-1,m}$. By Lemma D.6.4, CDCL flips $eh'_{w-1,m}$ and doesn't backtrack further. Then $\text{charge}(EH_m) = 1$ and we recursively call $\mathtt{SolveTreelike}(m+1, r)$. Note that the restricted sum of charges in the vertices of column $m+1$ is now odd, so the preconditions for the inductive step on $[m+1, r]$ hold. After executing $\mathtt{SolveTreelike}(m+1, r)$, by Lemma D.6.4, CDCL backtracks in the intended way; for the last leaf, all edges in EH_m are implications, so backtracking to the assignment before the recursive call occurs. □

Procedure MainLoopTreelike(l,r)

```
1 $m \leftarrow \lfloor (l+r)/2 \rfloor$
2 Decide $eh'_{w-1,m} = \text{charge}(EH_j \setminus \{eh'_{w-1,m}\})$/d
3 SolveTreelike ($l$, $m$)
4 Assert $eh'_{w-1,m} = 1 - \text{charge}(EH_j \setminus \{eh'_{w-1,m}\})$/u
5 SolveTreelike ($m+1$, $r$)
```

Procedure SolveTreelike(l,r)

```
1 if $l = r$ then
2     BF ([$ev_{0,l}, ev_{1,l}, \dots, ev_{w-2,l}$],None)
3 else
4     $m \leftarrow \lfloor (l+r)/2 \rfloor$
5     BF ([$eh_{0,m}, eh'_{0,m}, \dots, eh_{w-2,m}, eh'_{w-2,m}, eh_{w-1,m}$],MainLoopTreelike ($l$,$r$))
```

Theorem D.6.3 follows from the time-efficient and space-efficient refutations we just discussed together with Theorem D.6.2.

D.7 Concluding Remarks

In this paper, we present a proof system that closely models conflict-driven clause learning (CDCL) and yields natural measures not only of running time but also of memory usage and number of restarts. To the best of our knowledge, previous papers considered either zero restarts or very frequent restarts, and none of the models captured space. We show that lower bounds on proof size and space in resolution carry over to this CDCL proof system. Furthermore, we establish that currently known trade-offs between size and space in resolution can be transformed into essentially equally strong trade-offs between time and memory usage for CDCL, where the upper bounds are achieved by CDCL without any restarts using the standard 1UIP clause learning scheme, and the lower bounds apply even for arbitrarily frequent restarts and arbitrary clause learning schemes.

The focus of our work is theoretical, namely to see if CDCL proof search is in principle subject to the kind of trade-offs shown previously for the resolution proof system in which it searches for proofs. Since the answer turns out to be yes, an interesting direction for future work would be to investigate experimentally whether anything like these time-space trade-offs show up also in practice (when variable decisions have to be made constructively, typically using the VSIDS decision scheme).

Two other interesting problems are whether CDCL with 1UIP clause learning can simulate general resolution efficiently with respect to both time and space (measuring time only, a polynomial simulation follows from [152]), and whether CDCL with 1UIP and without restarts can simulate or be separated from regular resolution. If one believes that a separation should be more likely, a first step could be to revisit the formulas in [45, 51] and study them in our more restrictive setting, which more closely models actual CDCL search and hence might make proving lower bounds easier. It should be said, though, that both of these problems still look like formidable challenges, but one could hope that it would be possible to shed new light on them by focusing on a model that better describes how CDCL proof search works.

A more specialized question along the same lines, but still quite intriguing, is what can be said if VSIDS and phase saving is plugged into our CDCL model. The VSIDS heuristic seems like an important part of what makes CDCL SAT solvers so successful in practice, and yet there are also theoretical combinatorial formulas where it seems to be less useful. It would be interesting if one could find explicit examples of formulas where VSIDS in combination with phase saving goes provably wrong compared to the best possible resolution proof, causing a large polynomial or even superpolynomial blow-up in proof size.

Acknowledgements

We are grateful to the anonymous *SAT* conference reviewers for detailed comments that helped improve the exposition in this paper.

The third author performed this work while at KTH Royal Institute of Technology, and most of the work of the second and fourth author was done while visiting KTH. The first, third, fifth, and sixth author were funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 279611 as well as by Swedish Research Council grant 621-2012-5645. The third author was also supported by the European Research Council under the European Union's Horizon 2020 Research and Innovation Programme / ERC grant agreement no. 648276.

Paper E

Hardness of Approximation in PSPACE and Separation Results for Pebble Games

Siu Man Chan, Massimo Lauria, Jakob Nordström, and Marc Vinyals

Full length version of the article published in Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS '15), October 2015, pp. 466–485.

Abstract

We consider the pebble game on DAGs with bounded fan-in introduced in [Paterson and Hewitt '70] and the reversible version of this game in [Bennett '89], and study the question of how hard it is to decide exactly or approximately the number of pebbles needed for a given DAG in these games.

We prove that the problem of deciding whether s pebbles suffice to reversibly pebble a DAG G is PSPACE-complete, as was previously shown for the standard pebble game in [Gilbert, Lengauer and Tarjan '80]. Via two different graph product constructions we then strengthen these results to establish that both standard and reversible pebbling space are PSPACE-hard to approximate to within any additive constant. To the best of our knowledge, these are the first hardness of approximation results for pebble games in an unrestricted setting (even for polynomial time). Also, since [Chan '13] proved that reversible pebbling is equivalent to the games in [Dymond and Tompa '85] and [Raz and McKenzie '99], our results apply to the Dymond–Tompa and Raz–McKenzie games as well, and from the same paper it follows that resolution depth is PSPACE-hard to determine up to any additive constant.

We also obtain a multiplicative logarithmic separation between reversible and standard pebbling space. This improves on the additive logarithmic separation

previously known and could plausibly be tight, although we are not able to prove this.

We leave as an interesting open problem whether our additive hardness of approximation result could be strengthened to a multiplicative bound if the computational resources are decreased from polynomial space to the more common setting of polynomial time.

E.1 Introduction

In the *pebble game* first studied by Paterson and Hewitt [149], one starts with an empty directed acyclic graph (DAG) G with bounded fan-in (and which in this paper in addition will always have a single sink) and places pebbles on the vertices according to the following rules:

- If all (immediate) predecessors of an empty vertex v contain pebbles, a pebble may be placed on v.
- A pebble may be removed from any vertex at any time.

The goal is to get a pebble on the sink vertex of G with all other vertices being empty, and to do so while minimizing the total number of pebbles on G at any given time (the *pebbling price* of G). This game models computations with execution independent of the actual input. A pebble on a vertex indicates that the corresponding value is currently kept in memory and the objective is to perform the computation with the minimum amount of memory.

The pebble game has been used to study flowcharts and recursive schemata [149], register allocation [171], time and space as Turing-machine resources [65, 104], and algorithmic time-space trade-offs [58, 173, 168, 174, 176]. In the last 10–15 years, there has been a renewed interest in pebbling in the context of proof complexity as discussed in the survey [145] (although in this context one is often interested also in the slightly more general *black-white pebble game* introduced in [67]), and pebbling has also turned out to be useful for applications in cryptography [73, 8]. An excellent overview of pebbling up to ca. 1980 is given in [153] and some more recent developments are covered in the upcoming survey [146].

Bennett [31] introduced the *reversible pebble game* as part of a broader program [30] to investigate possibilities to eliminate (or significantly reduce) energy dissipation in logical computation. Another reason reversible computation is of interest is that observation-free quantum computation is inherently reversible. In the reversible pebble game, the moves performed in reverse order should also constitute a legal pebbling, which means that the rules for pebble placement and removal become symmetric as follows:

- If all predecessors of an empty vertex v contain pebbles, a pebble may be placed on v.

- If all predecessors of a pebbled vertex v contain pebbles, the pebble on v may be removed.

Reversible pebblings of DAGs have been studied in [133, 126] and have been employed to shed light on time-space trade-offs in reversible simulation of irreversible computation in [132, 129, 183, 48]. In a different line of work Potechin [155] implicitly used the reversible pebble game for proving lower bounds on monotone space complexity, with the connection made explicit in the follow-up works [57, 81].

Another pebble game on DAGs that will be of interest in this paper is the *Dymond–Tompa game* [74] played on a DAG G by a *Pebbler* and a *Challenger*. This game is played in rounds, with both players starting at the sink in the first round. In the following rounds, Pebbler places a pebble on some vertex of G after which Challenger either stays at the current vertex or moves to the newly pebbled vertex. This repeats until at the end of a round Challenger is standing on a vertex with all (immediate) predecessors pebbled (or on a source, in which case the condition vacuously holds), at which point the game ends. Intuitively, Challenger is challenging Pebbler to "catch me if you can" and wants to play for as many rounds as possible, whereas Pebbler wants to "surround" Challenger as quickly as possible. The *Dymond–Tompa price* of G is the smallest number r such that Pebbler can always finish the game in at most r rounds. The Dymond–Tompa game has been used to establish that for parallel time a speed-up by a logarithmic factor is always possible [74], and in [182] it was shown that a slightly modified variant of this game exactly characterizes parallelism in complexity classes like AC^i, NC, and P, and can be used to re-derive, for instance, Savitch's theorem. Furthermore, collapses or separations of these classes can in principle be recast (or discovered) as bounds on Dymond–Tompa price. Interestingly, this characterization of parallelism extends to proof complexity as well as discussed in [54].

A final game with pebbles that we want to just mention without going into any details is the *Raz–McKenzie game* introduced in [158] to study the depth complexity of decision trees solving search problems. The reason for bringing up the Dymond–Tompa and Raz–McKenzie games is that it was shown in [54] that both games are actually equivalent to the reversible pebble game. Hence, any bounds derived for the reversible pebble game also hold for Dymond–Tompa price and Raz–McKenzie price.

The main focus of this paper is to study how hard it is to decide exactly or approximately the pebbling price of a DAG. For the standard pebble game Gilbert et al. [88] showed that given a DAG G and a positive integer s it is PSPACE-complete to determine whether space s is sufficient to pebble G or not. It would seem natural to suspect that reversible pebbling price should be PSPACE-complete as well, but the construction in [88] cannot be used to show this.

Given that pebbling price is hard to determine exactly, an even more interesting question is if anything can be said regarding the hardness of approximating pebbling price. Although this seems like a very natural and appealing question, apparently next to nothing has been known about this.

Wu et al. [186] showed that "one-shot" standard pebbling price is hard to approximate to within any multiplicative constant assuming the so-called Small Set Expansion (SSE) hypothesis. In a one-shot pebbling one is only allowed to pebble each vertex once, however, and this is a major restriction since the complexity now drops from PSPACE-complete to NP-complete [171]. Note that containment in NP is easy to see since any one-shot pebbling can be described concisely just by listing the order in which the vertices should be pebbled (and it is easy to compute when a pebble is no longer needed and can be removed). In contrast, in the general case pebbling strategies that are optimal with respect to space can sometimes provably require exponential time.

One can also go in the other direction and study more general pebble games, such as the AND/OR pebble game introduced by Lingas [134] in one of the works leading up to [88]. Here every vertex is labelled AND or OR. For AND-vertices we have the usual pebbling rule, but for OR-vertices it is sufficient to just have one pebble on some predecessor in order to be allowed to pebble the vertex. This game has a relatively straightforward reduction from hitting set [80], which shows that it is hard to approximate to within a logarithmic factor, but the reduction crucially depends on the OR-nodes.

We remark that hardness of approximation in PSPACE for other problems has been studied in [63], but those techniques seem hard to adapt to pebble games since the reduction from QBF to pebbling is inherently unable to preserve gaps.

Another problem that we study in the current paper is the relation between standard pebbling price and reversible pebbling price. Clearly, the space needed to reversibly pebble a graph is at least the space required in the standard pebble game. It is also not hard to see that there are graphs that require strictly more pebbles in a reversible setting: for a directed path on n vertices only 2 pebbles are needed in the standard game, while it is relatively straightforward to show that the reversible pebbling space is $\Theta(\log n)$ [31, 133]. However, for "classic" graphs studied in the pebbling literature, such as binary trees, pyramids, certain superconcentrators, and the worst-case graphs in [151], the reversible and standard pebbling prices coincide asymptotically, and are sometimes markedly closer than an additive logarithm apart.

This raises the question whether reversible and standard pebbling can be asymptotically separated with respect to space. It might be worth pointing out in this context that for Turing machines it was proven in [129] that any computation can be simulated reversibly in exactly the same space. In the more restricted pebbling model, it was shown in [126] that if the standard pebbling price of a DAG G on n vertices is s, then G can be reversibly pebbled with at most $s^2 \log n$ pebbles. Thus, if there is not only an additive but also a multiplicative separation between standard and reversible pebbling price, such a separation cannot be too large.

E.1.1 Our Results

We obtain the following results:

1. We establish an asymptotic separation between standard and reversible pebbling by exhibiting families of graphs $\{G_n\}_{n=1}^{\infty}$ of size $\Theta(n)$ with a single sink and fan-in 2 which have standard pebbling price $s(n)$ and reversible pebbling price $\Omega(s(n)\log n)$. This construction works for any $s(n) = \mathrm{O}\big(n^{1/2-\epsilon}\big)$ with $\epsilon > 0$ constant, where the constant hidden in the asymptotic notation in the lower bound has a (mild) dependence on ϵ.

2. We prove that determining reversible pebbling price is PSPACE-complete. That is, given a single-sink DAG G of fan-in 2 and a parameter s, it is PSPACE-complete to decide whether G can be reversibly pebbled in space s or not.

3. Finally, we present two different graph products (for standard and reversible pebbling, respectively) that take DAGs G_i of size n_i with pebbling price s_i for $i = 1, 2$ and yield a DAG of size $\mathrm{O}\big((n_1 + n_2)^2\big)$ with pebbling price $s_1 + s_2 + K_p$ (for $K_p = \pm 1$ depending on the flavour of the pebble game). Combining these graph products with the PSPACE-completeness results for standard pebbling in [88] and reversible pebbling in item 2, we deduce that for any fixed K the promise problem of deciding for a DAG G (with a single sink and fan-in 2) whether it can be pebbled in space s or requires space $s + K$ is PSPACE-hard in both the standard and the reversible pebble game.

We need to provide more formal definitions before going into a detailed discussion of techniques, but want to stress right away that a key feature of the above results is the bounded fan-in condition. This is the standard setting for pebble games in the literature and is also crucial in most of the applications mentioned above. Without this constraint it would be much easier, but also much less interesting, to prove our results.[1]

Another aspect worth pointing out is that although the reversible pebble game is weaker than the standard pebble game, it is technically much more challenging to analyze. The reason for this is that a standard pebbling will always progress in a "forward sweep" through the graph in topological order, and so one can often assume without loss of generality that once one has pebbled through some subgraph the pebbling will never touch this subgraph again. For a reversible pebbling this is not so, since any pebble placed on any descendant of vertices in the subgraph will also have to be removed at some later time, and this has to be done in reverse topological order. Therefore, in any reversible pebbling there will be alternating phases of "forward sweeps" and "reverse sweeps," and these phases can also be interleaved at various levels. For this reason, controlling the progress of a reversible pebbling is substantially more complicated. Despite the additional technical difficulties, however, we consider the reversible pebble game to be

[1]The reason to emphasize this is that for unbounded fan-in the first author proved a PSPACE-completeness result for reversible pebbling in [55], but this result uses simpler constructions and techniques that do not transfer to the bounded fan-in setting. Another, somewhat related, example is that deciding space in the black-white pebble game has also been shown to be PSPACE-complete for unbounded indegree in [102], but there the unbounded fan-in can be used to eliminate the white pebbles completely, and again the techniques fail to transfer to the bounded indegree case.

at least as interesting to study as the standard and black-white pebble games in view of its tight connection with parallelism in circuit and proof complexity as described in [54].

E.1.2 Organization of This Paper

We present the necessary preliminaries in Section E.2 and then give a detailed overview of our results in Section E.3. We prove an asymptotic separation between standard and reversible pebbling in Section E.4. In Section E.5 we compute the exact price of some classic graphs, trees and pyramids, that we use in Section E.6 to construct technical gadgets. These play a key role in Section E.7, where we show that reversible pebbling is PSPACE-complete. We detail the graph product for reversible pebbling in Section E.8 and its counterpart for standard pebbling in Section E.9. Some concluding remarks are presented in Section E.10.

E.2 Preliminaries

All logarithms in this paper are base 2 unless otherwise specified. For a positive integer n we write $[n]$ to denote the set of integers $\{1, 2, \ldots, n\}$. We use Iverson bracket notation

$$[\![B]\!] = \begin{cases} 1 & \text{if the Boolean expression } B \text{ is true;} \\ 0 & \text{otherwise;} \end{cases} \tag{E.2.1}$$

to convert Boolean values to integer values.

E.2.1 Boolean Formula Notation and Terminology

A *literal* a over a Boolean variable x is either the variable x itself or its negation $\overline{x}$ (a *positive* or *negative* literal, respectively). A *clause* $C = a_1 \lor \cdots \lor a_k$ is a disjunction of literals. A *k-clause* is a clause that contains at most k literals. A formula F in *conjunctive normal form (CNF)* is a conjunction of clauses $F = C_1 \land \cdots \land C_m$. A *k-CNF formula* is a CNF formula consisting of k-clauses. We think of clauses and CNF formulas as sets, so that the order of elements is irrelevant and there are no repetitions.

A *quantified Boolean formula (QBF)* is a formula $\phi = Q_1 x_1\, Q_2 x_2 \ldots Q_n x_n\, F$, where F is a CNF formula over variables $x_1, \ldots, x_n$ and $Q_i \in \{\forall, \exists\}$ are universal or existential quantifiers (i.e., the formula is in prenex normal form with all variables bound by quantifiers). It was shown in [172] that it is PSPACE-complete to decide whether a QBF is true or not (where we can assume without loss of generality that F is a 3-CNF formula).

E.2.2 Graph Notation and Terminology

We write $G = (V, E)$ to denote a graph with vertices $V(G) = V$ and edges $\mathsf{edges}G = E$. All graphs in this paper are directed acyclic graphs (DAGs). An edge $(u, v) \in \mathsf{edges}G$ is an *outgoing edge* of u and an *incoming edge* of v, and we say that u is a *predecessor* of v and that v is a *successor* of u. We write $pred_G(v)$ to denote the set of all predecessors of v in G and $succ_G(v)$ to denote all its successors. Vertices with no incoming edges are called *sources* and vertices with no outgoing edges are called *sinks*. For brevity, we will sometimes refer to a DAG with a unique sink as a *single-sink DAG*, and this sink will usually be denoted z.

Taking the transitive closures of the predecessor and successor relations, we define the *ancestors* $anc_G(v)$ of v to be the set of vertices that have a path to v and the *descendants* $desc_G(v)$ to be the set of vertices on some path from v. By convention, v is an ancestor and descendant of itself. We write $anc^*_G(v) = anc_G(v) \setminus \{v\}$ and $desc^*_G(v) = desc_G(v) \setminus \{v\}$ to denote the *proper ancestors* and *proper descendants* of v, respectively. These concepts are extended to sets of pairwise incomparable vertices by taking unions so that $anc_G(U) = \bigcup_{u \in U} anc_G(u)$, $anc^*_G(U) = \bigcup_{u \in U} anc^*_G(u)$, et cetera, where we say that the vertices in U are pairwise incomparable when no vertex in the set is an ancestor of any other vertex in the set. When the graph G is clear from context we will sometimes drop it from the notation.

E.2.3 Standard and Reversible Pebble Games

A *pebble configuration* on a DAG $G = (V, E)$ is a subset of vertices $\mathbb{P} \subseteq V$. We consider the following three rules for manipulating pebble configurations:

1. $\mathbb{P}' = \mathbb{P} \cup \{v\}$ for $v \notin \mathbb{P}$ such that $pred_G(v) \subseteq \mathbb{P}$ (a *pebble placement* on v).

2. $\mathbb{P}' = \mathbb{P} \setminus \{v\}$ for $v \in \mathbb{P}$ (a *pebble removal* from v).

3. $\mathbb{P}' = \mathbb{P} \setminus \{v\}$ for $v \in \mathbb{P}$ such that $pred_G(v) \subseteq \mathbb{P}$ (a *reversible pebble removal* from v).

A *standard pebbling* $\mathcal{P}$ from $\mathbb{P}_0$ to $\mathbb{P}_\tau$ is a sequence of pebble configurations $(\mathbb{P}_0, \mathbb{P}_1, \ldots, \mathbb{P}_\tau)$ where each configuration is obtained from the preceding one by the rules 1 and 2 while in a *reversible pebbling* rules 1 and 3 should be used. The *time* of a pebbling $\mathcal{P} = (\mathbb{P}_0, \ldots, \mathbb{P}_\tau)$ is $\mathsf{time}(\mathcal{P}) = \tau$, and the *space* is $\mathsf{space}(\mathcal{P}) = \max_{0 \le t \le \tau}\{|\mathbb{P}_t|\}$.

We say that a pebbling is *unconditional* if $\mathbb{P}_0 = \emptyset$ and *conditional* otherwise. The *pebbling price* $\mathsf{Peb}_G(\mathbb{P})$ of a pebble configuration $\mathbb{P}$ is the minimum space of any unconditional standard pebbling on G ending in $\mathbb{P}_\tau = \mathbb{P}$, and we define the *reversible pebbling price* $\mathsf{RPeb}_G(\mathbb{P})$ by taking the minimum over all unconditional reversible pebblings reaching $\mathbb{P}$. The pebbling price of a single-sink DAG G with sink z is $\mathsf{Peb}(G) = \mathsf{Peb}_G(\{z\})$, and the reversible pebbling price of G is $\mathsf{RPeb}(G) = \mathsf{RPeb}_G(\{z\})$. We refer to such pebblings as *(complete) pebblings of G* or *pebbling strategies for G*. Again, when G is clear from context we can drop it from the notation, and from now on we will usually abuse notation by omitting the curly brackets around singleton vertex sets.

For technical reasons, we will often be interested in distinguishing particular flavours of reversible pebblings. Suppose that v is a vertex in G and that $\mathcal{P} = (\mathbb{P}_0 = \emptyset, \mathbb{P}_1, \ldots, \mathbb{P}_\tau)$ is a reversible pebbling. We will use the following terminology and notation:

- $\mathcal{P}$ is a *visiting pebbling* of v if $v \in \mathbb{P}_\tau$. The *visiting price* $\mathsf{RPeb}^V(v)$ of v is the minimal space of any such pebbling.

- $\mathcal{P}$ is a *surrounding pebbling* of v if $pred(v) \subseteq \mathbb{P}_\tau$ and the *surrounding price* $\mathsf{RPeb}^S(v)$ is the minimal space of any such pebbling.

- $\mathcal{P}$ is a *persistent pebbling* of v if it is a reversible pebbling of v in the sense defined before, i.e., such that $P_\tau = \{v\}$. We will sometimes refer to $\mathsf{RPeb}(v)$ as the *persistent price* of v to distinguish it from the visiting and surrounding prices.

We also define the visiting price for a single-sink DAG G with sink z as $\mathsf{RPeb}^V(G) = \mathsf{RPeb}^V_G(z)$ and the surrounding price as $\mathsf{RPeb}^S(G) = \mathsf{RPeb}^S_G(z)$.

Note that because of reversibility we could obtain exactly the same visiting space measure by defining a visiting pebbling of v to be a pebbling $\mathcal{P} = (\mathbb{P}_0, \mathbb{P}_1, \ldots, \mathbb{P}_\tau)$ such that $\mathbb{P}_0 = \mathbb{P}_\tau = \emptyset$ and $v \in \bigcup_{0 \le t \le \tau} \mathbb{P}_t$, and let the visiting price be the minimal space of any such pebbling. This is because once we have reached a configuration containing v we can simply run the pebbling backwards (because of reversibility) until we reach the empty configuration again. We can therefore think of a pebbling as *visiting* v if there is a pebble on v at some point but this pebble does not stay on v until the end of the pebbling. In a *persistent* pebbling the pebble remains on v until all other pebbles have been removed. A *surrounding* pebbling, finally, is a pebbling that reaches exactly the point where a pebble could be placed on v, since all its predecessors are covered by pebbles (i.e., v is "surrounded" by pebbles), but where v is not necessarily pebbled.

It is not hard to see that for a single-sink DAG G we have the inequalities

$$\mathsf{Peb}(G) \le \mathsf{RPeb}^V(G) \tag{E.2.2}$$

and

$$\mathsf{RPeb}^S(G) \le \mathsf{RPeb}^V(G) \le \mathsf{RPeb}(G) \ . \tag{E.2.3}$$

Perhaps slightly less obviously, we also have the following useful equality.

Proposition E.2.1. *For any vertex v in a DAG G it holds that $\mathsf{RPeb}^S(v) = \mathsf{RPeb}(v) - 1$.*

Proof. To see that $\mathsf{RPeb}(v) \le \mathsf{RPeb}^S(v) + 1$ consider a surrounding pebbling $\mathcal{P}^S$ of space $\mathsf{RPeb}^S(v)$. Let $\mathcal{P}^*$ be the pebbling which first runs $\mathcal{P}^S$ to surround v, then puts a pebble on v, and finally runs the reverse of $\mathcal{P}^S$ to "unsurround" v (while keeping the pebble on v). Since $\mathcal{P}^*$ is a persistent pebbling of space $\mathsf{RPeb}^S(v) + 1$, the inequality follows.

We now prove that $\mathsf{RPeb}^S(v) \le \mathsf{RPeb}(v) - 1$. Consider a persistent pebbling $\mathcal{P}$ for v of space $\mathsf{RPeb}(v)$. Let t be the last time that a pebble is put on v. Then vertex v is surrounded at time t, and there is a pebble on v since time t. Let $\mathcal{P}_{\ge t}$ be the conditional pebbling obtained from $\mathcal{P}$ after time t, with the modification that vertex v has no pebble throughout $\mathcal{P}_{\ge t}$, and let $\mathcal{P}^R_{\ge t}$ be this pebbling run in reverse. Then $\mathcal{P}^R_{\ge t}$ is a surrounding pebbling in space at most $\mathsf{RPeb}(v) - 1$, and the inequality follows. $\square$

E.2.4 The Dymond–Tompa and Raz–McKenzie Games

As described above, the *Dymond–Tompa game* on a single-sink DAG G is played in rounds by two players *Pebbler* and *Challenger*. In the first round Pebbler places a pebble on the sink z and Challenger challenges this vertex. In all subsequent rounds, Pebbler places a pebble on an arbitrary empty vertex and Challenger chooses to either challenge this new vertex (which we refer to as *jumping*) or to re-challenge the previously challenged vertex (referred to as *staying*). The game ends when at the end of a round all the (immediate) predecessors of the currently challenged vertex are covered by pebbles.[2] The *Dymond–Tompa price* $\mathsf{DT}(G)$ of G is the maximal number of pebbles r needed for Pebbler to finish the game, or expressed differently the smallest number r such that Pebbler has a strategy to make the game end in at most r rounds regardless of how Challenger plays.

Let us also for completeness describe the *Raz–McKenzie game*, which is also played on a single-sink DAG G by two players *Pebbler* and *Colourer*. In the first round Pebbler places a pebble on the sink z and Colourer colours it red. In all subsequent rounds, Pebbler places a pebble on an arbitrary empty vertex and Colourer then colours this new pebble either red or blue. The game ends when there is a vertex with a red pebble, while all its predecessors in the graph have blue pebbles. The *Raz–McKenzie price* $\mathsf{RM}(G)$ of G is the smallest number r such that Pebbler has a strategy to make the game end in at most r rounds regardless of how Colourer plays.

The intuition for this game is that the vertices on the graphs have assigned values true (blue) or false (red), with the condition that each vertex has value equal to the conjunction of the values of its predecessors. Colourer claims that the sink is false, but the above condition vacuously implies that all source vertices must be true. Colourer loses when Pebbler discovers a violation of the condition. Pebbler wants to find the violation as soon as possible, while Colourer wants to fool Pebbler for as long as possible.

In [54] the first author proved that the equalities

$$\mathsf{DT}(G) = \mathsf{RM}(G) = \mathsf{RPeb}(G) \tag{E.2.4}$$

hold for any single-sink DAG G, i.e., that the reversible pebbling price, the Dymond–Tompa price and the Raz–McKenzie price all coincide. Thus, any result we prove for one of these games is also guaranteed to hold for the other games. The above equalities are very convenient in that they allow us to switch back and forth between the reversible pebble game and the Dymond–Tompa game (or Raz–McKenzie game) when proving upper and lower bounds, depending on which perspective is more suitable at any given time. In particular, when proving lower bounds for reversible pebblings it is often helpful to do so by devising good Challenger strategies in the Dymond–Tompa game. One final technical remark in this context is that in all such strategies that we construct it holds that Challenger will either stay or jump to an ancestor of the currently challenged

[2]We remark that our description follows [54] and thus differs slightly from the original definition in [74], but the two versions are equivalent for all practical purposes.

vertex. Because of this we can assume without loss of generality that Pebbler only pebbles vertices in the subgraph consisting of ancestors of the currently challenged vertex. If Pebbler pebbles some vertex outside of this subgraph Challenger will just stay put on the current vertex, and so Pebbler just wastes a round.

E.3 Overview of Results and Sketches of Proofs

In this section we give a detailed overview of our results and also sketch some of the main ideas in the proofs. In the rest of the paper, we then provide all the missing technical definitions and present the actual formal proofs.

E.3.1 Separation Between Standard and Reversible Pebbling

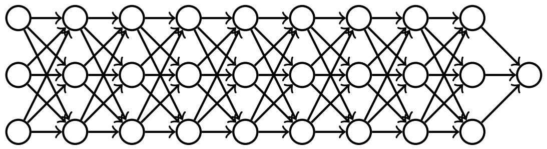
As mentioned in Section E.1, the strongest separation hitherto known between standard and reversible pebbling is for the length-ℓ path on vertices $\{v_1, v_2, \ldots, v_{\ell+1}\}$ with edges (v_i, v_{i+1}) for all $i \in [\ell]$, which has a standard pebbling with 2 pebbles whereas reversible pebblings require space $\Theta(\log \ell)$ [31, 133]. We give a simple construction improving this to a multiplicative logarithmic separation.

Theorem E.3.1. *For any function $s(n) = \mathrm{O}\big(n^{1/2-\epsilon}\big)$ for $\epsilon > 0$ constant there are DAGs $\{G_n\}_{n=1}^{\infty}$ of size $\Theta(n)$ with a single sink and fan-in 2 such that $\mathsf{Peb}(G) = \mathrm{O}(s(n))$ and $\mathsf{RPeb}(G) = \Omega(s(n)\log n)$ (where the hidden constant depends linearly on ϵ).*

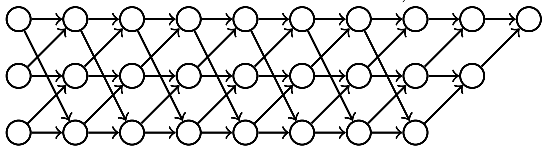
A first observation is that if we did not have the bounded fan-in restriction, Theorem E.3.1 would be very easy. In such a case we could just take the path of length ℓ, blow up every vertex v_i to s vertices $v_i^1, \ldots, v_i^s$, and add edges $\big(v_i^j, v_{i+1}^{j'}\big)$ for all $j, j' \in [s]$, so that we get a sequence of complete bipartite graphs $K_{s,s}$ glued together as shown in Figure E.1a. It is not hard to show that any reversible pebbling of this DAG would have to do s parallel, synchronized pebblings of the paths $\big\{v_1^j, v_2^j, \ldots, v_{\ell+1}^j\big\}$ for $j \in [s]$, which would require space $\Omega(s \log \ell)$, whereas a standard pebbling would clearly only need space $\mathrm{O}(s)$.

For bounded indegree it is not a priori clear what to do, however, or indeed whether there should even be a multiplicative separation. But it turns out that one can actually simulate a lower bound proof along the same lines as above by considering a layered graph as in Figure E.1b, with s parallel paths of length up to ℓ and with every path having an extra edge fanning out to its "neighbour path" above (or at the bottom for the top row) at each level. We will refer to this construction as a *road graph* of length ℓ and width s (where a path is a maximally narrow road of width 1). It is easy to verify that the standard pebbling price of a road of width $s \geq 2$ is $s + 2$. We claim that the reversible pebbling price is $\Omega\big(s \log(\ell/s)\big)$, from which Theorem E.3.1 follows.

To prove the reversible pebbling lower bound it is convenient to think instead in terms of Challenger strategies in the Dymond–Tompa game. The idea is that Challenger will stay put on the sink until Pebbler has pebbled enough vertices so that there are no



(a) Path blown up to sequence of $K_{3,3}$-graphs.



(b) Road graph of length 9 and width 3.

Figure E.1: Modifications of path graphs to amplify difference between reversible and standard pebbling price.

pebble-free paths from any source vertex to the sink. Intuitively, the cheapest way for Pebbler to disconnect the graph is with a straight cut over some layer. When this happens, Challenger looks at the latest pebbled vertex and compares the subgraph between the sources and the cut with the subgraph between the cut and the sink. If more than half of the graph is before the cut, Challenger jumps to the latest pebbled vertex. If not, Challenger stays on the sink. This strategy is then repeated on a graph of at least half the length. Since every cut by Pebbler requires s pebbles, Challenger can survive for roughly $s \log \ell$ rounds (except that the rigorous argument is not quite this simple, and the slightly smaller factor $\log(\ell/s)$ in the formal statement of the theorem is in fact inherent).

E.3.2 PSPACE-Completeness of Reversible Pebbling

Moving on to technically more challenging material, let us next discuss our PSPACE-completeness result for reversible pebbling, which we restate here more formally for the record.

Theorem E.3.2. *Given a single-sink DAG G of fan-in 2 and a parameter s, it is* PSPACE-*complete to decide whether G can be reversibly pebbled in space s or not. In more detail, given a QBF $\phi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n\ F$, where F is a 3-CNF formula over variables $x_1, \ldots, x_n$, there is a polynomial-time constructible single-sink graph $G(\phi)$ of fan-in 2 and a polynomial-time computable number $\gamma(\phi)$ such that $\mathsf{RPeb}\big(G(\phi)\big) = \gamma(\phi) + [\![\phi \text{ is } \textsc{false}]\!]$.*

At a high level, our proof is similar to that in [88] for standard pebbling: we build gadgets for variables, clauses, and universal and existential quantifiers, and then glue them together in the right way so that pebbling through the gadgets corresponds to verifying satisfying assignments for universally and existentially quantified subformulas of the QBF ϕ. However, the execution of this simple idea is highly nontrivial even in [88], and we run into several additional technical difficulties when we want to do an analogous reduction for reversible pebbling.

For starters, since the difference in pebbling price for graphs $G(\phi)$ obtained from true and false QBFs ϕ is just an additive 1, we need exact control over the pebbling price of all components used in the reduction. For standard pebbling there is no problem here—exact bounds on pebbling price are known for quite a wide selection of graphs—but in the reversible setting this becomes an issue already for almost the simplest possible graph: the complete binary tree of height h. An easy inductive argument shows that the standard pebbling price of such a tree is exactly $h + 2$. Since reversible pebblings find paths more challenging than do standard pebblings, one could perhaps expect an extra additive $\log h$ or so in the reversible pebbling bound. However, the asymptotically correct bound turns out to be $h + \Theta(\log^* h)$ as shown in [126], and the upper and lower bounds on the multiplicative constant obtained in that paper are far from tight.

The story is even worse for the workhorse of the construction in [88] (and many other pebbling results), namely *pyramids* of height h, which have i vertices at level i for $i = 1, \ldots, h + 1$, and where the jth vertex at level i has incoming edges from the jth and $(j + 1)$st vertices at level $i + 1$. There is a very neat proof in [65] that the standard pebbling price is again exactly $h + 2$, but for reversible pebbling price nothing has been known except that it has to be somewhere between $h + 2$ and $h + O(\log^* h)$ (where the latter bound follows since any strategy for a complete binary tree of height h works for any DAG of height h). As a crucial first step towards establishing Theorem E.3.2, we exactly determine the reversible pebbling price of pyramids (and also binary trees).

Theorem E.3.3. *For Δ denoting a positive integer, let g be the function defined recursively as*

$$g(\Delta) = \begin{cases} 0 & \text{if } \Delta = 1; \\ 2^{g(\Delta-1)+\Delta-2} + g(\Delta - 1) & \text{otherwise;} \end{cases}$$

and let the inverse g^{-1} of this function be defined as

$$g^{-1}(h) = \min\{\Delta \mid g(\Delta) \geq h\} \ .$$

Then the persistent pebbling price of a pyramid of height h, as well as of a complete binary tree of height h, is $h + g^{-1}(h)$, where g^{-1} is efficiently computable.

Even though Theorem E.3.3 is an important step, we immediately run into new problems when trying to use it as a building block in our reduction for reversible pebbling. In the standard pebble game a complete pebbling is any pebbling that reaches the sink. For the reversible game there is a subtle distinction in that we can ask whether it is sufficient to just reach the sink or whether the rest of the graph must also be cleared of pebbles. As discussed in Section E.2, this leads to two different flavours of reversible pebblings, namely *persistent pebblings*, which leave a pebble on the sink with the rest of the graph being empty, and *visiting pebblings*, which just reach the sink (and can then be thought to run in reverse after having visited the sink to clear the whole graph including the sink from pebbles). The pebblings we actually care about are the persistent ones, but we cannot rule out the possibility that subpebblings of gadgets are visiting pebblings. Clearly, the difference in pebbling space is at most 1, but this is exactly the additive 1 of which we cannot afford to lose control! To make things worse, for pyramids it turns out that persistent and visiting pebbling prices actually do differ except in very rare cases.

Because of this, we have to build more involved graph gadgets for which we can guarantee that visiting and persistent prices coincide. These gadgets are constructed in two steps. First, we take a pyramid and append a path of suitable length, depending on the height of the pyramid, to the pyramid sink, resulting in a graph that we call a *teabag*. Second, we take such teabags of smaller and smaller size and stack them on top of one another, which yields a graph that looks a bit like a *Christmas tree*. These Christmas tree graphs are guaranteed to have the same pebbling price regardless of whether a reversible pebbling is visiting or persistent.

With this in hand we are almost ready to follow the approach in the PSPACE-completeness reduction for standard pebbling in [88]. The idea is that we want to build gadgets for the quantifiers in a formula $\phi = \forall x \exists y \cdots Qz\, F$ of specified pebbling price so that the only way to pebble the graph $G(\phi)$ without using too much space is to first pebble the gadget for $\forall x$, then $\exists y$, et cetera, in the correct order until all quantifier gadgets have been pebbled. Once we get to the clause gadgets, we would like that the pebbles in the quantifier gadgets are locked in place encoding a truth value assignment to the variables, and that the only way to pebble through the clause gadgets without exceeding the space budget is if every clause contains at least one literal satisified by this truth value assignment.

In order to realize this plan, there remains one more significant technical obstacle to overcome, however. To try to explain what the issue is, we need to discuss the PSPACE-completeness reduction in [88] in slightly more detail. The way this reduction imposes an order in which the quantifier gadgets have to be pebbled is that pyramid graphs are included "at the bottom" of the gadgets (i.e., topologically first in order). The source vertices of the quantifier gadgets all appear in such pyramids, and one has to pebble through these pyramids to reach the rest of a gadget (where pebble placements encode variable assignments as mentioned above).

In the first, outermost quantifier gadget the pyramids have large height. In the second gadget the pyramid heights are slightly smaller, et cetera, down to the last, innermost quantifier gadget where the pyramids have smallest height. In this way, the pyramids are used to "lock up" pebbles and force a strict order of pebbling of the gadgets. It can be shown that in order not to exceed the pebbling space budget, any pebbling strategy has to start by pebbling the highest pyramids in the first gadget. If the pebbling starts anywhere else in the graph, this will mean that there are already pebbles elsewhere in the graph when the pebbling strategy reaches the first, highest pyramids in the outermost quantifier, but if so the overall pebbling has to use up too much space to pebble through this pyramid. One can also show that once the pyramids in the outermost quantifier gadget have been pebbled, the pebbling cannot proceed until the next quantifier gadget is pebbled. The pyramids in this gadget have smaller height, but there are also pebbles stuck in place in the outermost gadget, meaning that pyramids must again be pebbled in exactly the right order to stay within the space budget.

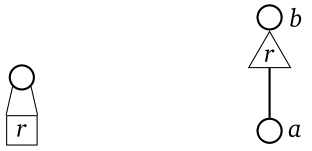
These properties can be used to *normalize* pebbling strategies in the standard pebble game. Without loss of generality, one can assume that any strategy that starts pebbling a pyramid in a gadget will complete this local pebbling in one go, leaving a pebble at the sink of the pyramid, and will not place pebbles anywhere else until the pebbling of the pyramid has been completed. Also, once a pyramid in a quantifier gadget has been pebbled in this way, one can prove that it will never be pebbled again since there is now at least one additional pebble at some vertex later in the topological order in the graph, and a repeated pebbling of the pyramid in question would therefore exceed the space budget. Thus, not only do the pyramids enforce that the gadgets are pebbled in the right order—they also serve as single-entry access points to the gadgets, making sure that each gadget is pebbled exactly once.

There is no hope of building gadgets with such properties in a reversible pebbling setting. It is simply not true that a reversible pebbling will pebble through a subgraph and then never return. Instead, as already discussed reversible pebblings will proceed in alternating phases of interleaved "forward sweeps" and "reverse sweeps," and subgraphs will be entered also in reverse topological order. Therefore, it is not sufficient to add "space-locking" subgraphs at the source vertices of the gadgets. Rather, we have to insert "single-passage points" inside and in between the gadgets for quantifiers and clauses. We obtain such subgadgets by further tweaking our Christmas tree construction so that it can also connect two vertices in such a way that any pebbling has to "pay a toll" to go through this subgraph. We cannot describe these gadgets, which we call *turnpikes*, in detail here, but mention that the "space-locking" property that they have is that when the entrance vertex is eliminated by having a pebble placed on that vertex, then the cost of pebbling through the rest of the turnpike drops by 1. This is critically used in the subgraph compositions described next.

Assuming the existence of the necessary technical subgraph constructions sketched above, we can now describe the overall structure of our reduction from quantified Boolean formulas to reversible pebbling (where all parameters shown in the figures are fixed appropriately in the formal proofs). In the following figures we denote a Christmas tree of (visiting and persistent) pebbling price r by the symbol in Figure E.2a, where we only display the sink vertex. We denote the turnpike gadget just discussed by the symbol in Figure E.2b. We write r to denote the *toll* parameter of the turnpike, where a turnpike with toll r has persistent price $r+2$, but only $r+1$ if we do not count the source a as part of the turnpike.

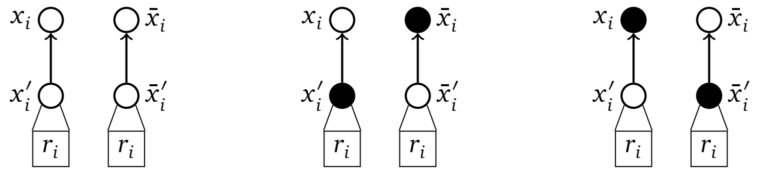
For every variable x_i we have a *variable gadget* as shown in Figure E.3a, where we think of a truth value assignment ρ as represented by pebbles on vertices $\{\bar{x}_i, x'_i\}$ when $\rho(x_i) = \text{FALSE}$ and on $\{x_i, \bar{x}'_i\}$ when $\rho(x_i) = \text{TRUE}$, as shown in Figures E.3b and E.3c, respectively.

For every clause C_j we have a *clause gadget* as depicted in Figure E.4a. The vertices



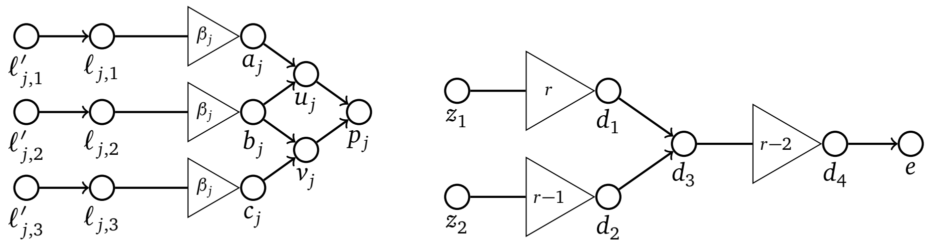
(a) Christmas tree. (b) Turnpike.

Figure E.2: Legend for technical gadget building blocks.



(a) Variable gadget. (b) FALSE position. (c) TRUE position.

Figure E.3: Gadget for variable x_i and pebble positions corresponding to truth value assignments.

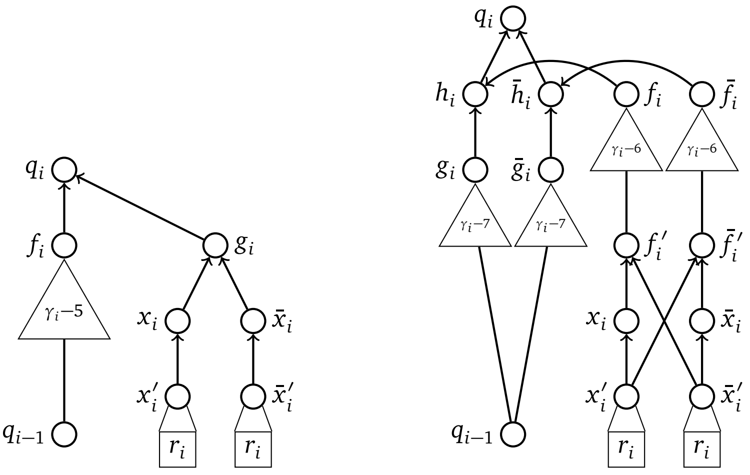


(a) Clause gadget. (b) Conjunction gadget.

Figure E.4: Gadgets for clauses and CNF formulas.

labelled $\ell'_{j,k}$ and $\ell_{j,k}$ in Figure E.4a are identified with the corresponding vertices for the positive or negative literal $\ell_{j,k}$ in the variable gadget in Figure E.3a. If ρ satisfies a literal, then there is a pebble on the entrance vertex of the corresponding turnpike, meaning that we can pebble through the gadget for a clause containing that literal with one less pebble than if ρ does not satisfy the clause.

To build the subgraph corresponding to a 3-CNF formula $F = \bigwedge_{j=1}^{m} C_j$ we join clause gadgets sequentially using the *conjunction gadget* in Figure E.4b. For technical reasons we start by joining a dummy graph with the first clause gadget, then we join the result to the second clause gadget, and so on up to the mth clause of F. The resulting graph



(a) Existential quantifier gadget. (b) Universal quantifier gadget.

Figure E.5: Quantifier gadgets for variable x_i.

has the property that if pebbles are placed on the variable gadgets according to an assignment ρ that satisfies F, then the number of additional pebbles needed to pebble the graph is one less than if the assignment is falsifying.

Finally we have one *quantifier gadget* for each variable. To describe this part of the construction, we sort the variables indices in reverse order from the innermost to the outermost quantifier and denote by ϕ_i the subformula with just the i innermost quantifiers, so that $\phi_0 = F = \bigwedge_{j=1}^{m} C_j$, $\phi_i = Q_i x_i\, \phi_{i-1}$ for $Q_i \in \{\forall, \exists\}$, and $\phi = \phi_n$. We construct graphs $G^{(i)} := G(\phi_i)$, starting with $G^{(0)}$ which is just the subgraph corresponding to the CNF formula F. To construct $G^{(i+1)}$ from $G^{(i)}$ we add an existential gadget as in Figure E.5a if x_i is existentially quantified and a universal gadget as in Figure E.5b if x_i is universally quantified. An example of the full construction can be found in Figure E.6.

Given this construction we argue along the same lines as in in [88], although as mentioned above there are numerous additional technical complications that we cannot elaborate on in this brief overview of the proof. We show that given an assignment ρ_i to $\{x_n, \ldots, x_{i+1}\}$, the number of additional pebbles needed to pebble $G^{(i)}$ differs by 1 depending on whether ϕ_i is true under the assignment ρ_i or not. An existential gadget can be optimally pebbled by setting x_i to any value that satisfies ϕ_{i-1}. To pebble a universal gadget one needs to assign x_i to some value, pebble through the gadget, unset x_i and assign it to the opposite value, and finally pebble through the gadget again, and both assignments to x_i must yield satisfying assignments to ϕ_{i-1} in order for the pebbling not to go over budget. Proceeding by induction, we establish that the complete graph $G^{(n)}$ can be pebbled within the specified space budget only if $\phi = \phi_n$ is true, which yields Theorem E.3.2.

E.3.3 PSPACE-Inapproximability up to Additive Constants

Let us conclude the detailed overview of our contributions by describing what is arguably the strongest result in this paper, namely a strengthening of the PSPACE-completeness of standard pebbling in [88] and of reversible pebbling in Theorem E.3.2 to PSPACE-hardness results for approximating standard and reversible pebbling price to within any additive constant K.

Theorem E.3.4. *For any fixed positive integer K it is* PSPACE*-complete to decide whether a single-sink DAG G with fan-in* 2 *has (standard or reversible) pebbling price at most s or at least $s + K$.*

We remark that it would of course have been even nicer to prove multiplicative hardness results. We want to stress again, though, that to the best of our knowledge these are the first results ever for hardness of approximation of pebble games in a general setting. The fact that these results hold even for PSPACE could perhaps be taken both as an indication that it should be possible to prove much stronger hardness results for algorithms limited to polynomial time, and as a challenge to do so.

We obtain Theorem E.3.4 by defining and analyzing two graph product constructions, one for standard and one for reversible pebbling, which take two graphs and output product graphs with pebbling price equal to the sum of the pebbling prices of the two input graphs (except for an additive adjustment). These graph products can then be applied iteratively $K - 1$ times to the graphs obtained by the reductions from QBFs. In the next theorem we state the formal properties of these graph products.

Theorem E.3.5. *Given single-sink DAGs G_i of fan-in 2 and size n_i for $i = 1, 2$, there are polynomial-time constructible single-sink DAGs $\mathcal{S}(G_1, G_2)$ and $\mathcal{R}(G_1, G_2)$ of fan-in 2 and size $\mathrm{O}\big((n_1 + n_2)^2\big)$ such that*

- *For standard pebbling it holds that $\mathsf{Peb}(\mathcal{S}(G_1, G_2)) = \mathsf{Peb}(G_1) + \mathsf{Peb}(G_2) - 1$.*
- *For reversible pebbling it holds that $\mathsf{RPeb}\big(\mathcal{R}(G_1, G_2)\big) = \mathsf{RPeb}(G_1) + \mathsf{RPeb}(G_2) + 1$.*

In the remainder of this section we try to convey some of the flavour of the arguments used to prove Theorem E.3.5 and to give a sense of some of the technical obstacles that have to be overcome during the analysis. In what follows, we will mostly focus on the reversible pebble game, since it is the technically more challenging and therefore also the more interesting case. We will briefly discuss the product construction for standard pebbling at the very end of the section. We will refer to G_1 as the *outer graph* and G_2 as the *inner graph* in the graph products $\mathcal{R}(G_1, G_2)$ and $\mathcal{S}(G_1, G_2)$.

Intuitively, when taking the graph product of G_1 and G_2 the idea is to replace every vertex v of the outer graph G_1 with a (possibly slightly modified) copy of the inner graph G_2. We will refer to this copy as the v-block in the product graph. The edges

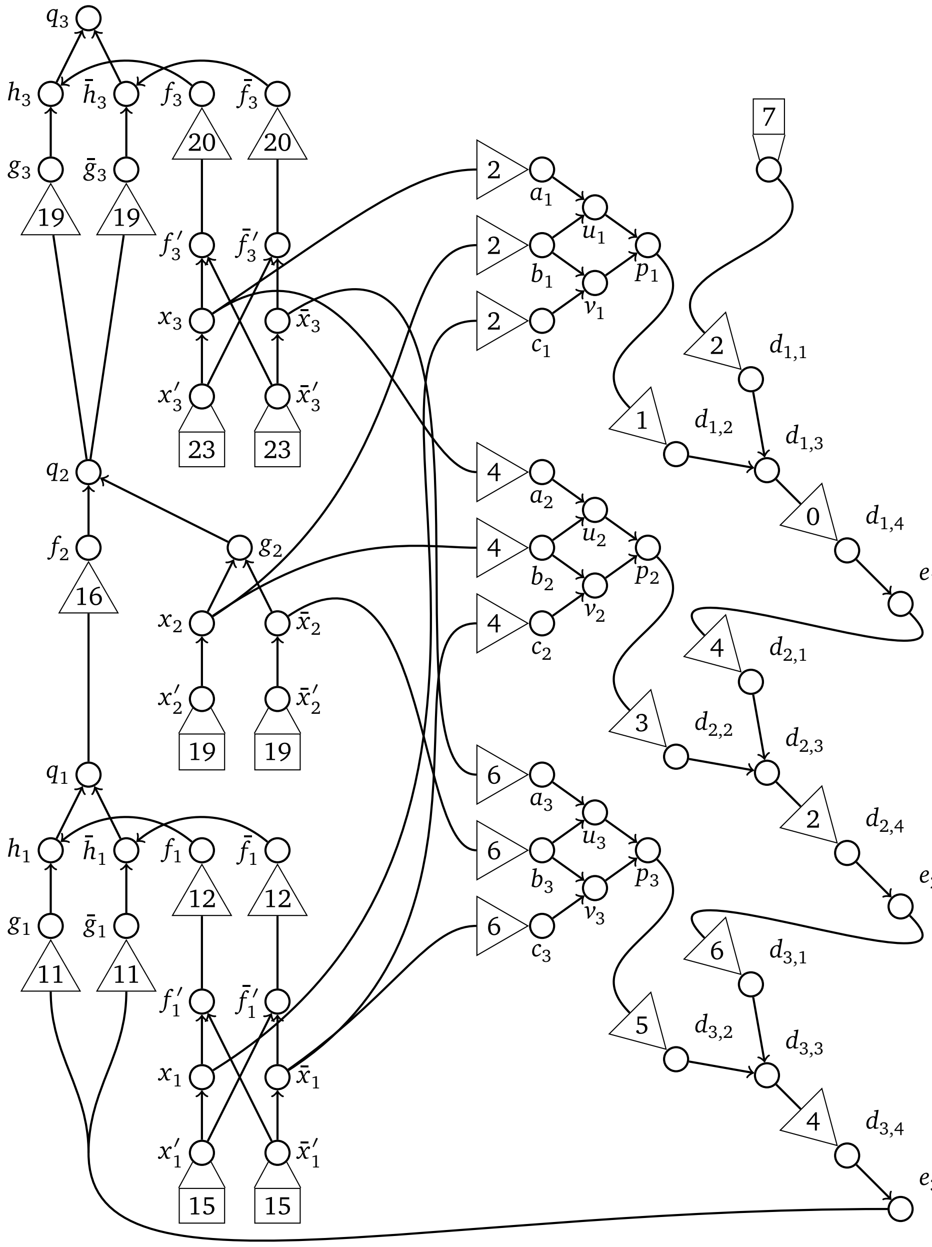


Figure E.6: Example of QBF-to-DAG reduction for $\forall x_3 \exists x_2 \forall x_1 (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$.

inside blocks are specified by the inner graph. For edges $(u, v) \in \mathsf{edges} G_1$ in the outer graph, we will need to connect the sink of the u-block to vertices in the v-block in some way, and this is the crux of the construction.

A first naive approach would be to add an edge from the sink of the u-block to every source vertex of the v-block (as shown in the graph product $\mathcal{N}$ in Figure E.7). Sadly, this simple idea fails for both standard and reversible pebbling. It is not hard to find examples showing that the pebbling price of $\mathcal{N}(G_1, G_2)$ is not a function of the pebbling prices of G_1 and G_2.

A slightly more refined idea is to add edges from the sink of the u-block to all vertices in the v-block (as in the graph $\mathcal{T}(G_1, G_2)$ in Figure E.7). While we can observe right away that this idea is a non-starter, since it will blow up the fan-in of the product DAG (and with no bounds on fan-in the gap amplification would be trivial), it turns out that the analysis yields interesting insights for the graph product that we will actually use. We will therefore employ this toy construction to showcase some of the ideas and technical challenges that arise in the actual proof of Theorem E.3.5.

Recall that we want to prove that $\mathsf{RPeb}\big(\mathcal{T}(G_1, G_2)\big) = s_1 + s_2 - 1$, where $s_i = \mathsf{RPeb}\big(G_i\big)$ for $i = 1, 2$. To reversibly pebble the product graph $\mathcal{T}(G_1, G_2)$ in at most this amount of space we simulate a minimal space pebbling of G_1, where pebble placement or removal involving a vertex v of G_1 invokes a complete pebbling (or unpebbling) of the copy of G_2 corresponding to the v-block. This simulation uses at most s_2 pebbles in the relevant v-block and at most $s_1 - 1$ pebbles on sinks of other blocks, i.e., no more than $s_1 + s_2 - 1$ pebbles in total.

Proving the lower bound $\mathsf{RPeb}\big(\mathcal{T}(G_1, G_2)\big) \geq s_1 + s_2 - 1$ is the difficult part. Here the approach is to assume that we are given a complete pebbling $\mathcal{P}^{\mathcal{T}}$ of $\mathcal{T}(G_1, G_2)$ and extract from it a pebbling strategy $\mathcal{P}$ for G_1 with the hope that an expensive configuration in $\mathcal{P}$ will also help us to pinpoint an expensive configuration in $\mathcal{P}^{\mathcal{T}}$.

The most straightforward way to obtain a pebbling strategy $\mathcal{P}$ for G_1 from $\mathcal{P}^{\mathcal{T}}$ would be to make a vertex v in G_1 contain a pebble or not depending only on the local pebble configuration of the v-block in $\mathcal{T}(G_1, G_2)$. A natural idea is that v should get a pebble if the v-block has a pebble on its sink and that this pebble should be removed from v when the corresponding block has been emptied of pebbles. If we apply this reduction to a pebbling $\mathcal{P}^{\mathcal{T}}$ of $\mathcal{T}(G_1, G_2)$ we obtain a valid pebbling of G_1. The problem, however, is that $\mathcal{P}^{\mathcal{T}}$ might locally be doing a visiting pebbling (as defined in Section E.2.3) of the copy of G_2 corresponding to the v-block as a way of moving pebbles on or off other blocks. The consequence of this would be that a configuration of maximal space s_1 in $\mathcal{P}$ may result from a configuration in $\mathcal{P}^{\mathcal{T}}$ that uses space only $s_1 + s_2 - 2$, which is off by one compared to what we need and hence destroys the gap in pebbling price that we are trying to create.

If the visiting price of G_2 is the same as its persistent price, then this problem does not arise, but since this does not hold for graphs in general we need to argue more carefully. It is true that a visiting pebbling of a copy of G_2 might save one pebble as compared to a persistent pebbling, but whenever the sink contains a pebble in a visiting

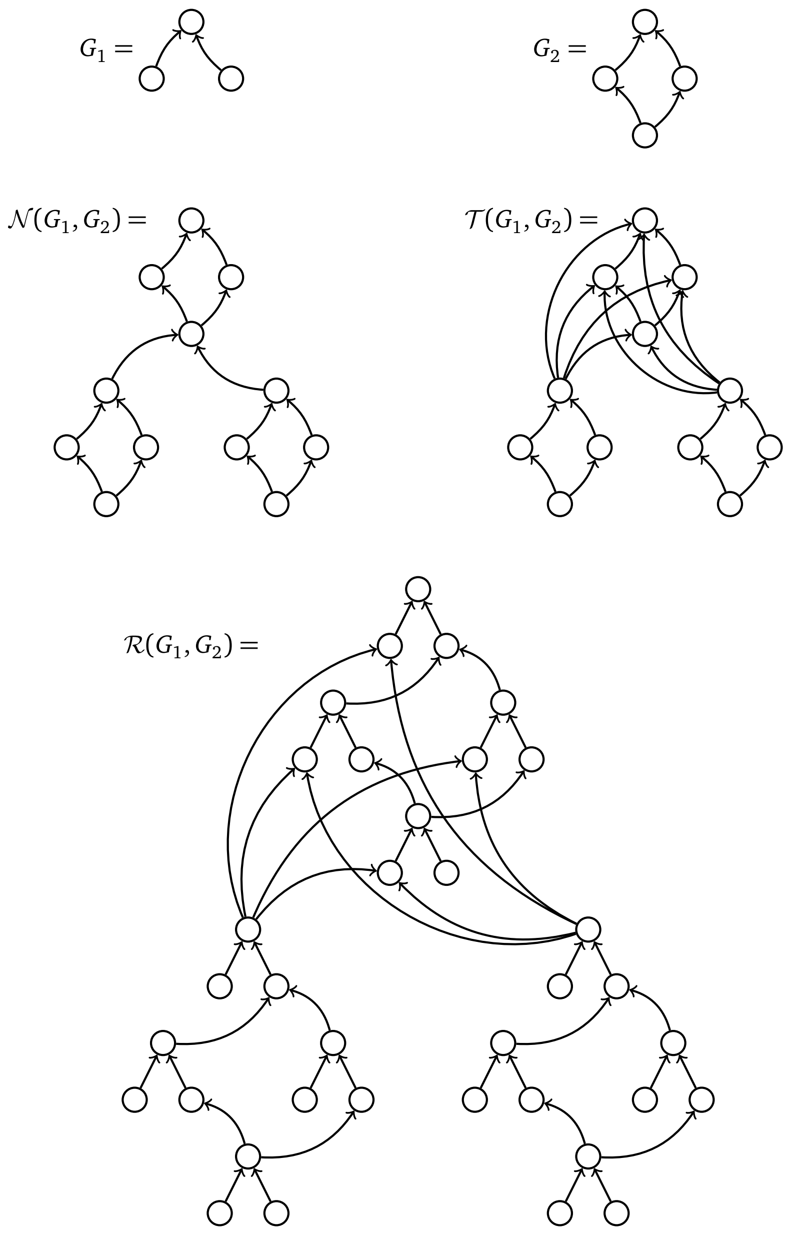

$G_1 =$
$G_2 =$
$\mathcal{N}(G_1, G_2) =$
$\mathcal{T}(G_1, G_2) =$
$\mathcal{R}(G_1, G_2) =$

Figure E.7: Examples of graph products as applied to a pyramid of height 1 (denoted G_1) and a rhombus (denoted G_2).

but not persistent pebbling we know that there must also be some other vertex in G_2 that has a pebble (or else the pebbling would be persistent by definition). We need to count such pebbles also in our analysis.

To this end, we make a distinction between blocks that have paid the persistent price and the blocks that have paid the visiting price but not the persistent price. We say that the copy of G_2 corresponding to some v-block is *visiting-locked*, or just *v-locked* for brevity, at some point in time if the current pebble configuration on its vertices requires reversible pebbling space $s_2 - 1$ to be reached, and that the v-block is *persistent-locked* (or *p-locked* for short) if the configuration has reversible pebbling price s_2.

We can now define a more refined way of projecting $\mathcal{P}^{\mathcal{T}}$-configurations to $\mathcal{P}$-configurations as follows. If a v-block has paid the persistent price, we put a pebble on the corresponding vertex v in G_1. If a block has paid just the visiting price but not the persistent price, then we might still put a pebble on v in G_1, but we only do so if an additional (and slightly delicate) technical condition[3] holds for the pebbling configurations in the blocks corresponding to predecessors of v. This technical condition is designed so that with some additional work[4] we are still able to extract a legal pebbling strategy for G_1 by applying this projection. Furthermore, it will be the case that every pebbling move on a vertex in the outer graph G_1 is the result of the copy of G_2 corresponding to some v-block paying the persistent or visiting price.

The reversible pebbling $\mathcal{P}$ thus extracted will be a persistent pebbling of G_1 by construction, so it must contain a configuration with s_1 pebbles. If this configuration was reached because a block paid the persistent price, then that block contains s_2 pebbles at a time when at least $s_1 - 1$ other blocks have at least 1 pebble each, which is the lower bound that we are after. If the pebble configuration on G_1 in $\mathcal{P}$ was reached because a block paid the visiting price, however, then we are potentially still one pebble short. This is where the additional technical condition mentioned above comes into play. This condition on the predecessor blocks implies that we can find at least one other block that also paid just the visiting price and therefore must contain two pebbles. Summing up, we obtain one block that has at least $s_2 - 1$ pebbles, another block that has at least 2 pebbles, and at least $s_1 - 2$ additional blocks that contain at least 1 pebble each, and

[3]We do not want to get into too detailed a technical argument here, but just for the record pebble configurations on $\mathcal{T}(G_1, G_2)$ can be projected to configurations on G_1 in two stages as follows:

1. Let $\mathbb{P} \subseteq V(G_1)$ consist of all vertices u such that the configuration on the u-block in $\mathcal{T}(G_1, G_2)$ is persistent-locked.
2. Let $\mathbb{P}' \subseteq V(G_1) \setminus \mathbb{P}$ consist of all vertices v such that (a) v is not already surrounded by $\mathbb{P}$, and (b) the configuration on the v-block in $\mathcal{T}(G_1, G_2)$ is visiting-locked.

With this notation, the projected pebble configuration on G_1 is defined to be $\mathbb{P} \cup \mathbb{P}'$.

[4]One added technical complication that we have to take care of here is that when we apply our projection to a pebbling $\mathcal{P}^{\mathcal{T}}$ of $\mathcal{T}(G_1, G_2)$ to obtain a sequence of pebble configurations on G_1, this sequence need not be a valid pebbling of G_1. However, when the projected pebble configuration on G_1 changes after a pebbling move we can insert a legal pebbling sequence between the two projected configurations that passes through all vertices of G_1 corresponding to v-locked blocks, where pebbles are added in topological order and removed in inverse topological order, and this local pebbling does not affect the overall argument.

so the lower bound holds in this case as well. (Incidentally, this second case is the one where our first, naive, graph product $\mathcal{N}(G_1, G_2)$ fails.)

We already observed, however, that the construction $\mathcal{T}(G_1, G_2)$ does not get us very far because it blows up the indegree of the resulting product graph. Therefore, in the actual proof of Theorem E.3.5 we have to consider a different construction. Briefly, the idea is to start with the graph $\mathcal{T}(G_1, G_2)$ but to bring the indegree down by splitting each vertex w in every block into three vertices $w_{\mathrm{ext}}, w_{\mathrm{int}}, w_{\mathrm{out}}$. All edges to w from other blocks are routed to w_{ext}, all edges from within the block are routed to w_{int}, and finally we add edges from w_{ext} and w_{int} to w_{out}. This is the graph product $\mathcal{R}(G_1, G_2)$ that we use to amplify differences in reversible pebbling price, and that is also illustrated in Figure E.7. Now we have to prove that the ideas just outlined work for this new construction where each vertex has been replaced by a small "cloud" of three vertices. The proof of this is much more technically challenging than for the toy case discussed above, and there is no room to go into details here.

At this point we want to switch gears a bit and briefly discuss an application in proof complexity of the PSPACE-hardness result for reversible pebbling. Perhaps the most well-studied proof system for proving the unsatisfiability of, or refuting, CNF formulas is *resolution* (we do not give any formal definition here, referring instead to, for instance, [170] for the necessary details). Every resolution proof can be represented as a DAG, and the *depth* of this proof is the length of a longest path in this DAG. The *resolution depth* of refuting an unsatisfiable CNF formula is the smallest depth of any resolution proof for the formula. It was shown in [54] that computing the reversible pebbling price of a graph of fan-in ℓ reduces to computing the resolution depth of a $(\ell+1)$-CNF formula, and from this we can obtain the following corollary.

Corollary E.3.6. *For any fixed positive integer K, it is* PSPACE*-complete to compute the resolution depth of refuting* 3*-CNF formulas up to an additive error K.*

Proof. Assuming that we can efficiently compute the resolution depth within an additive error at most K, we show how to efficiently compute the reversible pebbling price of any graph G within an additive error $K + 1$, contradicting Theorem E.3.4.

Letting z denote the unique sink of G, we consider a new graph G' which is G augmented with a new successor z' of z (i.e., $G' = (V \cup \{z'\}, E \cup \{(z, z')\})$ in formal notation). The reversible pebbling prices of G and G' differ by at most one. For any graph G, [54] exhibits an efficiently constructible unsatisfiable CNF formula F_G that requires resolution depth equal to the reversible pebbling price of G'. The width of the formula is equal to the fan-in of G plus one, so the result holds for 3-CNFs.

Hence, if we could estimate the resolution depth of refuting F_G, i.e., the reversible pebbling price of G', within error K, this would yield an estimate of the reversible pebbling price of G to within error $K + 1$. □

We wrap up this section by switching back to pebbling and describing the product construction $\mathcal{S}(G_1, G_2)$ used to amplify standard pebbling price. In this construction we

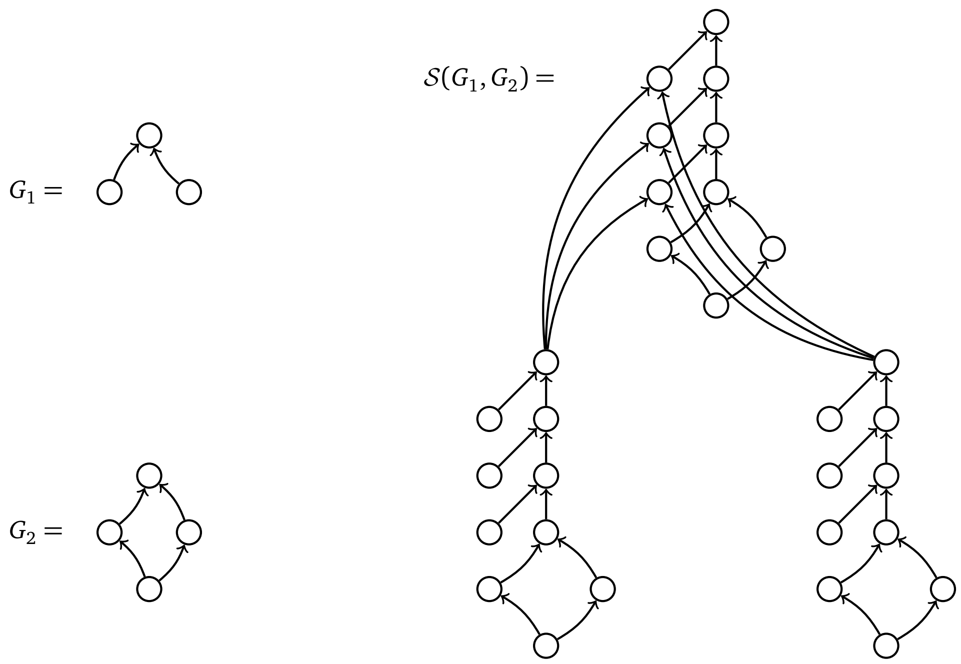


Figure E.8: Illustration of standard pebbling graph product $\mathcal{S}(G_1, G_2)$.

also replace every vertex of G_1 with a copy of G_2, but this time we append what we refer to as a *centipede* graph to the sink of every copy. A centipede is a path where each vertex but the source has an extra, unique predecessor. To connect the blocks, for every edge $(u, v) \in E(G_1)$ we add edges from the sink of the u-centipede to every source of the v-centipede. See Figure E.8 for an illustration.

Setting $s_i = \mathsf{Peb}\big(G_i\big)$ for $i = 1, 2$, we can pebble the graph product $\mathcal{S}(G_1, G_2)$ in space $s_1 + s_2 - 1$ by simulating an optimal pebbling of G_1: placing a pebble on a vertex v of G_1 is simulated by optimally pebbling the sink of the corresponding v-block, and removing a pebble is simulated by removing the pebble on the sink.

This pebbling strategy is in fact optimal, and we can show this by projecting any standard pebbling $\mathcal{P}^{\mathcal{S}}$ of $\mathcal{S}(G_1, G_2)$ to a strategy $\mathcal{P}$ for G_1. Each time any block in $\mathcal{S}(G_1, G_2)$ contains s_2 pebbles, we pebble all vertices in G_1 whose predecessors have pebbles and whose corresponding block in $\mathcal{S}$ has a pebble. When a block in $\mathcal{S}(G_1, G_2)$ becomes empty, we remove the pebble from the corresponding vertex in G_1. This projection has the property that when the sink of a block is pebbled, the corresponding vertex in G_1 is also pebbled. Arguing similarly to in the reversible case, we show that a strategy $\mathcal{P}^{\mathcal{S}}$ for $\mathcal{S}$ using s pebbles yields a strategy for G_1 using $s - s_2 + 1$ pebbles. Therefore, $\mathcal{P}^{\mathcal{S}}$ must use space at least $s_1 + s_2 - 1$, and hence the graph product $\mathcal{S}(G_1, G_2)$ has the property claimed in Theorem E.3.5.

E.4 Separation between Standard and Reversible Pebbling

In this section we discuss how the reversible pebbling price compares with the standard one. A reversible pebbling is also a legal standard pebbling, but the opposite is not always true. However it is possible to construct a reversible pebbling from a standard pebbling of time τ that costs at most $\log \tau$ times the price of the standard pebbling.

Theorem E.4.1 ([126]). *If graph G has a standard pebbling of time τ and space p, then G has reversible pebbling price at most $p\lceil \log \tau \rceil$.*

Proof sketch. Let $\mathcal{P} = (\mathbb{P}_0, \dots, \mathbb{P}_\tau)$ be a standard pebbling of G in space p. We show a Pebbler strategy for the Dymond–Tompa game on G that allows Pebbler to win in at most $p\lceil \log \tau \rceil$ rounds. Since $\mathsf{DT}(G) = \mathsf{RPeb}(G)$ this is sufficient. Pebbler keeps as an invariant an interval $[a, b]$ such that the challenged pebble is in $\mathbb{P}_b$ and all vertices in $\mathbb{P}_a$ are pebbled but not challenged. Initially the interval is $[0, \tau]$, and the strategy proceeds by bisection. At each bisection step Pebbler starts pebbling the vertices in the configuration $\mathbb{P}_m$, with $m = (a + b)/2$, in any order. If Challenger jumps to a vertex v, then let t be the smallest number such that $a \le t$ and $v \in \mathbb{P}_t$. Pebbler now plays in $[a, t]$. The interval halves because $t \le m$. If Challenger stays in all moves, then Pebbler plays in $[m, b]$ and the interval also halves. When the interval becomes unit, the Pebbler invariant implies that the move from $\mathbb{P}_a$ to $\mathbb{P}_b$ is precisely a placement on the challenged vertex. Therefore, the predecessors of the challenged vertex are pebbled and the game ends. The game considers at most $\lceil \log \tau \rceil$ configurations of $\mathcal{P}$ and spends at most p rounds on each. □

We already know that the difference between the standard and reversible pebbling price is unbounded. For example the standard pebbling price for a path of length n is 2, while its reversible pebbling price is $\Theta(\log n)$. It follows that if a DAG G has depth d and a standard pebbling of time τ and space p, then

$$\max\{p, \log d\} \le \mathsf{RPeb}(G) \le p \log \tau$$

We rule out the possibility of a simulation with only an additive loss. Indeed we show a separation which is multiplicative in terms of the logarithm of the size of the graph.

Theorem E.4.2. *For any function $s(n) = \mathrm{O}\big(n^{1/2-\epsilon}\big)$ where $\epsilon > 0$ is constant there are DAGs $\{G_n\}_{n=1}^{\infty}$ of size $\Theta(n)$ with a single sink and fan-in 2 such that $\mathsf{Peb}(G) = \mathrm{O}(s(n))$ and $\mathsf{RPeb}(G) = \Omega(s(n) \log n)$ (where the hidden constant depends linearly on ϵ).*

The graphs that we use to witness the separation are the chains or “wide paths”. A *chain* of width w and length ℓ is a graph with $\ell + 1$ layers, each having w vertices, where the i-th vertex of a layer has two incoming edges from the i and $i + 1$-th vertices of the

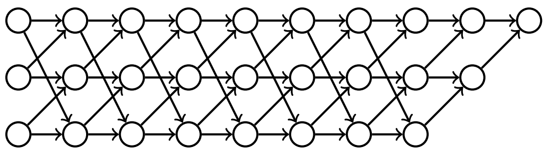


Figure E.9: Road graph of length 9 and width 3.

previous layer (modulo w). The layers are indexed from 0 (the layer of the sources) to ℓ (the layer of the sinks).

Since we want single sink graphs, we define a *road* of width w and length ℓ to be a chain of width w and length $\ell - w + 1$ plus a pyramid of height $w - 1$, where we identify the sinks of the chain with the sources of the pyramid. The layers are indexed in the same way as in the chain.[5]

By pebbling each layer in order, we get a standard pebbling of a road of width w which uses $w + 2$ pebbles. The reversible pebbling of a road depends on its length: a road of width w and length ℓ has a reversible pebbling price $\mathrm{O}(w \log \ell)$. The idea is to simulate in parallel w copies of the reversible pebbling of the path of length ℓ, which has price $\mathrm{O}(\log \ell)$. We prove Theorem E.4.2 by choosing for each n a road of width $w = s(n)$ and length $\ell = n/w$, and showing that this pebbling is essentially optimal.

A *blocking set* for a vertex set $T \subseteq V(G)$ is a subset of vertices $B \subseteq V(G)$ such that every path from any source to any vertex in T must contain a vertex in B. We also say that B *blocks* T. A blocking set for all sinks of a directed acyclic graph G is also called a blocking set of G. We say that B is a minimal blocking set if no subset of B is a blocking set.

A chain of width w has blocking sets with w vertices, all in the same layer. It turns out that if a minimal blocking set has vertices in multiple layers then it must be larger than that. We say that a blocking set *spreads* over d layers when a and b are the lowest and the highest layers that the blocking set intersects, respectively, and $d = b - a + 1$.

Lemma E.4.3. *A minimal blocking set of a chain that spreads over d layers has size at least $d + w - 1$.*

Proof. Consider such a minimal blocking set B. Sort the layers of the chain from 0 (sources) to ℓ (sinks) and let a and b be the first and last layers with vertices in B, so that $d = b - a + 1$. If $a = b$ then B must contain w vertices and the Lemma holds. For the rest of the proof we assume $a < b$.

Define $f(i)$ to be the number of vertices at layer i that can reach a sink without passing through a vertex in B (and are not in B themselves). By definition and minimality

[5]Equivalently, a road of length ℓ and width $w \leq \ell$ is an induced subgraph of a chain of length ℓ and width w. Fix one arbitrarily sink s in the chain: the subgraph induced by vertices in $anc(s)$ is indeed a road of length ℓ and width w.

we get that $f(i) = 0$ for all layers $i \leq a$; $0 < f(i) < w$ for all layers $a < i \leq b$ and $f(i) = w$ for $i > b$.

Now we compute the intersection between B and each layer. All vertices at layer b can reach the sink unless they are in B, therefore B must have $w - f(b)$ elements at the last layer. We claim that that B contains at least $f(i+1) - f(i) + 1$ vertices from layer i, for $a \leq i < b$.

Indeed, consider the set of the $f(i+1)$ vertices at layer $i+1$ that can reach a sink, and define N_i to be the set of their predecessors in layer i. Since $0 < f(i+1) < w$, there are at least $f(i+1)+1$ vertices in N_i.

A vertex in the i-th layer can reach a sink if and only if it is in N_i and not blocked by B. Since we assumed that exactly $f(i)$ of them can reach a sink, it must be that $|N_i| - f(i)$ vertices are blocked by B right on layer i, i.e., they are contained in B. Thus the intersection between B and layer i is $|B_i| \geq |N_i| - f(i) = f(i+1) - f(i) + 1$.

Using these facts we get that

$$|B| = \sum_{i=a}^{b} |B_i| \geq w - f(b) + \sum_{i=a}^{b-1} f(i+1) - f(i) + 1 = w + f(a) + (d-1) = d + w - 1 \ .$$

□

We need to generalize Lemma E.4.3 to a road in order to handle blocking sets within the pyramid part.

Lemma E.4.4. *A minimal blocking set of a road that spreads over d layers has size at least $d + q - 1$, where q is the width of the topmost layer.*

Proof. If the blocking set is located on a single layer the lemma follows immediately. Otherwise the proof is very similar to the one of Lemma E.4.3, except that the intersection between B and its last layer b has size at least $q - f(b)$. □

Now we prove the lower bound in Theorem E.4.2

Lemma E.4.5. *The reversible pebbling price of a road of width w and length ℓ is at least $w\log(\ell/w)/2$.*

Proof. We give a Challenger strategy by induction over ℓ that lasts for $w\log(\ell/w)/2$ moves. Furthermore, the strategy stays as long as the sink is connected to the sources. The base case is a road of length $\ell \leq w-1$, i.e., a pyramid of height ℓ, in which case the lemma holds vacuously.

We say that a directed path in the graph is *semiopen* when there are no pebbles on it except for its last vertex. A semiopen path from a vertex to itself is a single vertex with a pebble on it.

During the game Challenger *focuses* on a subgraph of the road, and keeps the following invariant at every round: there is a semiopen path from the sink of this subgraph to the currently challenged pebble. This concretely means that if Pebbler places a pebble inside the subgraph then Challenger plays according to its strategy for that subgraph. Instead if the new pebble blocks the semiopen path between the currently challenged pebble and the sink of the subgraph, Challenger jumps to the new pebble—essentially making the path shorter.

If at some round Challenger focuses on a subgraph, in later rounds Challenger will never challenge a vertex which is neither in the subgraph nor in the semiopen path between its sink and the current challenge.

Let us now give the strategy for playing inside the subgraph. As long as the sink of the subgraph is connected to the sources, Challenger stays. If the sink is disconnected from the sources by a blocking set, Challenger decides to jump or to stay depending on the position of the blocking set. Before describing how Challenger decides, we describe how the strategy continues in both cases.

We consider a minimal blocking set B and note that the last pebbled vertex u is in any blocking set. Indeed, there is a path from the sources to the sink that only has pebbles at u and at the sink, otherwise the sink would have already been blocked.

We first consider the strategy after Challenger decides to jump. Let v be the vertex in the semiopen path from the sources to u at the layer immediately before all the vertices in B. From now on Challenger focuses on the subgraph induced by the ancestors of v, which is a road. This road is not blocked, has no vertex in the blocking set B, and there is a semiopen path from v to the challenged pebble u.

If Challenger decides to stay, it focuses on the road with sources at the layer immediately after all vertices in B. Again, this road is not blocked, it is disjoint from the blocking set B, and there is a semiopen path from its sink to the currently challenged pebble.

It remains to describe how Challenger decides to jump or to stay. If the last layer of the blocking set has width $q < w$, then by Lemma E.4.4 it has size at least $q + d - 1$. In this case Challenger jumps and focuses on a road of length $\ell - (q + d - 2)$. Let $DT(w, \ell)$ be the Dymond–Tompa price of a road of width w and length ℓ. Overall the Challenger strategy lasts for

$$\begin{aligned} &|B| + DT(w, \ell - (q + d - 2)) \\ &\qquad \geq q + d - 1 + w \log((\ell - (q + d - 2))/w)/2 \geq w \log(\ell/w)/2 \end{aligned} \tag{E.4.1}$$

steps.

Otherwise the blocking set B has size at least $w + d - 1 \geq w$ for some $d \geq 1$ and by Lemma E.4.3 it spreads over at most d layers, where a is the lowest and b is the highest, with $d = b - a + 1$ and $m = (a + b)/2$. If $m \leq \ell/2$, Challenger stays and focuses on the road of length $\ell - b - 1$ obtained considering only the vertices at the layers from $b + 1$

to ℓ. Overall the Challenger strategy lasts for

$$\begin{aligned} |B| + DT(w, b-1) &\geq w + d - 1 + w\log((b-1)/w)/2 && \text{(E.4.2)} \\ &\geq w + d - 1 + w\log((\ell - d - 1)/2w)/2 && \text{(E.4.3)} \\ &= w/2\big(\log((\ell - d - 1)/2w) + 2 + 2(d-1)/w\big) && \text{(E.4.4)} \\ &\geq w/2\big(\log(\ell/2w) + 1\big) = w\log(\ell/w)/2 && \text{(E.4.5)} \end{aligned}$$

steps. If $m > \ell/2$, Challenger jumps and focuses on a road of length $a-1$ obtained considering one vertex at layer $a-1$ which is connected to the sources, and taking all vertices which have a path toward such vertex. Overall the Challenger strategy lasts for

$$\begin{aligned} |B| + DT(w, a-1) &\geq w + d - 1 + w\log((a-1)/w)/2 \\ &\geq w + d - 1 + w\log((\ell - d - 1)/2w)/2 \geq w\log(\ell/w)/2 \quad \text{(E.4.6)} \end{aligned}$$

steps. □

Note that Theorem E.4.2 follows if the road width is $w = s(n)$ and the length is $\ell = n/w$ because $w\log(\ell/w) = w\log(n/w^2) = \Theta(w\log n)$ if $w = O\big(n^{1/2-\epsilon}\big)$.

E.5 Tight Bounds for Trees and Pyramids

In this section we show matching upper and lower bounds for the persistent pebbling price of complete binary trees and pyramids. Asymptotically tight results for trees were given in [126]. The pyramid graph of height h has a vertex for every pair (i, j) with $0 \leq i \leq j \leq h$. The sources are the vertices $(0, j)$ for $j \geq 0$, the sink is the vertex (h, h), and every vertex (i, j) for $i < h$ has one outgoing edge going left to vertex $(i+1, j)$ if $j > i$ and one outgoing edge going right to vertex $(i+1, j+1)$ if $j < h$. A pyramid can be obtained from a complete binary tree of height h by identifying together some of its vertices, in such a way that the left and right predecessors of two vertices that get identified, get pairwise identified as well. For this reason an upper bound for binary trees also holds for the pyramids, and a lower bound for pyramids also holds for binary trees.

In the following, let $p = h + \Delta$ be the persistent price of a pyramid. A pyramid of height h has standard pebbling price $h + 2$ [65], which means that in the standard pebbling only two extra pebbles are needed compared to the height of the pyramid. In a similar fashion Δ can be interpreted as the extra space needed by persistent pebbling, with respect to pyramind height. We want to estimate the height of the pyramid that has persistent pebbling with at most Δ extra pebbles.

Definition E.5.1. Consider $\Delta \in \mathbb{N}^+$ and let $g(\Delta)$ be the function defined by the following recursion,

$$g(\Delta) = \begin{cases} 0 & \text{if } \Delta = 1 \\ 2^{g(\Delta-1)+\Delta-2} + g(\Delta-1) & \text{otherwise.} \end{cases}$$

We define its inverse as

$$g^{-1}(h) = \min\{\Delta \mid g(\Delta) \geq h\}.$$

We show that $g(\Delta)$ is the maximum height of a pyramid that can be persistently pebbled using Δ pebbles on top of h. Observe that $g(\Delta) = \Omega(\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{\Delta})$, therefore $g^{-1}(h) = \mathrm{O}(\log^* h)$.

Proposition E.5.2. *We can compute $g^{-1}(h)$ in time $(\log h)^{\mathrm{O}(1)}$ and space $\mathrm{O}(\log h)$.*

Proof. If $h = 0$ then $g^{-1}(0)$ is 1 by definition. If $h > 0$ then we need to find the smallest $\Delta > 1$ such that $h \geq g(\Delta)$. We start from $\Delta := 2$ and go upward. At each step we keep in memory the value $g(\Delta - 1)$, which is smaller than h, and we test the condition

$$h < g(\Delta) \quad \text{equivalent to} \quad \lfloor \log(h - g(\Delta - 1)) \rfloor < g(\Delta - 1) + \Delta - 2 \ .$$

The latter test can be achieved in time in $\log h$ by checking the length of the bit representation of $h - g(\Delta - 1)$. If the test fails we output Δ otherwise we store the value $g(\Delta)$, we fix $\Delta := \Delta + 1$ and we continue. We can do the whole computation by storing at most 4 numbers less than h, each step is polynomial in the length of the binary representation of the numbers involved, and we need to do at most $\mathrm{O}(\log^* h)$ steps. □

Theorem E.5.3. *The persistent pebbling price of a binary tree of height h and a pyramid of height h is $h + g^{-1}(h)$.*

To prove the theorem we need the exact value of the persistent pebbling price of paths.

Lemma E.5.4 (Path graphs [133]). *The persistent pebbling price of a path of length h (i.e., with $h + 1$ vertices) is $\lfloor \log(h) \rfloor + 2$.*

Lemma E.5.5 (Upper bound for binary trees). *The persistent pebbling price of a complete binary tree of height $h \leq g(\Delta)$ is at most $h + \Delta$.*

Proof. We are going to prove the lemma by induction over h. For the base case we observe that a binary tree of height 0 can be pebbled with 1 pebbles. For the general case we assume the statement of the lemma for height $i < h$, and we show that the surrounding pebbling price of the complete binary tree of height h is at most $h + \Delta - 1$. Proposition E.2.1 immediately implies the lemma for height h.

Let us denote the root by v_h and the right child of v_i by v_{i-1}. The strategy is as follows. First we persistently pebble the left child of v_i for i from h down to $k := g(\Delta - 1) + 1$, in this order. By the induction hypothesis $(i - 1) + \Delta$ pebbles are enough to persistently pebble the left child of v_i, and there are $h - i$ pebbles left on the rest of the graph from previous steps. So we are within the bound $h - 1 + \Delta$, and after the last step we have $h - (k - 1)$ pebbles on the tree.

Then we persistently pebble v_{k-1}, the right child of v_k. Since $k-1 = g(\Delta-1)$, by induction hypothesis $(k-1)+(\Delta-1)$ pebbles are enough and we are within the bound. Let $j := h-k+1$. So far we used $j+1 = h-k+2$ pebbles.

Finally we surround the sink of path $(v_k, v_{k+1}, \ldots, v_h)$, which has j vertices, using $\lfloor \log(j-1) \rfloor + 1$ pebbles. Observe that by construction $j-1 = h-k \leq g(\Delta)-g(\Delta-1)-1 < 2^{g(\Delta-1)+\Delta-2}$, hence we have the bound $\lfloor \log(j-1) \rfloor < g(\Delta-1)+\Delta-2$. Counting the total number of pebbles in the graph gives $\bigl(h-k+2\bigr)+\lfloor \log(j-1) \rfloor + 1 \leq \bigl(h-g(\Delta-1)+1\bigr)+\bigl(g(\Delta-1)+\Delta-3\bigr)+1 = h+\Delta-1$ pebbles. □

We prove the lower bound for a slight generalization of pyramids in order to obtain a lower bound on the visiting price in addition to the persistent price.

Definition E.5.6. An (h,ℓ)-teabag is the union of a pyramid of height h and a path of length ℓ, where we identify the sink of the pyramid and the source of the path.

Observe that an $(h,0)$-teabag is a pyramid.

For the lower bound we will also need the following basic fact about pyramids. Recall that a blocking set is a subset of vertices $B \subseteq V(G)$ such that every path from any source to the sink must contain a vertex in B. Also recall that a directed path in the graph is semiopen when there are no pebbles on it except for its last vertex.

Proposition E.5.7 ([65]). *Consider a blocking set B on a pyramid of height h; consider a vertex v at level k such that there is a path between v and the sink whose intersection with B is at most $\{v\}$. Let U be the set of vertices in the sub-pyramid rooted at v. Then $|B \setminus U| \geq h-k$.*

Proof. Pick an arbitrary path which starts at vertex v, reaches the pyramid sink and does not intersect B anywhere other than in v. Denote such path as $(v_k, v_{k+1}, \ldots, v_h)$ where v_k is another name for v and v_h is the sink. Each v_i is at height i in the pyramid. On pyramids there is a natural notion for edges to go either left or right. For each $i > k$ we define the path P_i as follows: if edge (v_{i-1}, v_i) goes right then P_i is the unique path that starts at a source vertex, always goes left, and ends at v_i; if edge (v_i, v_{i-1}) goes left then P_i is the unique path that starts at a source vertex, always goes right, and ends at v_i. It is easy to verify that none of $P_h, \ldots, P_{k+1}$ intersects any of the vertices in U, and that these paths are all pairwise vertex disjoint. Since B is a blocking set it must contain one vertex for each P_i and the proposition follows. □

Lemma E.5.8. *The persistent pebbling price of the (h,ℓ)-teabag is at least $h+\Delta+1$ if either of the following holds:*

- *$h > g(\Delta)$,*
- *$h > g(\Delta-1)$ and $\ell > g(\Delta)-h$.*

Proof. We define a Challenger strategy for the Dymond–Tompa game by induction over h and ℓ in this order. Furthermore this strategy stays on the sink until Pebbler blocks the graph. For the base case $h = 0$, the statement is trivial.

Assume that the last Pebbler move blocks the graph, meaning that the currently pebbled vertices form a blocking set, and fix B to be a minimal one. The vertex v pebbled at that round must be in B. We have two cases depending on k the layer of vertex v.

- Case $k > g(\Delta - 1)$: Challenger jumps to v. The pebble on v blocks the sources from the sink, so there must be a semiopen path between v and a source. Let U be the set of pebbles contained in the vertices of P_v, the subgraph of predecessors of v (notice that $v \in U$). Consider a new game on P_v, in which the first actions of Pebbler are to pebble $U \setminus \{v\}$ in any order, while Challenger stays on v. The set $U \setminus \{v\}$ does not block the subgraph. If $k \geq h$ then the new sub-game ends in at least $h + \Delta$ steps, and the total number of rounds is at least $1 + h + \Delta$. Otherwise v is inside the pyramid, and the new sub-game ends in $k + \Delta$ steps. We use Proposition E.5.7 to claim that $|B \setminus U| \geq h - k$. So in total the rounds in the game are at least

$$\underbrace{1}_{\text{challenge to sink}} + \underbrace{|B \setminus U|}_{\text{outside } P_v} + \underbrace{k + \Delta}_{\text{subgame on } P_v} \geq h + \Delta + 1 \ .$$

- Case $k \leq g(\Delta - 1)$: there is a path of length $h - k + \ell$ from v to the sink having only a pebble on each end. So any optimal Pebbler strategy must contain a strategy for playing on the semiopen path of length $h - k + \ell - 1$ from one unpebbled successor of v to the sink. Fix $q = \lfloor \log(h - k + \ell - 1) \rfloor$, the sub-game on the path of length $h - k + \ell - 1$ lasts at least $q + 2$ rounds (see Lemma E.5.4). The initial challenge on the sink of the graph is part of this sub-game, but all moves on B are not, so the total number of rounds is

$$\underbrace{|B|}_{\text{blocking set}} + \underbrace{q + 2}_{\text{subgame on path}} \geq \underbrace{h - k + 1}_{\text{blocking set}} + \underbrace{g(\Delta - 1) + \Delta}_{\text{subgame on path}} \geq h + \Delta + 1 \ .$$

  The bound on $|B|$ holds because Proposition E.5.7 on vertex v implies $|B \setminus \{v\}| \geq h - k$. The bound on $q + 2$ holds because by hypothesis $h - k + \ell - 1 \geq g(\Delta) - g(\Delta - 1)$, which implies that $q \geq \lfloor \log(g(\Delta) - g(\Delta - 1)) \rfloor \geq g(\Delta - 1) + \Delta - 2$. □

Corollary E.5.9 (Lower bound for pyramids). *The persistent pebbling price of a pyramid of height $h > g(\Delta - 1)$ is at least $h + \Delta$. The visiting price of a pyramid of height $h = g(\Delta)$ is at least $h + \Delta$.*

Proof. Lemma E.5.8 claims the first statement. The second one holds because if the pyramid of height $g(\Delta)$ had visiting price $h + \Delta - 1$, then the $(g(\Delta), 1)$-teabag would have persistent pebbling price $h + \Delta$, which contradicts Lemma E.5.8. □

E.6 Technical Constructions

In order to discuss lower bounds on pebbling price we need to identify expensive pebbling configurations, namely the configurations that are expensive to reach from the empty configuration. We will often use the reverse direction, i.e., that the empty configuration cannot be reached without passing through an expensive configuration.

Definition E.6.1. A configuration $\mathbb{P}$ is *v-locked* if $\mathsf{RPeb}_G(\mathbb{P}) = \mathsf{RPeb}^V(G)$. A configuration $\mathbb{P}$ is *p-locked* if $\mathsf{RPeb}_G(\mathbb{P}) = \mathsf{RPeb}(G)$.

E.6.1 Christmas Tree Construction

This section builds on the pyramid graphs to provide a graph T_r with equal visiting and persistent prices r for every $r \in \mathbb{N}^+$. As a preliminary step we show a graph G_p with persistent price p for every $p \in \mathbb{N}^+$.

Lemma E.6.2 (Modified Pyramids). *There is a family of graphs $\{G_p\}_{p\in\mathbb{N}^+}$ such that*

1. $\mathsf{RPeb}(G_p) = p$*;*
2. *G_p has in-degree at most two and a unique sink; and*
3. *G_p is polynomial-time computable given p, and G_p has at most p^2 nodes.*

Proof. The value of $g^{-1}(h)$, which is the extra pebbling price of pyramids with respect to the height, increases only when $h = g(\Delta)+1$. Therefore the persistent pebbling price of a pyramid increases by 1 unless $h = g(\Delta)+1$, in which case it increases by 2. If $p = h + g^{-1}(h)$ for some $h \in \mathbb{N}$ we let G_p be the pyramid graph of height h. In this way G_p is defined for every $p > 0$, unless $p = h + g^{-1}(h) + 1$ for some $h = g(\Delta)$. In this case we let G_p be the $(h,1)$-teabag which, by Lemmas E.5.5 and E.5.8, has persistent pebbling price $p = h + g^{-1}(h) + 1$. □

We want a polynomial-time computable family of graphs $\{T_r\}_{r\in\mathbb{N}^+}$ with matching visiting price and persistent price, i.e., $\mathsf{RPeb}^V(T_r) = \mathsf{RPeb}(T_r) = r$. The idea is to stack up r appropriately chosen graphs, so that any visiting or persistent pebbling strategy has to spend one pebble per graph.

We will use r graphs from the family $\{G_p\}_{p\in\mathbb{N}^+}$ where each G_p has persistent pebbling price p, as constructed in Lemma E.6.2. The resulting graph is a stack of modified pyramids of increasing sizes. If there is justice in this world, the resulting graph should be called a Christmas Tree; though a graph theorist may have a hard time calling this a "tree".

Construction E.6.3 (Christmas Tree). Let $\{G_p\}_{p\in\mathbb{N}^+}$ be given as in Lemma E.6.2. Given $r \in \mathbb{N}^+$, construct a graph $T_r := (V,E)$ as follows. Its vertex set $V := V^1 \sqcup V^2 \sqcup \cdots \sqcup V^r$ is a disjoint union of r layers, where for $1 \le t < r$ the tth layer is a copy of G_{r-t} with vertices

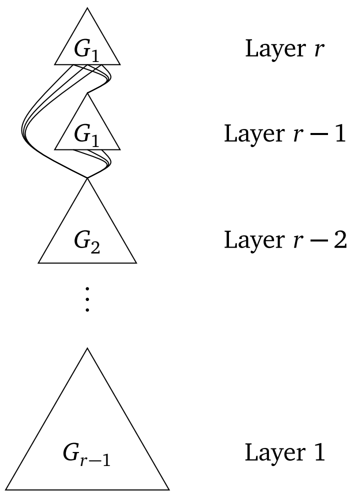


Figure E.10: Illustration of a Christmas Tree in Construction E.6.3.

$V^t := V(G_{r-t})$, and the top-most layer is another copy of G_1 with vertices $V^r := V(G_1)$. Its edge set $E := E_{\text{intra}} \sqcup E_{\text{inter}}$ consists of intra- and inter-layer edges. The intra-layer edges $E_{\text{intra}} := E^1 \sqcup E^2 \sqcup \cdots \sqcup E^r$ come from the corresponding copies of G_p, i.e., for $1 \le t < r$ the edges on the tth layer E^t are copies of $E(G_{r-t})$ and and the edges on the top-most layer E^r are copies of $E(G_1)$. The inter-layer edges E_{inter} connect, for each $1 \le t \le r-2$, the sink of the subgraph at layer t with all sources of the subgraphs at layer $t+1$ and $t+2$, and also connect the sink of the copy of G_1 at layer $r-1$ with the sources of the copy of G_1 at layer r.

Lemma E.6.4 (Christmas Tree). *The family of graphs $\{T_r\}_{r\in\mathbb{N}^+}$ satisfies*

1. $\mathsf{RPeb}^V(T_r) = \mathsf{RPeb}(T_r) = r$;
2. *T_r has in-degree at most two and a unique sink; and*
3. *T_r is polynomial-time computable given r, and T_r has at most r^3.*

Proof. For Item 3, note that each of the r layers has at most r^2 nodes.

For Item 2, if v is a node in T_r we have two cases depending on whether v is some layer's source node or not. If it is, then at most two inter-layer edges from lower layers point to v, and no intra-layer edge does. If v is not a source on any layer then only two intra-layer edges point to it, since all G_p have fan-in at most 2. The only sink of T_r is the sink of layer r.

To see that $\mathsf{RPeb}^S(T_r) \le r-1$, and thus that $\mathsf{RPeb}(T_r) \le r$, persistently pebble the sink node of G_{r-t} on layer t, for t from 1 to $r-1$, keeping only the pebbles on the sinks of the previous layers. To persistently pebble layer t takes $r-t$ pebbles, assuming there is a pebble on each of the sinks of the $t-1$ lower layers, so in total $r-1$ pebbles suffice. Note that G_1 is a single node, hence when the sinks of the lower layers are all pebbled the sink of T_r is surrounded. The bound $\mathsf{RPeb}(T_r) \leq r$ follows by Proposition E.2.1.

To see that $\mathsf{RPeb}^V(T_r) \geq r$, we argue that when visiting layer r there is a v-locked pebble configuration on each of the previous layers (see Definition E.6.1). In particular, any layer with a pebble on the sink has a v-locked configuration and if a configuration is v-locked, then it contains a pebble. Given a pebbling configuration on T_r, for the rest of this proof we say that layer t is *v-locked* if the configuration, restricted to the corresponding subgraph, is v-locked for the subgraph.

Claim E.6.5 (Christmas Tree Locker). *Consider any pebbling that uses less than r pebbles. In such a pebbling, whenever some layer $(t-1)$ and layer t contain some pebbles, for $2 \leq t \leq r$, then layers $1, \ldots, t-2$ are all v-locked.*

Proof. The claim is true for $t = 2$ vacuously, establishing the base case. When $t > 2$, consider a time that layer t starts to have a pebble: a source node on layer t is pebbled, hence there are pebbles on the sink nodes of layers $t-1$ and $t-2$. Thus layers $t-1$ and $t-2$ are v-locked and each has a pebble. Induction hypothesis (on $t-1$) further says that layer η is v-locked for any $1 \leq \eta < t-2$. As long as there are pebbles on layers t and $t-1$, all lower layers remain v-locked: for $1 \leq \eta \leq t-2$, to unlock layer η takes $\mathsf{RPeb}^V(G_{r-\eta}) \geq \mathsf{RPeb}^S(G_{r-\eta}) = r-\eta-1 \geq r-t+1$ pebbles (recall Equation (E.2.3)), but there are $t-1$ pebbles on layers other than η, which cannot be done with less than r pebbles. ☐

Assume for some $r \geq 2$, the sink node of layer r is pebbled using less than r pebbles. When a source node of layer r is pebbled, there is a pebble on the sink node of layer $r-1$. Claim E.6.5 shows that there is a pebble on layer η for $1 \leq \eta \leq r-2$, for a total of r pebbles, contradicting that less than r pebbles are used. This shows $\mathsf{RPeb}^V(T_r) \geq r$ for $r \geq 2$, and the case for $r = 1$ is obvious. In the end we get that

$$r \leq \mathsf{RPeb}^V(T_r) \leq \mathsf{RPeb}(T_r) = \mathsf{RPeb}^S(T_r) + 1 \leq r \tag{E.6.1}$$

by Equation (E.2.3) and Proposition E.2.1, which gives Item 1. ☐

E.6.2 Molding

Given a graph G we want to construct a graph $M(G)$ with a special source s and a single sink, such that any pebbling that visits the sink must go through a configuration with at least $\mathsf{RPeb}^V(M(G))$ pebbles, one of which is on vertex s.

Construction E.6.6 (Molding). Given a graph G, we construct a graph $M(G)$ as follows. For every vertex $v \in V(G)$, we add to $M(G)$ two vertices v_{in} and v_{out}, and a directed edge $(v_{\text{in}}, v_{\text{out}})$. Also, for every edge $(u, v) \in E(G)$, we add to $M(G)$ a corresponding edge

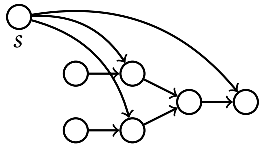


Figure E.11: Example of Construction E.6.6: molding of a pyramid of height 1.

$(u_{\text{out}}, v_{\text{in}})$. Finally we add to $M(G)$ a special new vertex s that we connect to all vertices v_{out}, i.e., for every $v \in V(G)$ we add edge (s, v_{out}) to $M(G)$. Formally, $V(M(G)) := \{s\} \sqcup \{v_{\text{in}}, v_{\text{out}} : v \in V(G)\}$ and $E(M(G)) := E_1 \sqcup E_2 \sqcup E_3$, where $E_1 := \{(v_{\text{in}}, v_{\text{out}}) : v \in V(G)\}$, $E_2 := \{(u_{\text{out}}, v_{\text{in}}) : (u, v) \in E(G)\}$ and $E_3 := \{(s, v_{\text{out}}) : v \in V(G)\}$.

By construction, if G has in-degree at most two and a unique sink then so does $M(G)$.

Lemma E.6.7 (Molding). *Given a graph G, the graph $M(G)$ has the following properties.*

1. $\mathsf{RPeb}\big(M(G)\big) \le \mathsf{RPeb}(G) + 2$*; and*
2. *For any visiting pebbling $\mathcal{P}' = \langle \mathbb{P}'_0, \mathbb{P}'_1, \dots, \mathbb{P}'_\tau \rangle$ of $M(G)$, there is a configuration P'_b (for some $0 \le b \le \tau$) using at least $\mathsf{RPeb}^V(G) + 2$ pebbles and containing s.*

Proof. For Item (1), fix any persistent pebbling $\mathcal{P}$ of G. Simulate the pebbling $\mathcal{P}$ as a persistent pebbling $\mathcal{P}'$ of $M(G)$ as follows. First, pebble the special source s of $M(G)$ in $\mathcal{P}'$. Afterwards, whenever there is a move in $\mathcal{P}$ to pebble a node $v \in V(G)$, make a phase of three moves in $\mathcal{P}'$: pebble v_{in}, pebble v_{out}, unpebble v_{in}. Similarly, whenever there is a move in $\mathcal{P}$ to unpebble a node $v \in V(G)$, make a phase of three moves in $\mathcal{P}'$: pebble v_{in}, unpebble v_{out}, unpebble v_{in}. If the current configuration in $\mathcal{P}$ is $\mathbb{P}$, and the configuration in $\mathcal{P}'$ at the end of a phase is $\mathbb{P}'$, then $\mathcal{P}'$ maintains the invariant that $\mathbb{P}' = \{s\} \cup \{v_{\text{out}} : v \in \mathbb{P}\}$. As a result, $\mathcal{P}'$ is a legal pebbling on $M(G)$: whenever v_{in} is pebbled or unpebbled, all its predecessors $\mathit{pred}(v_{\text{in}}) = \{u_{\text{out}} : u \in \mathit{pred}(v)\}$ have pebbles in $\mathbb{P}'$, since $\mathit{pred}(v)$ have pebbles in $\mathbb{P}$; whenever v_{out} is pebbled or unpebbled, its predecessors s and v_{in} have pebbles. If we add a final move in $\mathcal{P}'$ to unpebble s, then $\mathcal{P}'$ persistently pebbles $M(G)$. Note that whenever $\mathcal{P}$ pebbles or unpebbles $v \in V(G)$ to get to configuration $\mathbb{P}$, the simulating pebbling $\mathcal{P}'$ uses at most two more pebbles to get to $\mathbb{P}'$, namely s and v_{in}. Hence $\mathsf{RPeb}\big(M(G)\big) \le \mathsf{RPeb}(G) + 2$.

For Item (2), we start with a visiting pebbling $\mathcal{P}' = \langle \mathbb{P}'_0, \mathbb{P}'_1, \dots, \mathbb{P}'_\tau \rangle$ of $M(G)$ and we construct a visiting pebbling $\mathcal{P}$ of G. We now define two projection operators that turn pebble configurations for $M(G)$ into configurations for G. Let $\mathit{proj}_{\text{out}}(\mathbb{P}'_t) := \{v \in V(G) : v_{\text{out}} \in \mathbb{P}'_t\}$ be the projection of $\mathbb{P}'_t$ to $V(G)$ via v_{out}, and $\mathit{proj}_{\text{any}}(\mathbb{P}'_t) := \{v \in V(G) : v_{\text{in}} \in \mathbb{P}'_t \text{ or } v_{\text{out}} \in \mathbb{P}'_t\}$ be the projection of $\mathbb{P}'_t$ to $V(G)$ via v_{in} or v_{out}. By definition $\mathit{proj}_{\text{out}}(\mathbb{P}'_t) \subseteq \mathit{proj}_{\text{any}}(\mathbb{P}'_t)$. Whenever a vertex of $M(G)$ is pebbled or unpebbled during a pebbling step from $\mathbb{P}'_t$ to $\mathbb{P}'_{t+1}$,

(i) if the vertex is the special source s, then both $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t) = \mathit{proj}_{\mathrm{out}}(\mathbb{P}'_{t+1})$ and $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) = \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_{t+1})$;

(ii) if the vertex is v_{in} for some vertex $v \in V(G)$, then $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t) = \mathit{proj}_{\mathrm{out}}(\mathbb{P}'_{t+1})$ but $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t)$ may differ from $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_{t+1})$; and

(iii) if the vertex is v_{out} for some vertex $v \in V(G)$, then $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) = \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_{t+1})$ but $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t) \neq \mathit{proj}_{\mathrm{out}}(\mathbb{P}'_{t+1})$.

To construct $\mathcal{P}$, we analyze in order each configuration $\mathbb{P}'_t$ in $\mathcal{P}'$. Depending on how the sequences of $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t)$ and $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t)$ evolve, we may append new configurations to $\mathcal{P}$. In the following η is the index of the last configuration added to $\mathcal{P}$ and t is the configuration of $\mathcal{P}'$ under analysis. We maintain the following invariants:

(a) $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t) \subseteq \mathbb{P}_\eta$; and

(b) for any v in $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) \triangle \mathbb{P}_\eta$ it holds that $\mathit{pred}(v) \subseteq \mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t)$.

Initially at $t = 0$ and $\eta = 0$, $\mathbb{P}'_t = \mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t) = \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) = \mathbb{P}_\eta = \emptyset$, so the invariant holds for $t = 0$. Consider a pebbling move in $\mathcal{P}'$ from $\mathbb{P}'_t$ to $\mathbb{P}'_{t+1}$.

(I) If $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) = \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_{t+1})$ and $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t) = \mathit{proj}_{\mathrm{out}}(\mathbb{P}'_{t+1})$ the construction does not append any new $\mathbb{P}_\eta$ and the invariant is preserved.

(II) If $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) \neq \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_{t+1})$ then we are in case (ii) above, so $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t)$ does not change and some node v_{in} is pebbled or unpebbled. Hence the current configuration $\mathbb{P}'_t \supseteq \mathit{pred}(v_{\mathrm{in}}) = \{u_{\mathrm{out}} : u \in \mathit{pred}(v)\}$, thus $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t) \supseteq \mathit{pred}(v)$. The construction does not append a new $\mathbb{P}_\eta$ and the invariant is preserved.

(III) If $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t) \neq \mathit{proj}_{\mathrm{out}}(\mathbb{P}'_{t+1})$ then we are in case (iii) above, so $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t)$ does not change and some node v_{out} is pebbled or unpebbled. The construction appends to $\mathcal{P}$ the two sequences of moves ("Eras") described below. After each move $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) \triangle \mathbb{P}_\eta$ gets smaller. Note that for any $u \in \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) \triangle \mathbb{P}_\eta$, the invariant gives $\mathit{pred}(u) \subseteq \mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t) \subseteq \mathbb{P}_\eta$, so they can be pebbled or unpebbled in $\mathbb{P}_\eta$.

Unpebble Era while $\mathbb{P}_\eta \setminus \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) \neq \emptyset$, pick any $u \in \mathbb{P}_\eta \setminus \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t)$, and unpebble u in $\mathcal{P}$ (increment $\eta := \eta + 1$ and then set $\mathbb{P}_\eta := \mathbb{P}_{\eta-1} \setminus \{u\}$). Since $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t) \subseteq \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t)$, $u \notin \mathit{proj}_{\mathrm{out}}(\mathbb{P}'_t)$ and the invariant is preserved at time t.

Pebble Era while $\mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) \setminus \mathbb{P}_\eta \neq \emptyset$, pick any $u \in \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) \setminus \mathbb{P}_\eta$, and pebble u in $\mathcal{P}$ (increment $\eta := \eta + 1$ and then set $\mathbb{P}_\eta := \mathbb{P}_{\eta-1} \cup \{u\}$). The invariant is preserved at time t.

At the end of the two sequences, $\mathit{proj}_{\mathrm{out}}(\mathbb{P}'_{t+1}) \subseteq \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_{t+1}) = \mathit{proj}_{\mathrm{any}}(\mathbb{P}'_t) = \mathbb{P}_\eta$, so the invariant now holds also at time $t + 1$.

We complete the proof of Item (2). For any visiting pebbling $\mathcal{P}'$ of $M(G)$, the corresponding pebbling $\mathcal{P}$ is a visiting pebbling of G by invariant (a), thus some constructed configuration $\mathbb{P}_\eta$ has at least $\mathsf{RPeb}^V(G)$ pebbles. The configuration has been appended to $\mathcal{P}$ in case (III) above, and without loss of generality we can assume it is either at the beginning of an "unpebble era" or at the end of a "pebble era", since the number of pebbles in $\mathbb{P}_\eta$ decreases in the former and increases in the latter. Since the beginning of an "unpebble era" other than the first is also the end of a "pebble era", we can furthermore assume the latter. This means that for some t we have $proj_{\text{any}}(\mathbb{P}'_t) = proj_{\text{any}}(\mathbb{P}'_{t+1}) = \mathbb{P}_\eta$, so either the corresponding configuration $\mathbb{P}'_t$ (when v_{out} is unpebbled) or $\mathbb{P}'_{t+1}$ (when v_{out} is pebbled) has at least $\mathsf{RPeb}^V(G) + 2$ pebbles, including s, v_{in} and v_{out}. This completes Item (2). □

E.6.3 Turnpikes

As an application of molding (Construction E.6.6) we show a construction that controls the visiting price of a node, and that allows us to construct gadgets for the components of a quantified boolean formula.

Construction E.6.8 (Turnpike). For any non-negative integer r we define the *turnpike* of toll r from a to b (represented by Fig. E.12) as follows. If $r = 0$ then the turnpike just joins the vertices a and b with an edge (a, b). If $r > 0$ let T_r be the graph having $\mathsf{RPeb}^V(T_r) = \mathsf{RPeb}(T_r) = r$ given by Lemma E.6.4. The turnpike of toll r from a to b is the graph $M(T_r)$, identifying a with the special source s in $M(T_r)$, and identifying b with the unique sink in $M(T_r)$.

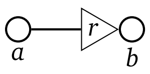


Figure E.12: Turnpike of toll r from a to b.

Let G be any graph that contains a turnpike of toll r from a to b. Call the nodes $\tilde{R} := anc_G(b) \setminus anc_G(a)$ to be *properly* in the turnpike, and call the nodes $R := \tilde{R} \cup \{a\}$ to be in the turnpike. The turnpike construction is sensitive to whether the pebble on node a is counted, i.e., whether the pebbling prices are restricted to R or to $\tilde{R}$.

Lemma E.6.9 (Turnpike). *We have* $\mathsf{RPeb}_R(b) = \mathsf{RPeb}^V_R(b) = r + 2$ *and* $\mathsf{RPeb}_{\tilde{R}}(b) = \mathsf{RPeb}^V_{\tilde{R}}(b) = r + 1$.

Proof. $\mathsf{RPeb}_R(b) \leq r+2$ and $\mathsf{RPeb}_{\tilde{R}}(b) \leq r+1$ by (the proof of) Item (1) of Lemma E.6.7 (since pebbles outside of R or of $\tilde{R}$ are not counted), and $\mathsf{RPeb}^V_R(b) \geq r + 2$ and $\mathsf{RPeb}^V_{\tilde{R}}(b) \geq r + 1$ by Item (2) of Lemma E.6.7. □

The fact that $\mathsf{RPeb}_R(b)$ and $\mathsf{RPeb}_{\check{R}}(b)$ do not differ by more than one holds not only for turnpikes but for any graph.

Lemma E.6.10 (Source Difference). *Consider regions R_1 and R_2 such that $R_1 = R_2 \setminus \{s_2\}$ for some source s_2 of R_2 (i.e., $pred(s_2) \cap R_2 = \emptyset$). We have*

$$\begin{array}{rcccl} \mathsf{RPeb}_{R_1}(v) & \leq & \mathsf{RPeb}_{R_2}(v) & \leq & \mathsf{RPeb}_{R_1}(v)+1 \\ \mathsf{RPeb}^V_{R_1}(v) & \leq & \mathsf{RPeb}^V_{R_2}(v) & \leq & \mathsf{RPeb}^V_{R_1}(v)+1 \\ \mathsf{RPeb}^S_{R_1}(v) & \leq & \mathsf{RPeb}^S_{R_2}(v) & \leq & \mathsf{RPeb}^S_{R_1}(v)+1 \ . \end{array}$$

E.7 PSPACE-Completeness

In this section we give all details of the construction in Theorem E.3.2, restated in the following theorem, and its full proof.

Theorem E.7.1. *Given a quantified 3-CNF ϕ, there is a polynomial-time constructible graph $G(\phi)$ and a polynomial-time computable number $\gamma(\phi)$ such that $\mathsf{RPeb}\big(G(\phi)\big) = \gamma(\phi) + [\![\phi \text{ is FALSE}]\!]$.*

Let $x_1 \ldots x_n$ be the variables of ϕ. We sort the variables from the innermost to the outermost quantifier and denote by ϕ_i the quantified 3-CNF with just the i innermost quantifiers, namely $\phi_0 = \bigwedge_{j=1}^m C_j$, $\phi_i = Q_i x_i \phi_{i-1}$ for $Q \in \{\forall, \exists\}$, and $\phi = \phi_n$. For each ϕ_i we consider the gadget $G(\phi_i)$, where $G(\phi_0)$ is built as defined in Construction E.7.24 and each $G(\phi_i)$ for $i \in [n]$ is built according to either Construction E.7.26 or Construction E.7.31, depending on the i^{th} innermost quantifier. Furthermore we fix the sequence of integers $\{\gamma_i\}_{i=0}^n$ where $\gamma_0 := 2m+7$, $\gamma_i := 3 + \gamma_{(i-1)}$ if the i^{th} innermost quantifier is existential, and $\gamma_i = 5 + \gamma_{(i-1)}$ if the i^{th} innermost quantifier is universal. To analyze the gadgets for the subformulas we need the next definition.

Definition E.7.2. A *region* is an induced subgraph of the final gadget $G(\phi)$ (or of component gadgets as we build up the final gadget). We slightly abuse notation and refer to a region by a subset of vertices; for example, given a subset of vertices $\check{R}$ of a gadget G, we use $\mathsf{RPeb}_{\check{R}}(G)$, $\mathsf{RPeb}^V_{\check{R}}(G)$ and $\mathsf{RPeb}^S_{\check{R}}(G)$ to denote the different pebbling prices over the subgraph of G induced on $\check{R}$.

With these notations and definitions in place, Theorem E.7.1 follows immediately from the next lemma when i is equal to n. In this case ρ and $\check{R}$, as stated in the lemma, are respectively the empty assignment and the full graph $G(\phi)$. In the proof of the lemma we refer to definitions, lemmas and constructions that we will present in full details in the coming subsections.

Lemma E.7.3 (Main Lemma). *Fix an arbitrary $0 \leq i \leq n$ and let*

- *ρ be an assignment to all but the first i variables;*

- *S be the canonical set of ρ according to Definition E.7.11;*
- *$\check{R}$ be the subset of vertices of $G(\phi_i)$ defined as $V(G(\phi_i)) \setminus anc(S)$.*

It holds that

$$\mathit{RPeb}_{\check{R}}\big(G(\phi_i)\big) = \gamma_i + [\![\phi_i{\restriction_\rho} \text{ is FALSE}]\!] \ . \tag{E.7.1}$$

Proof. When $i = 0$, the gadget $G(\phi_0)$ is the CNF gadget from Construction E.7.24 and the base case follows immediately by Lemma E.7.25, where $\beta_m = 2m$. When $i > 0$, consider the two possible extensions of ρ that assign x_i, namely $\rho_0 := \rho \cup \{x_i = 0\}$ and $\rho_1 := \rho \cup \{x_i = 1\}$. Consider also the corresponding canonical sets S_0, S_1 and the regions $\check{R}_0 := V(G(\phi_{(i-1)})) \setminus anc(S_0)$ and $\check{R}_1 := V(G(\phi_{(i-1)})) \setminus anc(S_1)$. By induction it holds that

- $\mathit{RPeb}_{\check{R}_0}\big(G(\phi_{(i-1)})\big) = \gamma_{(i-i)} + [\![\rho_0 \text{ falsifies } \phi_{(i-1)}]\!]$
- $\mathit{RPeb}_{\check{R}_1}\big(G(\phi_{(i-1)})\big) = \gamma_{(i-1)} + [\![\rho_1 \text{ falsifies } \phi_{(i-1)}]\!]$

so the hypothesis of Lemmas E.7.37 and E.7.30 holds. When $\phi_i = \forall x_i \phi_{i-1}$ Lemma E.7.37 gives that

$$\mathit{RPeb}\big(G(\phi_i)\big) = 5 + \gamma_{(i-i)} + [\![\rho_0 \text{ falsifies } \phi_{(i-1)} \text{ or } \rho_1 \text{ falsifies } \phi_{(i-1)}]\!], \tag{E.7.2}$$

and $\phi_i = \exists x_i \phi_{(i-1)}$ Lemma E.7.30 gives that

$$\mathit{RPeb}\big(G(\phi_i)\big) = 3 + \gamma_{(i-i)} + [\![\rho_0 \text{ falsifies } \phi_{(i-1)} \text{ and } \rho_1 \text{ falsifies } \phi_{(i-1)}]\!] \ . \tag{E.7.3}$$

Equations (E.7.2) and (E.7.3) are equivalent, for the respective quantifier type, to Equation (E.7.1). □

E.7.1 Literal Gadget

Lemma E.6.4 allows us to create an edge (u, v) with $\mathit{RPeb}(v) = \mathit{RPeb}(u) + 1 = r + 1$ given any $r \in \mathbb{N}^+$. This is used for constructing gadgets for literals (represented as Figure E.13).

Construction E.7.4 (Literal Gadget). Fix a literal ℓ and an integer $r \in \mathbb{N}^+$. Let T_r be the graph having $\mathit{RPeb}^V(T_r) = \mathit{RPeb}(T_r) = r$ given by Lemma E.6.4. The literal gadget of price r for ℓ, is denoted as $L_\ell(r)$. To construct it, take a copy of T_r and call its sink ℓ'. Then add a new node, named ℓ as the corresponding literal, and add the edge (ℓ', ℓ).

Lemma E.7.5 (Literal Gadget). *In $L_\ell(r)$, $\mathit{RPeb}(\ell) - 1 = \mathit{RPeb}(\ell') = \mathit{RPeb}^V(\ell') = r$.*

Proof. Note that visiting node ℓ' is equivalent to surrounding node ℓ. Hence $\mathit{RPeb}(\ell) - 1 = \mathit{RPeb}^S(\ell) = \mathit{RPeb}^V(\ell') = \mathit{RPeb}(\ell') = r$ by Equation (E.2.3), Proposition E.2.1, and Lemma E.6.4. □

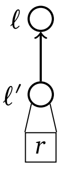


Figure E.13: Literal gadget of price r for ℓ.

We associate pebbling configurations on $L := L_r(\ell)$ with truth value TRUE, FALSE or $*$ (undefined) as follows.

Definition E.7.6 (Literal Position). Fix a literal gadget $L := L_r(\ell)$. Given a pebbling configuration $\mathbb{P}$, say node ℓ' is *v-locked* on $anc_L(\ell')$ if $\mathsf{RPeb}_L(\mathbb{P}) = \mathsf{RPeb}^V(\ell') = r$ (when restricted to $anc_L(\ell')$); that is, if the empty configuration cannot be reached without entering a configuration which has r pebbles. Given a pebbling configuration on L, say literal ℓ is in

- TRUE position, if node ℓ has a pebble and node ℓ' is not v-locked;
- FALSE position, if node ℓ' is v-locked; and
- $*$ position, if node ℓ has no pebble and node ℓ' is not v-locked.

A *transition* of literal ℓ is a change of position for ℓ (for instance from TRUE position to FALSE position, or from FALSE position to $*$ position) due to a pebble move. Finally, we identify certain *canonical* positions with configurations on L as follows:

- the canonical TRUE position is the configuration where only node ℓ has a pebble;
- the canonical FALSE position is the configuration where only node ℓ' has a pebble; and
- the canonical $*$ position is the empty configuration.

Clearly, the canonical TRUE position (resp. canonical FALSE position, canonical $*$ position) is indeed a TRUE position (resp. FALSE position, $*$ position).

Lemma E.7.7 (Literal Transition). *Fix a literal gadget $L := L_r(\ell)$.*

1. *it is impossible to transition directly from $*$ position to TRUE position, and vice versa; and*
2. *at a transition, there are r pebbles on $anc_L(\ell')$.*

Proof. Note that node ℓ' is v-locked on any configuration with a pebble on node ℓ'. To change from $*$ position to TRUE position, node ℓ needs to be pebbled, and hence node ℓ' must have a pebble at some time. At that time, node ℓ' is v-locked, so the configuration is in FALSE position. This gives Item (1).

Hence the only valid transitions are from $*$ position to FALSE position (or its reverse), and from FALSE position to TRUE position (or its reverse). In any of these, node ℓ' needs to switch between locked and unlocked status, which requires a configuration with r pebbles on $anc_L(\ell')$ by definition of v-locked. This gives Item (2). □

E.7.2 Variable Gadget

Let $r_i \in \mathbb{N}^+$ be an integer to be specified later, which is associated with the i^{th} variable x_i.

Inspired by previous works [88, 102, 55] truth values are represented using the gadget in Figure E.14.

Construction E.7.8 (Variable Gadget). For the variable x_i, its variable gadget $G(x_i)$ is constructed as the disjoint union two literal gadgets of price r_i, one for literal x_i and one for literal $\bar{x}_i$.

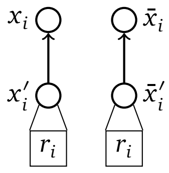


Figure E.14: Variable x_i.

For the gadget $G := G(x_i)$, its nodes are $V(G) := Anc_G(\{x_i, \bar{x}_i\})$.

Definition E.7.9 (Variable Position). Fix a variable gadget $G := G(x_i)$ consisting of literal gadgets $L_1 := L_{r_i}(x_i)$ and $L_0 := L_{r_i}(\bar{x}_i)$. We identify certain *canonical* positions with configurations on G as follows:

- the canonical TRUE position is the configuration where only nodes x_i and $\bar{x}_i'$ have pebbles;
- the canonical FALSE position is the configuration where only nodes x_i' and $\bar{x}_i$ have pebbles; and
- the canonical $*$ position is the empty configuration.

Lemma E.7.10 (Variable Assignment). *Variable x_i can be put into one among canonical TRUE and canonical FALSE positions using at most $r_i + 1$ pebbles.*

Proof. To put variable x_i in canonical TRUE position, persistently pebble node x_i, then persistently pebble $\bar{x}_i'$. It can be done with $r_i + 1$ pebbles by Lemma E.7.5. A symmetric argument shows that x_i can be put in canonical FALSE position with $r_i + 1$ pebbles. □

As we will see in later sections, the design of the quantifier gadgets would ensure that any pebbling strategy would effectively associate truth value via the canonical positions. This motivates the following definition.

Definition E.7.11 (Canonical Nodes). Given a partial assignment $\rho : [n] \to \{$TRUE, FALSE, $*\}$, the canonical nodes of variable x_i under ρ are

- $\{x_i, \bar{x}_i'\}$ if $\rho(i) =$ TRUE;
- $\{x_i', \bar{x}_i\}$ if $\rho(i) =$ FALSE; and
- $\{\}$ if $\rho(i) = *$.

Note that if $\rho(i) \neq *$, then there are two pebbles on the ancestors of the canonical nodes of x_i. For example, if $\rho(i) =$ TRUE, then there is a pebble on $anc(x_i)$ and a pebble on $anc(\bar{x}_i')$.

In general, we consider a partial assignment on variables $\rho : [n] \to \{$TRUE, FALSE, $*\}$ as a partial assignment on literals:

- if variable x_i is assigned TRUE under ρ (i.e., $\rho(i) =$ TRUE), then literal x_i is assigned TRUE and literal $\bar{x}_i$ is assigned FALSE;
- if variable x_i is assigned FALSE under ρ (i.e., $\rho(i) =$ FALSE), then literal x_i is assigned FALSE and literal $\bar{x}_i$ is assigned TRUE;
- if variable x_i undefined under ρ (i.e., $\rho(i) = *$), then literal x_i is $*$ and literal $\bar{x}_i$ is $*$.

As we will argue later, the design of the quantifier gadgets would ensure that "invalid variable assignments" would not be a problem: for example, the two literals of the same variable cannot be put into the TRUE position at the same time (for instance Claim E.7.29); also, it does not help to put the two literals of the same variable into the FALSE position at the same time, and each variable will be assigned eventually (Lemmas E.7.28 and E.7.34).

The canonical nodes (of a partial assignment) are useful for defining certain *regions* over different component gadgets in the overall construction.

E.7.3 Clause Gadget

Let $\beta_j \geq 2$ be an integer to be specified later, which is associated with the j^{th} clause C_j. The gadget for the j^{th} clause, $C_j = \ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3}$, uses as a component the turnpike gadget which is described in Construction E.6.8. Its skeleton is shown in Figure E.15 (the literal gadgets are simplified in Figure E.15 for a cleaner diagram). Assume that the literals $\ell_{j,1}, \ell_{j,2}, \ell_{j,3}$ are over distinct variables.

Construction E.7.12 (Clause Gadget). Assume that for each variable x_i, $1 \leq i \leq n$ we have the corresponding variable gadget $G(x_i)$, i.e., two literal gadgets for x_i and $\bar{x}_i$. For the j^{th} clause C_j, its clause gadget $G(C_j)$ is constructed as follows. Create nodes $a_j, b_j, c_j, u_j, v_j, p_j$, and edges

$$(a_j, u_j), (b_j, u_j), (b_j, v_j), (c_j, v_j), (u_j, p_j), (v_j, p_j). \tag{E.7.4}$$

Finally, add three turnpikes of toll β_j, from $\ell_{j,1}$ to a_j, from $\ell_{j,2}$ to b_j, and from $\ell_{j,3}$ to c_j (where the nodes $\ell_{j,1}, \ell_{j,2}, \ell_{j,3}$ are the ones from the corresponding literal gadgets).

Note that in Figure E.15 the six nodes $\ell_{j,1}$, $\ell'_{j,1}$, $\ell_{j,2}$, $\ell'_{j,2}$, $\ell_{j,3}$, and $\ell'_{j,3}$ come from the variable gadgets corresponding to the variables in literals $\ell_{j,1}$, $\ell_{j,2}$ and $\ell_{j,3}$. Recall the definitions of canonical nodes in Definition E.7.11. For example, if literal $\ell_{j,1}$ is in TRUE position, literals $\ell_{j,2}, \ell_{j,3}$ FALSE position, then their canonical nodes are $\ell_{j,1}, \ell'_{j,2}, \ell'_{j,3}$.

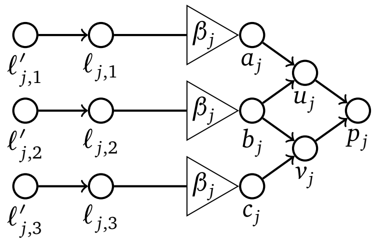


Figure E.15: Clause j.

In this subsection, focus on the gadget $G := G(C_j)$ constructed for the j^{th} clause C_j. The gadget G behaves like a disjunction in the sense that at least one literal is assigned TRUE if, and only if, $\beta_j + 3$ additional pebbles are needed to surround p_j.

Lemma E.7.13 (True Clause, upper bound). *Fix a partial assignment ρ. Assume none of the literals $\ell_{j,1}, \ell_{j,2}, \ell_{j,3}$ is assigned $*$, and at least one of them is assigned TRUE. Let S_j be their canonical nodes. Consider the region $R_j := anc_G(p_j) \setminus anc_G(S_j)$ beyond the canonical nodes. Then $\mathsf{RPeb}^S_{R_j}(p_j) \leq \beta_j + 3$.*

Proof. Note that the j^{th} clause gadget is symmetric in the three literals $\ell_{j,1}, \ell_{j,2}, \ell_{j,3}$ when restricting attention to $Anc_G(\{a_j, b_j, c_j\}) \setminus Anc^*_G(\{\ell'_{j,1}, \ell'_{j,2}, \ell'_{j,3}\})$. If at least one of the literals $\ell_{j,1}, \ell_{j,2}, \ell_{j,3}$ is assigned TRUE, we claim that it takes at most $\beta_j + 3$ pebbles over R_j to leave pebbles only on $\{a_j, b_j, c_j\}$; afterwards p_j can be surrounded by pebbling u_j and v_j (note that $\beta_j + 3 \geq \underbrace{2}_{u_j,v_j} + \underbrace{3}_{a_j,b_j,c_j} = 5$).

To prove the claim, note that if $\ell_{j,1}$ is assigned TRUE, then the intersection of R_j with the turnpike from $\ell_{j,1}$ to a_j is precisely the nodes *properly* in the turnpike; if $\ell_{j,1}$ is assigned FALSE, then the intersection is precisely the nodes in the turnpike. This holds similarly for literals $\ell_{j,2}$ and $\ell_{j,3}$. By symmetry over $Anc_G(\{a_j, b_j, c_j\}) \setminus Anc^*_G(\{\ell'_{j,1}, \ell'_{j,2}, \ell'_{j,3}\})$, assume $\ell_{j,1}$ is assigned TRUE, and each of $\ell_{j,2}$, $\ell_{j,3}$ is assigned TRUE or FALSE. Consider the following strategy to place three pebbles on $\{a_j, b_j, c_j\}$, where pebbles outside of R_j are not counted:

1. Persistently pebble the turnpike from $\ell_{j,2}$ to b_j. It takes at most $\beta_j + 2$ pebbles over R_j by Lemma E.6.9.

2. Persistently pebble the turnpike from $\ell_{j,3}$ to c_j. Now only b_j and c_j have pebbles over R_j. It takes at most $\underbrace{\beta_j + 2}_{anc(c_j)} + \underbrace{1}_{b_j} = \beta_j + 3$ pebbles over R_j by Lemma E.6.9.

3. Persistently pebble the turnpike from $\ell_{j,1}$ to a_j. Since $\ell_{j,1}$ is outside of R_j, it takes at most $\beta_j + 1$ pebbles over the intersection of the turnpike and R_j by Lemma E.6.9, for a total of $\underbrace{\beta_j + 1}_{anc(a_j)} + \underbrace{2}_{b_j,c_j} = \beta_j + 3$ pebbles. Now a_j, b_j and c_j have pebbles. □

Lemma E.7.14 (False Clause, lower bound). *Fix a partial assignment ρ. Assume all of the literals $\ell_{j,1}$, $\ell_{j,2}$ and $\ell_{j,3}$ are assigned FALSE. Let $S_j := \{\ell'_{j,1}, \ell'_{j,2}, \ell'_{j,3}\}$ be their canonical nodes. Consider the region $R_j := anc_G(p_j) \setminus anc_G(S_j)$ beyond the canonical nodes. Then $\mathsf{RPeb}^S_{R_j}(p_j) \geq \beta_j + 4$.*

Proof. Note that G consists of a pyramid whose sources are attached to three turnpikes. Since all literals are assigned FALSE, the intersection of R_j with each of the turnpike is precisely the nodes in the turnpike.

Fix an induced subgraph $F \subseteq G$ (for instance $F =$ the turnpike from $\ell_{j,1}$ to a_j) having a unique sink. Say a pebbling configuration on F is *v-locked* if $\mathsf{RPeb}_F(\mathbb{P}) = \mathsf{RPeb}^V(F)$, that is if in order to reach the empty configuration it is necessary to pass through a configuration with $\mathsf{RPeb}^V(F)$ pebbles. In particular, any configuration with a pebble on the sink of F is v-locked on F. Also, if a configuration is v-locked on F, then there is a pebble on F. Given a pebbling configuration on G, say a_j (resp. b_j, c_j) is v-locked if the configuration is v-locked on the turnpike from $\ell_{j,1}$ to a_j (resp. from $\ell_{j,2}$ to b_j, from $\ell_{j,3}$ to c_j).

With locking in mind, consider a "projected" configuration on the pyramid defined as follows. Given a pebbling configuration $\mathbb{P}$ on G, its projection to the pyramid is $proj(\mathbb{P}) := \big(\{u_j, v_j, p_j\} \cap \mathbb{P}\big) \cup \big\{t \in \{a_j, b_j, c_j\} : t \text{ is v-locked under } \mathbb{P}\big\}$. Note that given a strategy on G, its (configuration-wise) projection to the pyramid is a legal strategy on the pyramid.

We are interested in the truncated paths $\breve{\pi}$ on the pyramid (i.e., source to sink paths excluding the sink, which are in bijection to the edges (a_j, u_j), (b_j, u_j), (b_j, v_j), (c_j, v_j)), and in particular whether they are blocked under $\mathit{proj}(\mathbb{P})$. A truncated path $\breve{\pi}$ is *blocked* under $\mathit{proj}(\mathbb{P})$ if $\breve{\pi} \cap \mathit{proj}(\mathbb{P}) \neq \emptyset$.

Consider a strategy on G to surround p_j. Its projection to the pyramid is a strategy on the pyramid to surround p_j. At the beginning, all truncated paths on the pyramid are not blocked; at the end, all truncated paths are blocked. Consider the first time that all truncated paths on the pyramid are blocked: in the strategy projected on the pyramid, this must be the result of pebbling a source node a_j, b_j, or c_j. By symmetry (in the rest of this argument), assume a_j is pebbled, then there are at least two more pebbles on the pyramid in the projected configuration. Since a_j is being pebbled in the projected strategy, a_j is getting v-locked on G (i.e., restricting attention to the turnpike from $\ell_{j,1}$ to a_j, there are as many pebbles as the visiting price of the turnpike), accounting for $\beta_j + 2$ pebbles in the intersection of R_j and the turnpike to a_j by Lemma E.6.9. The two other pebbles in the projected strategy each account for one more pebble over R_j, for a total of $\beta_j + 4$ pebbles. □

The lower bound shown in Lemma E.7.14 holds for clauses which are falsified. We now prove a weaker lower bound on the surrounding price for the satisfied clauses, which matches the upper bound.

Lemma E.7.15 (Any Clause, lower bound). *Fix a partial assignment ρ. Assume none of literals $\ell_{j,1}$, $\ell_{j,2}$, or $\ell_{j,3}$ is assigned $*$. Let S_j be their canonical nodes. Consider the region $R_j := \mathit{anc}_G(p_j) \setminus \mathit{anc}_G(S_j)$ beyond the canonical nodes. Then $\mathsf{RPeb}^S_{R_j}(p_j) \geq \beta_j + 3$.*

Proof. Follow the proof of Lemma E.7.14 to define v-locked, projection to the pyramid, truncated paths on the pyramid, and blocking on the pyramid. The only difference is that, since some literal can be assigned TRUE, the intersection of R_j with some of the turnpike can be the nodes *properly* in the turnpike.

Consider a strategy on G to surround p_j. Its projection to the pyramid is a strategy on the pyramid to surround p_j. As in the proof of Lemma E.7.14, consider the first time that all truncated paths on the pyramid are blocked, which in the projected strategy must be the result of pebbling a source node, say, a_j. And there are at least two more pebbles on the pyramid in the projected configuration. Since a_j is being pebbled in the projected strategy, a_j is getting v-locked on G (i.e., restricting attention to the turnpike from $\ell_{j,1}$ to a_j, there are as many pebbles as the visiting price of the turnpike), accounting for $\beta_j + 1$ pebbles in the intersection of R_j and the turnpike to a_j by Lemma E.6.9 (note that node $\ell_{j,1}$ may be outside of R_j if literal $\ell_{j,1}$ is in TRUE position). The two other pebbles in the projected strategy each account for one more pebble over R_j, for a total of $\beta_j + 3$ pebbles. □

Assuming that the literal gadgets are in the position corresponding to ρ, we represent "whether ρ falsifies C_j" through *an increase* in the persistent price of the sink of the corresponding gadget, i.e., if ρ *satisfies* C_j, then the persistent price would be a certain number ($\beta_j + 4$); but if ρ *falsifies* C_j, then the persistent price would be one plus that number ($\beta_j + 5$). The difference in pebbling prices can be succintly expressed using the Iverson bracket notation $[\![\rho \text{ falsifies } C_j]\!]$.

Corollary E.7.16 (Clause Gadget). *Fix a partial assignment ρ. Assume none of literals $\ell_{j,1}$, $\ell_{j,2}$, or $\ell_{j,3}$ is assigned $*$. Let S_j be their canonical nodes. Consider the region $R_j := anc_G(p_j) \setminus anc_G(S_j)$ beyond the canonical nodes. Then $\mathsf{RPeb}_{R_j}(p_j) = \beta_j + 4 + [\![\rho \text{ falsifies } C_j]\!]$.*

Proof. If ρ satisfies C_j, i.e., at least one literal is assigned TRUE, then $\mathsf{RPeb}_{R_j}(p_j) = \beta_j + 4$ because persistent price is one plus surrounding price (Proposition E.2.1), and the upper bound (Lemma E.7.13) matches the lower bound (Lemma E.7.15). If ρ falsifies C_j, i.e., all literals are assigned FALSE, then $\mathsf{RPeb}_{R_j}(p_j) = \beta_j + 5$ because it has to increase (Lemma E.7.14) but not by more than one (Lemma E.6.10). □

E.7.4 Conjunction Gadget

To construct a gadget for the conjunction of m clauses, it suffices to repeatedly compose a gadget for the conjunction two smaller gadgets, using the *conjunction gadget* represented in Figure E.16.

Construction E.7.17 (Conjunction Gadget). Assume two gadgets G_1 and G_2 with unique sinks are constructed. Construct the *conjunction gadget* of weight r of G_1 and G_2, denoted $\Lambda_r(G_1, G_2)$, as follows. Call z_1 the sink of G_1, z_2 the sink of G_2. Construct nodes d_1, d_2, d_3, d_4, e, and edges (d_1, d_3), (d_2, d_3), (d_4, e). Add a turnpike of toll r from z_1 to d_1, a turnpike of toll $r - 1$ from z_2 to d_2, and a turnpike of toll $r - 2$ from d_3 to d_4.

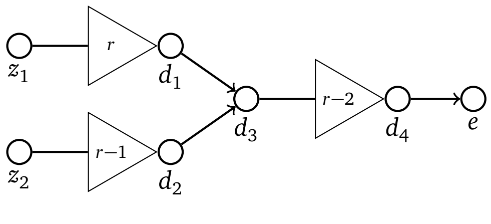


Figure E.16: Conjunction gadget of weight r of G_1 and G_2.

For the gadget $G := \Lambda_r(G_1, G_2)$, the nodes in the gadget are $V(G) = anc(e) \setminus \big(anc^*(z_1) \cup anc^*(z_2)\big)$. We want to analyze pebbling prices restricted to a certain region $\check{R}$ (in the final gadget containing the conjunction gadgets) which will be a superset of the nodes of G.

Lemma E.7.18 (True Conjunction). *Fix a region $\check{R}$ where $\check{R} \supseteq V(G)$. If $\mathsf{RPeb}_{\check{R}}(z_1) \leq r+1$ and $\mathsf{RPeb}_{\check{R}}(z_2) \leq r$, then $\mathsf{RPeb}^S_{\check{R}}(e) \leq r+2$.*

Proof. Consider the following strategy to visit d_4 (equivalently, surround e) using at most $r+2$ pebbles:

- Persistently pebble z_1, persistently pebble the turnpike from z_1 to d_1, persistently unpebble z_1. Now d_1 has a pebble. Over $\check{R}$, the first sub-step takes at most $r+1$ pebbles, the second sub-step takes at most $r+2$ pebbles by Lemma E.6.9, and the third sub-step takes at most $\underbrace{(r+1)}_{anc(z_1)} + \underbrace{1}_{d_1} = r+2$ pebbles.

- Persistently pebble z_2, persistently pebble the turnpike from z_2 to d_2, persistently unpebble z_2. Now d_1 and d_2 have pebbles. Over $anc(d_2) \cap \check{R}$, the first sub-step takes at most r pebbles, the second sub-step most $(r-1)+2 = r+1$ pebbles by Lemma E.6.9, and the third sub-step takes at most $\underbrace{r}_{anc(z_2)} + \underbrace{1}_{d_2}$ pebbles. Over $\check{R}$, this step takes at most $\underbrace{r+1}_{anc(d_2)} + \underbrace{1}_{d_1} = r+2$ pebbles.

- Pebble d_3, persistently pebble the turnpike from d_3 to d_4. Now d_1, d_2, d_3, d_4 have pebbles. Over $\check{R}' := \big(anc(e) \cap \check{R}\big) \setminus \big(anc(d_1) \cup anc(d_2)\big)$, the first sub-step takes 1 pebble, the second sub-step takes at most $(r-2)+2 = r$ pebbles by Lemma E.6.9. Over $\check{R}$, this step takes at most $\underbrace{r}_{\check{R}'} + \underbrace{2}_{d_1,d_2}$ pebbles. □

Lemma E.7.19 (Any Conjunction). *Fix a region $\check{R}$ where $\check{R} \supseteq V(G)$. We have the following:*

1. *$\mathsf{RPeb}^S_{\check{R}}(e) \geq r+2$; and*

2. *if $\mathsf{RPeb}_{\check{R}}(z_1) \leq r+2$ and $\mathsf{RPeb}_{\check{R}}(z_2) \leq r+1$, then $\mathsf{RPeb}^S_{\check{R}}(e) \leq r+3$.*

Proof. For Item (1), note that any strategy to surround e must visit d_1, and $\mathsf{RPeb}^V_{\check{R}}(d_1) \geq r+2$ by Lemma E.6.9.

Item (2) follows from the proof of Lemma E.7.18 to visit d_4 (equivalently, surround e):

1. persistently pebble z_1, persistently pebble the turnpike from z_1 to d_1, persistently unpebble z_1. Now d_1 has a pebble. This step takes at most $r+3$ pebbles over $\check{R}$.

2. persistently pebble z_2, persistently pebble the turnpike from z_2 to d_2, persistently unpebble z_2. Now d_1 and d_2 have pebbles. This step takes at most $r+3$ pebbles over $\check{R}$.

3. pebble d_3, persistently pebble the turnpike from d_3 to d_4. Now d_1, d_2, d_3, d_4 have pebbles. This step takes at most $r+2$ pebbles over $\check{R}$. □

Lemma E.7.20 (False Conjunction). *Fix a region $\check{R}$ where $\check{R} \supseteq V(G)$. We have the following:*

1. *if $\mathsf{RPeb}_{\check{R}}(z_1) \geq r+2$, then $\mathsf{RPeb}^S_{\check{R}}(e) \geq r+3$; and*

2. *if $\mathsf{RPeb}_{\check{R}}(z_2) \geq r+1$, then $\mathsf{RPeb}^S_{\check{R}}(e) \geq r+3$.*

Proof. Fix a turnpike $T \subseteq G$, say from s to z (for instance if T is the turnpike from z_1 to d_1, then $s = z_1$ and $z = d_1$). Say a pebbling configuration on T is *s-locked* if the pebbles cannot be cleared without using $\mathsf{RPeb}^V_{\check{R}}(T)$ pebbles including one on s (when restricted to T); that is, if the empty configuration cannot be reached without entering a configuration which has $\mathsf{RPeb}^V_{\check{R}}(T)$ pebbles and contains s. In particular, any configuration with a pebble on the sink of T is s-locked on T by Lemmas E.6.7 and E.6.9. Also, if a configuration is s-locked on T, then there is a pebble on some node *properly* in the turnpike (there is a pebble on $V(T) \setminus \{s\}$). Given a pebbling configuration on G, say d_1 (resp. d_2, d_4) is s-locked if the configuration is s-locked on on the turnpike from z_1 to d_1 (resp. from z_2 to d_2, from d_3 to d_4).

Fix an induced subgraph $F \subseteq G$ having a unique sink v (for instance $F = anc(z_1)$ and $v = z_1$). Say a pebbling configuration on F is *p-locked* if $\mathsf{RPeb}_F(\mathbb{P}) = \mathsf{RPeb}_{\check{R}}(F)$; this is if the pebbles cannot be cleared without using $\mathsf{RPeb}_{\check{R}}(F)$ pebbles (when restricted to F). In particular, the configuration with just a single pebble on v over $\check{R}$ is p-locked on F. Also, if a configuration is p-locked on F, then there is a pebble on $F \cap \check{R}$. Given a pebbling configuration on G, say z_1 (resp. z_2) is *p-locked* if the configuration is p-locked on $anc(z_1)$ (resp. $anc(z_2)$).

Claim E.7.21 (s-locked implies p-locked). *Fix any turnpike T on G, say of toll r from s to z. Assume $\mathsf{RPeb}_{\check{R}}(s) \geq r+2$. In any pebbling that uses at most $r+2$ pebbles over $anc(z) \cap \check{R}$, if z is s-locked then s is p-locked.*

Proof. Assume z starts to get s-locked. By definition of s-locked, there are $r+2$ pebbles on the turnpike T, and s has one of the pebbles. Since at most $r+2$ pebbles are used over $\check{R}$, only s has pebble over $anc(s) \cap \check{R}$, so s is p-locked. Until z is not s-locked, there is one pebble *properly* in the turnpike (i.e., over $V(T) \setminus \{s\}$). Since unlocking s requires $\mathsf{RPeb}_{\check{R}}(s) \geq r+2$ pebbles over $anc(s) \cap \check{R}$, until z stops being s-locked, s remains p-locked. □

Fix a strategy to surround node e, which at some time t_3 must pebble or unpebble d_3. At time t_3, both node d_1 and node d_2 have pebbles, hence both node d_1 and node d_2 are s-locked. At the beginning, both node d_1 and node d_2 are not s-locked. Let t_1 (resp. t_2) be the earliest time before time t_3 such that node d_1 (resp. d_2) remains s-locked between time t_1 and time t_3. Thus node d_1 (resp. d_2) is s-locked at time t_1 (resp. t_2).

Claim E.7.22 (s-locked). *If the pebbling uses at most $r+2$ pebbles over $\check{R}$ to surround e, then*

(i) $t_1 < t_2$*; and*

(ii) at time t_2*, there is exactly one pebble over* $anc(d_1) \cap \check{R}$*.*

Proof. For Item (i), if $t_2 < t_1$, then at time t_1 there is a pebble on the turnpike from z_2 to d_2 (as node d_2 is already s-locked), and there are $r+2$ pebbles on the turnpike from z_1 to d_1 by definition of s-locked, for a total of $r+3$ pebbles over $\check{R}$.

For Item (ii), there is at least one pebble on the turnpike from z_1 to d_1 as node d_1 is already s-locked, and there is at most one pebble over $anc(d_1) \cap \check{R}$ since there are at least $(r-1)+2 = r+1$ pebbles on the turnpike from z_2 to d_2 by definition of s-locked, and we assumed that at most $r+2$ pebbles are used over $\check{R}$. □

For Item (1), assume $\mathsf{RPeb}_{\check{R}}(z_1) \geq r+2$. If at most $r+2$ pebbles are used over $\check{R}$ to surround node e, then Claim E.7.22 shows that at time t_2, node d_1 is s-locked (as $t_1 < t_2$), and there is exactly one pebble on $anc(d_1) \cap \check{R}$. Since $\mathsf{RPeb}_{\check{R}}(z_1) \geq r+2$ and the turnpike from z_1 to d_1 has toll r, Claim E.7.21 says that when node d_1 is s-locked (which is the case at time t_2), node z_1 is p-locked. Therefore at time t_2, there are two pebbles over $anc(d_1) \cap \check{R}$: one *properly* on the turnpike from z_1 to d_1 (since d_1 is s-locked); and one on $anc(z_1) \cap \check{R}$ (since z_1 is p-locked). This contradiction shows that $\mathsf{RPeb}^S_{\check{R}}(e) \geq r+3$.

For Item (2), assume $\mathsf{RPeb}_{\check{R}}(z_2) \geq r+1$. Fix a strategy using at most $r+2$ pebbles over $\check{R}$ to surround node e, i.e., to visit node d_4. At the end, node d_4 is s-locked; at the beginning, node d_4 is not s-locked. Let t_4 be the earliest time such that node d_4 remains s-locked since t_4 until the end. Thus node d_4 is s-locked at time t_4.

Redefine t_3 if necessary, assume it is the last time before t_4 such that node d_3 is pebbled or unpebbled. Then time t_1 and time t_2 are defined (as above) relative to this t_3, giving $t_1 < t_2 < t_3 < t_4$ (the first inequality is by Claim E.7.22).

Note that at time t_3, node d_3 is being pebbled: to see this, we know that at time t_4, the turnpike from d_3 to d_4 is being s-locked, so there is a pebble on node d_3 by definition of s-locked. Since there is no pebble move on node d_3 after time t_3 and before time t_4, it follows that d_3 is being pebbled at time t_3, and there is a pebble on node d_3 between time t_3 and time t_4.

We know that both node d_1 and node d_2 are s-locked at time t_3. In fact, they remain s-locked between time t_3 and time t_4: to see this, note that to make node d_1 not s-locked takes $r+2$ pebbles over $anc(d_1) \cap \check{R}$, but there are two pebbles outside this region (one on the turnpike from z_2 to d_2, and one on d_3), which cannot be done with at most $r+2$ pebbles over $\check{R}$. Likewise, note that to make node d_2 not s-locked takes $(r-1)+2 = r+1$ pebbles over $anc(d_2) \cap \check{R}$, but there are two pebbles outside this region (one on the turnpike from z_1 to d_1, and one on d_3), which cannot be done with at most $r+2$ pebbles over $\check{R}$. Therefore, node d_1 is s-locked from time t_1 to time t_4, and node d_2 is s-locked from time t_2 to time t_4.

At time t_1, the turnpike from z_1 to d_1 is getting s-locked, so by definition of s-locked there are $r+2$ pebbles on the turnpike. By assumption, at most $r+2$ pebbles are used over $\check{R}$, so there is no pebble over $anc(d_2) \cap \check{R}$ at time t_1. Restrict attention to the sub-strategy $\mathcal{P}'$ between time t_1 and time t_4. In the sub-strategy $\mathcal{P}'$, node d_1 remains s-locked, so at most $(r+2)-1 = r+1$ pebbles can be used over $anc(d_2) \cap \check{R}$. Since $\mathsf{RPeb}_{\check{R}}(z_2) \geq (r-1)+2$ and the turnpike from z_1 to d_1 has toll $r-1$, Claim E.7.21 says that when node d_2 is s-locked, node z_2 is p-locked. At time t_4,

- node d_1 is s-locked, so there is a pebble properly in the turnpike from z_1 to d_1;
- node d_2 is s-locked, hence node z_2 is p-locked, so there is a pebble properly in the turnpike from z_2 to d_2, and a pebble over $anc(z_2) \cap \check{R}$; and
- there are $(r-2)+2 = r$ pebbles in the turnpike from d_3 to d_4.

This accounts for $1+2+r = r+3$ pebbles over $\check{R}$. This contradiction shows that $\mathsf{RPeb}^{S}_{\check{R}}(e) \geq r+3$. □

Recall that we are going to represent the unsatisfiability of a clause by increased persistent prices, i.e., let q_j be the condition that clause C_j is *satisfied*, and $\bar{q}_j$ be its negation, then $\mathsf{RPeb}_{\check{R}_j}(p_j) = \beta_j + 4 + [\![\bar{q}_j]\!]$ by Corollary E.7.16.

Corollary E.7.23 (Conjunction Gadget). *Fix a region $\check{R}$ where $\check{R} \supseteq V(G)$. Assume that for some conditions q_1 and q_2 if holds that $\mathsf{RPeb}_{\check{R}}(z_1) = r+1+[\![\bar{q}_1]\!]$ and $\mathsf{RPeb}_{\check{R}}(z_2) = r+[\![\bar{q}_2]\!]$. Then $\mathsf{RPeb}_{\check{R}}(e) = r+3+[\![\overline{q_1 \wedge q_2}]\!]$.*

Proof. If both q_1 and q_2 are TRUE, then $\mathsf{RPeb}_{\check{R}}(e) = r+3$ because persistent price is one plus surround price (Proposition E.2.1), and the upper bound (Lemma E.7.18) matches the lower bound (Item (1) of Lemma E.7.19). If q_1 or q_2 is FALSE, then $\mathsf{RPeb}_{\check{R}}(e) = r+4$ because the lower bound increases (Lemma E.7.20) to match the new upper bound (Item (2) of Lemma E.7.19). □

E.7.5 CNF Gadget

Assume $\Gamma = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ is a conjunction of m clauses. Let $\Gamma_k := \bigwedge_{1\leq j\leq k} C_j$ be the conjunction of the first k clauses. We will construct a gadget $F_k := G(\Gamma_k)$ for Γ_k with increasing k by successive conjunction of two smaller gadgets, then the CNF gadget for Γ is $G(\Gamma) = G(\Gamma_m)$.

For $1 \leq j \leq m$, let $\beta_j := 2j$, then $\beta_j \geq 2$.

Construction E.7.24 (CNF Gadget). Assume for each clause C_j, $1 \leq j \leq m$, a clause gadget $G(C_j)$ is constructed. Let $\Gamma_k := \bigwedge_{1\leq j\leq k} C_j$ be the conjunction of the first k clauses when $0 \leq k \leq m$. Construct a partial CNF gadget F_k for increasing k as follows. Construct $F_0 := T_7$ as the graph with $\mathsf{RPeb}^{V}(T_7) = \mathsf{RPeb}(T_7) = 7$ given by Lemma E.6.4. Then for $1 \leq k \leq m$, construct $F_k := \Lambda_{\beta_k}\big(F_{k-1}, G(C_k)\big)$ be the conjunction gadget of weight β_k of the previous partial CNF gadget (F_{k-1}) and the gadget of clause k ($G(C_k)$). The CNF gadget $G(\Gamma)$ for Γ is F_m.

For the gadget $G := G(\Gamma)$, the nodes in the gadget $V(G)$ contains the nodes of $F_0 = T_7$, the nodes of all clause gadgets $G(C_j)$, and the nodes of all intermediate conjunction gadgets.

Lemma E.7.25 (CNF Gadget). *Let ρ be an assignment, and let S be the canonical nodes in all variable gadgets according to ρ (Definition E.7.11). Let $\check{R} := V(G) \setminus anc(S)$ be the region beyond the canonical nodes of ρ. Then $\mathsf{RPeb}_{\check{R}}(F_m) = \beta_m + 7 + [\![\rho \text{ falsifies } \Gamma]\!]$.*

Proof. For $1 \le j \le m$, let q_j be the condition that clause j (C_j) is satisfied by ρ. For $0 \le k \le m$, let q'_k be the condition that the first k clauses (Γ_k) are satisfied by ρ. We show by induction that $\mathsf{RPeb}_{\check{R}}(F_k) = \beta_k + 7 + [\![\bar{q}'_k]\!]$.

When $k = 0$, Γ_k is satisfied vacuously, so q'_0 is TRUE. The base case holds as $\mathsf{RPeb}_{\check{R}}(F_0) = \mathsf{RPeb}(T_7) = 7$.

For the general case $1 \le k \le m$, plug $r = \beta_k + 4$ into Corollary E.7.23. Since induction hypothesis gives $\mathsf{RPeb}_{\check{R}}(F_{k-1}) = \beta_{k-1} + 7 + [\![\bar{q}'_{k-1}]\!] = \beta_k + 5 + [\![\bar{q}'_{k-1}]\!]$, and Corollary E.7.16 gives $\mathsf{RPeb}_{\check{R}}\big(G(C_k)\big) = \beta_k + 4 + [\![\bar{q}_k]\!]$, it follows that $\mathsf{RPeb}_{\check{R}}(F_k) = \beta_k + 7 + [\![\bar{q}'_k]\!]$, because $q'_k = q'_{k-1} \wedge q_k$. □

E.7.6 Existential Quantifier Gadget

Assume that we already have the gadget $G(\phi_{i-1})$ and that the ith inner-most quantifier is existential, i.e., $Q_i = \exists$. This quantifier refers to x_i, we set the parameter $r_i := \gamma_i - 2$ for the corresponding variable gadget. We construct $G(\phi_i)$ as follows.

Construction E.7.26 (Existential Quantifier Gadget). Let q_{i-1} denote the sink of the previous quantifier gadget $G(\phi_{i-1})$. Construct nodes f_i, g_i, q_i, and edges (x_i, g_i), $(\bar{x}_i, g_i)$, (f_i, q_i), (g_i, q_i). Add a turnpike of toll $\gamma_i - 5$ from q_{i-1} to f_i.

For the gadget $G := G(\phi_i)$, the nodes in the gadget $V(G)$ contains the nodes in the previous gadget $V\big(G(\phi_{i-1})\big)$, the new nodes f_i, g_i, q_i and the nodes in the turnpike from q_{i-1} to f_i.

Lemma E.7.27 (Existential Upper Bound). *Assume Lemma E.7.3 holds for $i - 1$.*

1. *If $\phi_i \restriction_\rho$ is TRUE, then $\mathsf{RPeb}^S_{\check{R}}(q_i) \le \gamma_i - 1$.*

2. *If $\phi_i \restriction_\rho$ is FALSE, then $\mathsf{RPeb}^S_{\check{R}}(q_i) \le \gamma_i$.*

Proof. For Item (1), by assumption $\phi_i \restriction_\rho = \exists x_i \phi_{i-1} \restriction_\rho$ is TRUE, so there is an assignment of x_i to TRUE or to FALSE to satisfy ϕ_{i-1}, i.e., $\phi_{i-1} \restriction_{\rho_1}$ is TRUE or $\phi_{i-1} \restriction_{\rho_0}$ is TRUE. Assume the former by symmetry. Since $\phi_{i-1} \restriction_{\rho_1}$ is TRUE, by the assumption that Lemma E.7.3 holds for $i - 1$, we have $\mathsf{RPeb}_{\check{R}_1}(q_{i-1}) = \gamma_{i-1} = \gamma_i - 3$. Consider the following strategy to surround q_i with at most $\gamma_i - 1$ pebbles over $\check{R}$:

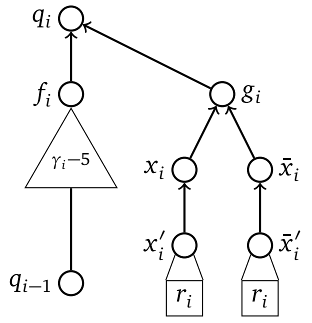


Figure E.17: Existentially quantified variable $\exists x_i$.

(i) Put x_i into the canonical TRUE position with $r_i+1 = \gamma_i-1$ pebbles by Lemma E.7.10. Now node x_i and node $\bar{x}'_i$ have pebbles;

(ii) Persistently pebble node q_{i-1} using at most $\gamma_i - 3$ pebbles over the new region $\check{R}_1$. Now x_i, $\bar{x}'_i$ and q_{i-1} have pebbles. Over the old region $\check{R}$, at most $(\gamma_i-3)+2 = \gamma_i-1$ pebbles are used (this step is legal as $\check{R} = \check{R}_1 \cup \big(anc(x_i) \cup anc(\bar{x}'_i)\big)$ and x_i and $\bar{x}_i$ have pebbles);

(iii) Persistently pebble the turnpike from q_{i-1} to f_i. Now x_i, $\bar{x}'_i$, q_{i-1} and f_i have pebbles. At most $(\gamma_i-5)+2 = \gamma_i-3$ pebbles are used over the turnpike (including the pebble on node q_{i-1}) by Lemma E.6.9, so at most $(\gamma_i - 3) + 2 = \gamma_i - 1$ pebbles are used over $\check{R}$; and

(iv) Pebble node $\bar{x}_i$ then node g_i. Now the six nodes x_i, $\bar{x}'_i$, $\bar{x}_i$, q_{i-1}, f_i, g_i have pebbles.

For Item (2), since $\phi_{i-1} \restriction_{\rho_1}$ is FALSE, by the assumption that Lemma E.7.3 holds for $i-1$, we have $\mathsf{RPeb}_{\check{R}_1}(q_{i-1}) = \gamma_{i-1} + 1 = (\gamma_i - 3) + 1 = \gamma_i - 2$. We run the same strategy as in Item (1) to surround q_i, using at most γ_i pebbles over $\check{R}$ (only Step (ii) uses one more pebble). □

Lemma E.7.28 (Existential Lower Bound). *Assume Lemma E.7.3 holds for $i-1$.*

1. $\mathsf{RPeb}^S_{\check{R}}(q_i) \geq \gamma_i - 1$.

2. *If $\mathsf{RPeb}^S_{\check{R}}(q_i) \leq \gamma_i - 1$, then $\phi_i \restriction_\rho$ is TRUE.*

Proof. Fix a strategy to surround q_i using at most $\gamma_i - 1$ pebbles over $\check{R}$. At the end, there is a pebble on the turnpike from q_{i-1} to f_i, and there is a pebble on g_i. Let t_2 be the earliest time such that since t_2 there is at least one pebble on the turnpike from q_{i-1} to f_i. Let t_3 be the earliest time such that since t_3 there is a pebble on g_i.

At time t_3, node g_i is being pebbled, so both nodes x_i and $\bar{x}_i$ have pebbles, and none of literal x_i or $\bar{x}_i$ is in $*$ position. Let t_0 (resp. t_1) be the last time before time t_3 such that literal $\bar{x}_i$ (resp. literal x_i) has a transition (Definition E.7.6). Note that neither literal $\bar{x}_i$ nor x_i can have a transition after time t_3: to make a transition for literal $\bar{x}_i$ (resp. x_i) takes $r_i = \gamma_i - 2$ pebbles on the ancestors of node $\bar{x}_i'$ (resp. x_i') by Lemma E.7.7, but there is a pebble on the other literal gadget, i.e., on $L_{r_i}(x_i)$ (resp. $L_{r_i}(\bar{x}_i)$), and there is a pebble on g_i, which cannot be done with at most $\gamma_i - 1$ pebbles over $\check{R}$. So time t_0 (resp. t_1) is in fact the last time that literal $\bar{x}_i$ (resp. x_i) has a transition, and literal $\bar{x}_i$ (resp x_i) is not in $*$ position since t_0 (resp. t_1).

Assume $t_0 < t_1$ by symmetry (in the rest of this argument).

Claim E.7.29 (Clearance). *At time t_1, there is no pebble over $\check{R}_0 = \check{R} \setminus \big(anc(x_i') \cup anc(\bar{x}_i)\big)$.*

Proof. At time t_1, there is a transition of literal x_i, accounting for $r_i = \gamma_i - 2$ pebbles on $anc(x_i')$ by Lemma E.7.7. And there is a pebble on the other literal gadget $L_{r_i}(\bar{x}_i)$, because literal $\bar{x}_i$ is not in $*$ position. This accounts for at least $\gamma_i - 1$ pebbles over $\check{R}$. □

The proof of Claim E.7.29 establishes Item (1).

By Claim E.7.29, we know that since time t_1 literal x_i is not in TRUE position, hence in FALSE position. As there is no transition of literals $\bar{x}_i$ or x_i after time t_1, there is a pebble on $anc(\bar{x}_i)$ and a pebble on $anc(x_i')$ since time t_1. Over the region $\check{R}_0$, there are at most $(\gamma_i - 1) - 2 = \gamma_i - 3$ pebbles since time t_1. Note that region $\check{R}_0$ is associated with the $(i-1)$-assignment ρ_0.

By Claim E.7.29, we have $t_0 < t_1 < t_2$, where t_2 is defined (in the first paragraph of this proof) as the earliest time since which there is at least one pebble on the turnpike from q_{i-1} to f_i. Note that at time $t_2 - 1$ there is no pebble on the turnpike from q_{i-1} to f_i, but at the end there is a pebble on f_i. The sub-strategy since time $t_2 - 1$ visits f_i when restricted to the turnpike from q_{i-1} to f_i, so by Lemma E.6.7, there is a time t_4 after t_2 such that there are $(\gamma_i - 5) + 2 = \gamma_i - 3$ pebbles over the turnpike, including one pebble on q_{i-1}. As a result, over the region $\check{R}_0 \cap anc(q_{i-1})$, there is only one pebble at time t_4, which is on node q_{i-1}. The sub-strategy from time t_1 to t_4 persistently pebble node q_{i-1} over the region $\check{R}_0 \cap anc(q_{i-1})$ using $\gamma_i - 3 = \gamma_{i-1}$ pebbles, so $\mathit{RPeb}_{\check{R}_0}\big(G(\phi_{i-1})\big) \le \gamma_{i-1}$. By the assumption that Lemma E.7.3 holds for $i-1$, we know $\phi_{i-1} \restriction_{\rho_0}$ is TRUE. As a result, $\exists x_i \phi_{i-1} \restriction_{\rho} = \phi_i \restriction_{\rho}$ is TRUE, giving Item (2). □

Lemma E.7.30 (Existential Quantifier Gadget). *Assume that Lemma E.7.3 holds for $i-1$. We have $\mathit{RPeb}_{\check{R}}\big(G(\phi_i)\big) = \gamma_i + [\![\phi_i \restriction_{\rho}$ is FALSE$]\!]$.*

Proof. Since persistent price is one plus surrounding price (Proposition E.2.1), it suffices to show that $\mathit{RPeb}^S_{\check{R}}\big(G(\phi_i)\big) = \gamma_i - 1 + [\![\phi_i \restriction_{\rho}$ is FALSE$]\!]$. If $\phi_i \restriction_{\rho}$ is TRUE, then $\mathit{RPeb}^S_{\check{R}}\big(G(\phi_i)\big) = \gamma_i - 1$, as the upper bound (Lemma E.7.27) matches the lower bound (Lemma E.7.28). If $\phi_i \restriction_{\rho}$ is FALSE, then $\mathit{RPeb}_{\check{R}}\big(G(\phi_i)\big) = \gamma_i$, as the upper bound (Lemma E.7.27) matches the lower bound (Lemma E.7.28). □

E.7.7 Universal Quantifier Gadget

Assume that we already have the gadget $G(\phi_{i-1})$ and that the i^{th} inner-most quantifier is existential, i.e., $Q_i = \forall$. This quantifier refers to x_i, we set the parameter $r_i := \gamma_i - 3$ for the corresponding variable gadget. We construct $G(\phi_i)$ as follows.

Construction E.7.31 (Universal Quantifier Gadget). Let q_{i-1} denote the sink of the previous quantifier gadget $G(\phi_{i-1})$. Construct nodes f_i', $\bar{f}_i'$, f_i, $\bar{f}_i$, g_i, $\bar{g}_i$, h_i, $\bar{h}_i$, q_i, and edges (x_i, f_i'), $(\bar{x}_i', f_i')$, (f_i, h_i), (g_i, h_i), $(\bar{x}_i, \bar{f}_i')$, $(x_i', \bar{f}_i')$, $(\bar{f}_i, \bar{h}_i)$, $(\bar{g}_i, \bar{h}_i)$, (h_i, q_i), $(\bar{h}_i, q_i)$. Add a turnpike of toll $\gamma_i - 6$ from f_i' to f_i, a turnpike of toll $\gamma_i - 6$ from $\bar{f}_i'$ to $\bar{f}_i$, a turnpike of toll $\gamma_i - 7$ from q_{i-1} to g_i, and a turnpike of toll $\gamma_i - 7$ from q_{i-1} to $\bar{g}_i$.

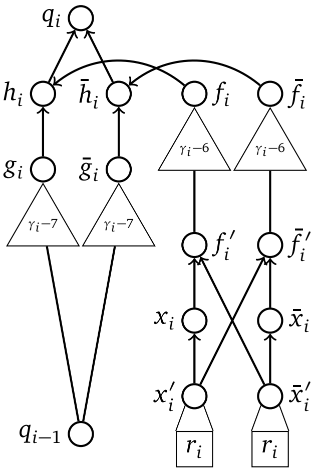


Figure E.18: Universally quantified variable $\forall x_i$.

For the gadget $G := G(\phi_i)$, the nodes in the gadget $V(G)$ contains the nodes in the previous gadget $V\big(G(\phi_{i-1})\big)$, the new nodes f_i', $\bar{f}_i'$, f_i, $\bar{f}_i$, g_i, $\bar{g}_i$, h_i, $\bar{h}_i$, q_i, and the nodes in the four turnpikes.

Lemma E.7.32 (One-Sided Upper Bound). *Assume Lemma E.7.3 holds for $i-1$.*

1. *If $\phi_{i-1} \restriction_{\rho_1}$ is TRUE, then using at most $\gamma_i - 2$ pebbles over $\check{R}$, we can leave pebbles on nodes x_i, $\bar{x}_i'$, f_i, q_{i-1}, g_i.*

2. *If $\phi_{i-1} \restriction_{\rho_0}$ is TRUE, then using at most $\gamma_i - 2$ pebbles over $\check{R}$, we can leave pebbles on nodes $\bar{x}_i$, x_i', $\bar{f}_i$, q_{i-1}, $\bar{g}_i$.*

3. *If $\phi_{i-1} \restriction_{\rho_1}$ is FALSE, then using at most $\gamma_i - 1$ pebbles over $\check{R}$, we can leave pebbles on nodes x_i, $\bar{x}_i'$, f_i, q_{i-1}, g_i.*

4. *If $\phi_{i-1} \upharpoonright_{\rho_0}$ is* FALSE*, then using at most $\gamma_i - 1$ pebbles over $\check{R}$, we can leave pebbles on nodes $\bar{x}_i$, x'_i, $\bar{f}_i$, q_{i-1}, $\bar{g}_i$.*

Proof. For Item (1), since $\phi_{i-1} \upharpoonright_{\rho_1}$ is TRUE, by the assumption that Lemma E.7.3 holds for $i-1$, we have $\mathsf{RPeb}_{\check{R}_1}(q_{i-1}) = \gamma_{i-1} = \gamma_i - 5$. Consider the following strategy to leave pebbles on x_i, $\bar{x}'_i$, f_i, q_{i-1}, g_i using at most $\gamma_i - 2$ pebbles over $\check{R}$:

(i) Put x_i into the canonical TRUE position with $r_i + 1 = \gamma_i - 2$ pebbles by Lemma E.7.10. Now node x_i and node $\bar{x}'_i$ have pebbles;

(ii) Pebble f'_i, persistently pebble the turnpike from f'_i to f_i, then unpebble f'_i. Now x_i, $\bar{x}'_i$ and f_i have pebbles. Over the turnpike from f'_i to f_i, at most $(\gamma_i - 6) + 2 = \gamma_i - 4$ pebbles are used by Lemma E.6.9. Over $\check{R}$, at most $(\gamma_i - 4) + \underbrace{2}_{x_i, \bar{x}'_i} = \gamma_i - 2$ pebbles are used.

(iii) Persistently pebble node q_{i-1} using at most $\gamma_i - 5$ pebbles over the new region $\check{R}_1$. Now x_i, $\bar{x}'_i$, f_i and q_{i-1} have pebbles. Over the old region $\check{R}$, at most $\underbrace{\gamma_i - 5}_{\check{R}_1} + \underbrace{3}_{x_i, \bar{x}'_i, f_i} = \gamma_i - 2$ pebbles are used (this step is legal since $\check{R} = \check{R}_1 \cup \big(anc(x_i) \cup anc(\bar{x}'_i)\big)$ and since x_i and $\bar{x}_i$ have the only pebbles on $anc(x_i) \cup anc(\bar{x}'_i)$);

(iv) Persistently pebble the turnpike from q_{i-1} to g_i. Now x_i, $\bar{x}'_i$, f_i, q_{i-1} and g_i have pebbles. Over the turnpike from q_{i-1} to g_i, at most $(\gamma_i - 7) + 2 = \gamma_i - 5$ pebbles are used by Lemma E.6.9. Over $\check{R}$, at most $(\gamma_i - 5) + \underbrace{3}_{x_i, \bar{x}'_i, f_i} = \gamma_i - 2$ pebbles are used.

Item (2) is symmetric to Item (1).

For Item (3), since $\phi_{i-1} \upharpoonright_{\rho_1}$ is FALSE, by the assumption that Lemma E.7.3 holds for $i-1$, we have $\mathsf{RPeb}_{\check{R}_1}(q_{i-1}) = \gamma_{i-1} + 1 = (\gamma_i - 5) + 1 = \gamma_i - 4$. We run the same strategy as in Item (1) to leave pebbles on x_i, $\bar{x}'_i$, f_i, q_{i-1}, g_i, using at most $\gamma_i - 1$ pebbles over $\check{R}$ (only Step (iii) uses one more pebble).

Item (4) is symmetric to Item (3). □

Lemma E.7.33 (Universal Upper Bound). *Assume Lemma E.7.3 holds for $i-1$.*

1. *If $\phi_i \upharpoonright_\rho$ is* TRUE*, then $\mathsf{RPeb}^S_{\check{R}}(q_i) \le \gamma_i - 1$.*

2. *If $\phi_i \upharpoonright_\rho$ is* FALSE*, then $\mathsf{RPeb}^S_{\check{R}}(q_i) \le \gamma_i$.*

Proof. For Item (1), by assumption $\phi_i \upharpoonright_\rho = \forall x_i \phi_{i-1} \upharpoonright_\rho$ is TRUE, so assigning x_i to TRUE and to FALSE both satisfy ϕ_{i-1}, i.e., $\phi_{i-1} \upharpoonright_{\rho_1}$ is TRUE and $\phi_{i-1} \upharpoonright_{\rho_0}$ is TRUE. Consider the following strategy to surround q_i with at most $\gamma_i - 1$ pebbles over $\check{R}$:

(i) Run Item (1) of Lemma E.7.32 to pebble nodes x_i, $\bar{x}_i'$, f_i, q_{i-1}, g_i, using at most $\gamma_i - 2$ pebbles over $\check{R}$.

(ii) Pebble h_i. Now the six nodes x_i, $\bar{x}_i'$, f_i, q_{i-1}, g_i, h_i have pebbles.

(iii) Run the reverse of Item (1) of Lemma E.7.32 to remove pebbles from nodes x_i, $\bar{x}_i'$, f_i, q_{i-1}, g_i. Now node h_i has a pebble. Over $\check{R}$, at most $\gamma_i - 2 + \underbrace{1}_{h_i} = \gamma_i - 1$ pebbles are used.

(iv) Run Item (2) of Lemma E.7.32 to pebble nodes $\bar{x}_i$, x_i', $\bar{f}_i$, q_{i-1}, $\bar{g}_i$. Now the six nodes h_i, $\bar{x}_i$, x_i', $\bar{f}_i$, q_{i-1}, $\bar{g}_i$ have pebbles. Over $\check{R}$, at most $\gamma_i - 2 + \underbrace{1}_{h_i} = \gamma_i - 1$ pebbles are used.

(v) Pebble $\bar{h}_i$ to surround q_i. Seven nodes have pebbles.

For Item (2), we run the same strategy as in Item (1) to surround q_i, using at most γ_i pebbles over $\check{R}$ (each of Steps (i), (iii), (iv) may use one more pebble by Items (3) and (4) of Lemma E.7.32). □

Lemma E.7.34 (Universal Lower Bound). *Assume Lemma E.7.3 holds for $i-1$.*

1. $\mathsf{RPeb}^S_{\check{R}}(q_i) \geq \gamma_i - 1$.

2. *If* $\mathsf{RPeb}^S_{\check{R}}(q_i) \leq \gamma_i - 1$, *then* $\phi_i \restriction_\rho$ *is* TRUE.

Proof. Fix a strategy to surround q_i using at most $\gamma_i - 1$ pebbles over $\check{R}$. Let $\check{R}_f := \{h_i\} \cup anc(f_i) \setminus anc^*(f_i')$ be the region to augment h_i to the turnpike from f_i' to f_i, and $\check{R}_{\bar{f}} := \{\bar{h}_i\} \cup anc(\bar{f}_i) \setminus anc^*(\bar{f}_i')$ be the region to augment $\bar{h}_i$ to the turnpike from $\bar{f}_i'$ to $\bar{f}_i$. At the end, the region $\check{R}_f$ has a pebble (on h_i) and the region $\check{R}_{\bar{f}}$ has a pebble (on $\bar{h}_i$). Let t_1 (resp. t_0) be the earliest time such that since time t_1 (resp. t_0) the region $\check{R}_f$ (resp. $\check{R}_{\bar{f}}$) has pebble.

Assume $t_0 < t_1$ by symmetry (in the rest of this argument). At time $t_1 - 1$, there is no pebble on $\check{R}_f$, and there are pebbles on nodes x_i and $\bar{x}_i'$ (so that node f_i' can be pebbled at time t_1). Hence literal $\bar{x}_i$ is in FALSE position, and literal x_i is not in $*$ position. Note that there is no transition of literals x_i or $\bar{x}_i$ since time t_1: to make a transition for literal x_i (resp. $\bar{x}_i$) takes $r_i = \gamma_i - 3$ pebbles over $anc(x_i')$ (resp. $anc(\bar{x}_i')$) by Lemma E.7.7, but there is a pebble on $\check{R}_{\bar{f}}$, a pebble on $\check{R}_f$, and a pebble on the other literal gadget $L_{r_i}(\bar{x}_i)$ (resp. $L_{r_i}(x_i)$); thus a transition cannot be done with at most $\gamma_i - 1$ pebbles over $\check{R}$. As such, there is a pebble on $anc(x_i)$ and a pebble on $anc(\bar{x}_i')$ since time t_1.

Because node f_i must be visited before node h_i can be pebbled, by Lemma E.6.9, at some later time $t_2 > t_1$ there are $(\gamma_i - 6) + 2 = \gamma_i - 4$ pebbles on the turnpike from f_i'

to f_i. At time t_2 over $\check{R}$, there are at least

$$\underbrace{\gamma_i - 4}_{\check{R}_f \setminus \{h_i\}} + \underbrace{1}_{\check{R}_{\bar{f}}} + \underbrace{2}_{Anc(\{x_i, \bar{x}_i'\})} = \gamma_i - 1 \tag{E.7.5}$$

pebbles, giving Item (1).

At time t_2, there is no pebble on $\{h_i\} \cup \check{R}_g$ where $\check{R}_g := anc(g_i) \setminus Anc(\{x_i, \bar{x}_i'\})$, as all $\gamma_i - 1$ pebbles over $\check{R}$ are on $\check{R}_f \setminus \{h_i\}$, $\check{R}_{\bar{f}}$ or $Anc(\{x_i, \bar{x}_i'\})$ by (E.7.5). Over $\check{R} \setminus \check{R}_g$, there are at least

$$\underbrace{1}_{\check{R}_f} + \underbrace{1}_{\check{R}_{\bar{f}}} + \underbrace{2}_{Anc(\{x_i, \bar{x}_i'\})} = 4$$

pebbles since time t_2, so at most $(\gamma_i - 1) - 4 = \gamma_i - 5$ pebbles over $\check{R}_g$. Because node g_i must be visited before node h_i can be pebbled, by Lemma E.6.7, at some later time $t_3 > t_2$ there are $(\gamma_i - 7) + 2 = \gamma_i - 5$ pebbles on the turnpike from q_{i-1} to g_i, including one on node q_{i-1}. Recall the region $\check{R}_1 = \check{R} \setminus \big(anc(x_i) \cup anc(\bar{x}_i')\big)$ that is associated with the $(i-1)$-assignment ρ_1. At time t_3, there is only one pebble over $\check{R}_1 \cap anc(q_{i-1})$, which is on node q_{i-1}. The sub-strategy from time t_2 to t_3 persistently pebble node q_{i-1} over the region $\check{R}_1 \cap anc(q_{i-1})$ using $\gamma_i - 5 = \gamma_{i-1}$ pebbles, so $\mathsf{RPeb}_{\check{R}_1}\big(G(\phi_{i-1})\big) \le \gamma_{i-1}$. By the assumption that Lemma E.7.3 holds for $i-1$, we know $\phi_{i-1} \restriction_{\rho_1}$ is TRUE.

We claim that at time t_2, node $\bar{h}_i$ has a pebble (by modifying the argument in the previous paragraph). Let $\check{R}_h := \big(\{h_i\} \cup anc(g_i)\big) \setminus anc(q_{i-1})$ be nodes properly in the turnpike from q_{i-1} to g_i plus h_i, and $\check{R}_{\bar{h}} := \big(\{\bar{h}_i\} \cup anc(\bar{g}_i)\big) \setminus anc(q_{i-1})$ be nodes properly in the turnpike from q_{i-1} to $\bar{g}_i$ plus $\bar{h}_i$.

Claim E.7.35 (Persistence). *At time t_2, node $\bar{h}_i$ has a pebble.*

Proof. For otherwise, at time t_2, there is no pebble on $\check{R}_h$ or $\check{R}_{\bar{h}}$, as all $\gamma_i - 1$ pebbles over $\check{R}$ are on $\check{R}_f \setminus \{h_i\}$, $\check{R}_{\bar{f}} \setminus \{\bar{h}_i\}$ or $anc(\{x_i, \bar{x}_i'\})$ by (E.7.5). Let t_3 (resp. t_4) be the earliest time such that since time t_3 (resp. t_4) the region $\check{R}_h$ (resp. $\check{R}_{\bar{h}}$) has pebble.

Claim E.7.36 (No Double Persistence). *Since t_3, there are at least two pebbles over $\check{R}_f \cup \check{R}_h$.*

Proof. Note that $\check{R}_f$ consists of the turnpike from f_i' to f_i plus node h_i, and $\check{R}_h$ consists of nodes *properly* in the turnpike from q_{i-1} to g_i plus node h_i.

Fix an induced subgraph $F \subseteq G$ (for instance $F =$ the turnpike from f_i' to f_i) having a unique sink. Say a pebbling configuration $\mathbb{P}$ is *v-locked* on F if $\mathsf{RPeb}_F(\mathbb{P}) = \mathsf{RPeb}^V(F)$; this is if the pebbles cannot be cleared without using $\mathsf{RPeb}^V(F)$ pebbles (when restricted to F). In particular, any configuration with a pebble on the sink of F is v-locked on F. Also, if a configuration is v-locked on F, then there is a pebble on F. Given a pebbling configuration on G, we say that f_i (resp. g_i) is *v-locked* if the configuration is v-locked on the turnpike from f_i' to f_i (resp. from q_{i-1} to g_i).

Assume for contradiction that at some time $t_7 \geq t_3$ there is only one pebble over $\check{R}_f \cup \check{R}_h$, which must be on h_i by the inclusion-exclusion principle. Let t_5 be the earliest time before t_7 such that there is a pebble on node h_i from t_5 to t_7. We know $t_0 < t_1 < t_3 < t_5 < t_7$. At time t_5 node h_i is pebbled, so node g_i and node f_i each has a pebble, and both are v-locked. At time t_7, nodes properly in the two turnpikes have no pebbles, and nodes g_i and f_i are not v-locked. Let t_6 be the earliest time after t_5 such that one of the turnpikes is not v-locked, then $t_5 < t_6 < t_7$.

- If the turnpike from f_i' to f_i stops being v-locked at time t_6, then at time t_6-1 there are $(\gamma_i - 6) + 2 = \gamma_i - 4$ pebbles over the turnpike from f_i' to f_i by Lemma E.6.9. Over $\check{R}$, there are

  $$\underbrace{\gamma_i - 4}_{\text{turnpike from } f_i' \text{ to } f_i} + \underbrace{1}_{\text{turnpike from } q_{i-1} \text{ to } g_i} + \underbrace{1}_{h_i} + \underbrace{1}_{\check{R}_{\bar{f}}} + \underbrace{2}_{Anc(\{x_i, \bar{x}_i'\})} = \gamma_i + 1$$

  pebbles, contradicting that at most $\gamma_i - 1$ pebbles are used over $\check{R}$.

- If the turnpike from q_{i-1} to g_i stops being v-locked at time t_6, then at time $t_6 - 1$ there are $(\gamma_i - 7) + 2 = \gamma_i - 5$ pebbles over the turnpike from q_{i-1} to g_i by Lemma E.6.9. Over $\check{R}$, there are

  $$\underbrace{\gamma_i - 5}_{\text{turnpike from } q_{i-1} \text{ to } g_i} + \underbrace{1}_{\text{turnpike from } f_i' \text{ to } f_i} + \underbrace{1}_{h_i} + \underbrace{1}_{\check{R}_{\bar{f}}} + \underbrace{2}_{Anc(\{x_i, \bar{x}_i'\})} = \gamma_i$$

  pebbles, contradicting that at most $\gamma_i - 1$ pebbles are used over $\check{R}$. □

Assume $t_3 < t_4$ by symmetry (in the rest of Claim E.7.35). At time $t_4 - 1$, there is no pebble on $\check{R}_{\bar{h}}$. By Lemma E.6.9, at some later time $t_5 > t_4$ there are $(\gamma_i - 7) + 2 = \gamma_i + 5$ pebbles on the turnpike from q_{i-1} to $\bar{g}_i$. Over $\check{R}$, there are at least

$$\underbrace{\gamma_i - 5}_{\text{turnpike from } q_{i-1} \text{ to } \bar{g}_i} + \underbrace{2}_{\check{R}_f \cup \check{R}_h} + \underbrace{1}_{\check{R}_{\bar{f}}} + \underbrace{2}_{Anc(\{x_i, \bar{x}_i'\})} = \gamma_i$$

pebbles, contradicting that at most $\gamma_i - 1$ pebbles are used over $\check{R}$. □

Claim E.7.35 shows that there is a pebble on $\bar{h}_i$ at time t_2, hence there is no pebble on the turnpike from $\bar{f}_i'$ to $\bar{f}_i$ by (E.7.5). Let t_3 be the earliest time before t_2 such that the only pebble on $\check{R}_{\bar{f}}$ is on $\bar{h}_i$, and let t_6 be the earliest time before t_3 such that $\bar{h}_i$ has a pebble from time t_6 to t_3. At time t_3, there are pebbles on nodes x_i' and $\bar{x}_i$ (so that node $\bar{f}_i'$ can be unpebbled at time $t_3 - 1$), hence literal x_i is in FALSE position, and literal $\bar{x}_i$ is not in $*$ position. Note that there is no transition of literals x_i or $\bar{x}_i$ between time t_6 and $t_3 - 1$: to make a transition for literal x_i (resp. $\bar{x}_i$) takes $r_i = \gamma_i - 3$ pebbles over $anc(x_i')$ (resp. $anc(\bar{x}_i')$) by Lemma E.7.7, but there is a pebble on $\bar{h}_i$, a pebble on $\check{R}_{\bar{f}} \setminus \{\bar{h}_i\}$, and a pebble on the other literal gadget $L_{r_i}(\bar{x}_i)$ (resp. $L_{r_i}(x_i)$); thus a transition cannot be done with at most $\gamma_i - 1$ pebbles over $\check{R}$. As such, there is a pebble on $anc(x_i')$ and a pebble on $anc(\bar{x}_i)$ between time t_6 and t_3.

At time t_6, node $\bar{g}_i$ and node $\bar{f}_i$ each has a pebble (so that node $\bar{h}_i$ can be pebbled at time $t_6 - 1$). At time t_3, the turnpike from $\bar{f}_i'$ to $\bar{f}_i$ has no pebble. By Lemma E.6.9, there is a time t_4 between t_6 and t_3 such that there are $(\gamma_i - 6) + 2 = \gamma_i - 4$ pebbles on the turnpike from $\bar{f}_i'$ to $\bar{f}_i$. We know $t_6 < t_4 < t_3 < t_2$. At time t_4 over $\check{R}$, there are at least

$$\underbrace{\gamma_i - 4}_{\check{R}_{\bar{f}} \setminus \{\bar{h}_i\}} + \underbrace{1}_{\bar{h}_i} + \underbrace{2}_{Anc(\{x_i', \bar{x}_i\})} = \gamma_i - 1 \tag{E.7.6}$$

pebbles.

At time t_4, there is no pebble on $\check{R}_{\bar{g}}$ where $\check{R}_{\bar{g}} := anc(\bar{g}_i) \setminus Anc(\{x_i', \bar{x}_i\})$, as all $\gamma_i - 1$ pebbles over $\check{R}$ are on $\check{R}_{\bar{f}}$ or $anc(\{x_i, \bar{x}_i'\})$ by (E.7.6). Over $\check{R} \setminus \check{R}_{\bar{g}}$, there are at least

$$\underbrace{1}_{\check{R}_{\bar{f}} \setminus \{\bar{h}_i\}} + \underbrace{1}_{\bar{h}_i} + \underbrace{2}_{Anc(\{x_i', \bar{x}_i\})} = 4$$

pebbles between time t_6 and $t_3 - 1$, so at most $(\gamma_i - 1) - 4 = \gamma_i - 5$ pebbles over $\check{R}_{\bar{g}}$. Because node $\bar{g}_i$ is visitied at time t_6 and the turnpike from q_{i-1} to $\bar{g}_i$ has no pebble at time t_4, by Lemma E.6.7, at some time t_5 between t_6 and t_4 there are $(\gamma_i - 7) + 2 = \gamma_i - 5$ pebbles on the turnpike from q_{i-1} to $\bar{g}_i$, including one on node q_{i-1}. Recall the region $\check{R}_0 = \check{R} \setminus \big(anc(x_i') \cup anc(\bar{x}_i)\big)$ that is associated with the $(i-1)$-assignment ρ_0. At time t_5, there is only one pebble over $\check{R}_0 \cap anc(q_{i-1})$, which is on node q_{i-1}. The (reverse of the) sub-strategy from time t_5 to t_4 persistently pebble node q_{i-1} over the region $\check{R}_0 \cap anc(q_{i-1})$ using $\gamma_i - 5 = \gamma_{i-1}$ pebbles, so $\mathsf{RPeb}_{\check{R}_0}\big(G(\phi_{i-1})\big) \le \gamma_{i-1}$. By the assumption that Lemma E.7.3 holds for $i-1$, we know $\phi_{i-1} \restriction_{\rho_0}$ is TRUE.

As a result, $\forall x_i \phi_{i-1} \restriction_\rho = \phi_i \restriction_\rho$ is TRUE, giving Item (2). □

Lemma E.7.37 (Universal Quantifier Gadget). *Assume that Lemma E.7.3 holds for $i-1$. We have $\mathsf{RPeb}_{\check{R}}\big(G(\phi_i)\big) = \gamma_i + [\![\phi_i \restriction_\rho \text{ is FALSE}]\!]$.*

Proof. Since persistent price is one plus surrounding price (Proposition E.2.1), it suffices to show that $\mathsf{RPeb}^S_{\check{R}}\big(G(\phi_i)\big) = \gamma_i - 1 + [\![\phi_i \restriction_\rho \text{ is FALSE}]\!]$. If $\phi_i \restriction_\rho$ is TRUE, then $\mathsf{RPeb}^S_{\check{R}}\big(G(\phi_i)\big) = \gamma_i - 1$, as the upper bound (Lemma E.7.33) matches the lower bound (Lemma E.7.34). If $\phi_i \restriction_\rho$ is FALSE, then $\mathsf{RPeb}_{\check{R}}\big(G(\phi_i)\big) = \gamma_i$, as the upper bound (Lemma E.7.33) matches the lower bound (Lemma E.7.34). □

E.8 Product Construction for Reversible Pebbling

The part of the proof of Theorem E.3.4 that deals with reversible pebbling uses as a black box the construction in Theorem E.3.5 for reversible pebbling. Now we state it again and we give its full proof.

Theorem E.8.1. *Given two graphs G_1 and G_2, there is a polynomial-time constructible graph $\mathcal{R}(G_1, G_2)$ of size $3|G_1| \cdot |G_2|$ with reversible pebbling price $\mathsf{RPeb}\big(\mathcal{R}(G_1, G_2)\big) = \mathsf{RPeb}(G_1) + \mathsf{RPeb}(G_2) + 1$.*

We want to construct a graph $\mathcal{R}$ to inherit structures from two graphs, which are called respectively the *exterior* graph G_1 and the *interior* graph G_2. Intuitively, for every node in G_1, we will construct a *block* with the structure of G_2: in each such block, for every node in G_2 we create a *cell* of three nodes, where different cells are connected according to the exterior graph G_1 and the interior graph G_2.

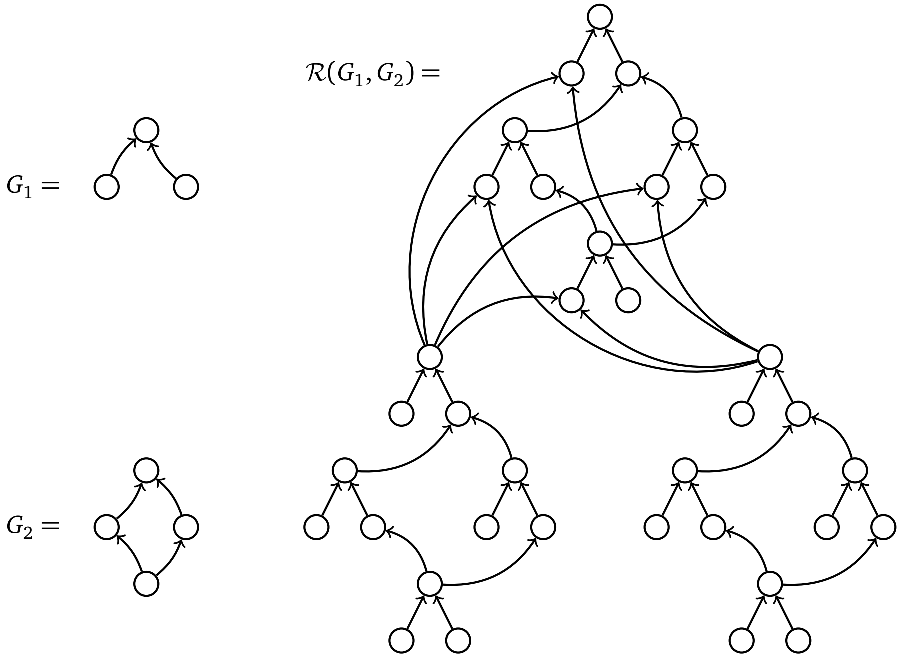


Figure E.19: Example of Construction E.8.2: product of a pyramid of height 1 and a rhombus.

Construction E.8.2 (Product for reversible pebbling). Fix two graphs G_1 and G_2, and denote z_2 as the unique sink of G_2. Construct a graph $\mathcal{R} := \mathcal{R}(G_1, G_2)$ as follows. For every node $(v_1, v_2) \in V(G_1) \times V(G_2)$, create three nodes $(v_1, v_2)_{\mathrm{out}}$, $(v_1, v_2)_{\mathrm{ext}}$, $(v_1, v_2)_{\mathrm{int}}$. Add an edge from the exterior node to the output node, i.e., from node $(v_1, v_2)_{\mathrm{ext}}$ to $(v_1, v_2)_{\mathrm{out}}$; add an edge from the interior node to the output node, i.e., from node $(v_1, v_2)_{\mathrm{int}}$ to $(v_1, v_2)_{\mathrm{out}}$. The exterior node supports the structure of the exterior graph G_1, in the sense that for every predecessor w_1 of v_1 in G_1, we create an edge from the sink the w_1-block of G_2, i.e., from node $(w_1, z_2)_{\mathrm{out}}$ to node $(v_1, v_2)_{\mathrm{ext}}$. The interior node supports the structure of the interior graph G_2, in the sense that for every predecessor w_2 of v_2 in G_2, we create an edge from the output node of w_2, i.e., from node $(v_1, w_2)_{\mathrm{out}}$ to node $(v_1, v_2)_{\mathrm{int}}$.

Formally, $V(\mathcal{R}) := \big\{(v_1, v_2)_{\mathrm{out}}, (v_1, v_2)_{\mathrm{ext}}, (v_1, v_2)_{\mathrm{int}} : v_1 \in V(G_1), v_2 \in V(G_2)\big\}$, and $E(\mathcal{R}) := E_{\mathrm{out}} \sqcup E_{\mathrm{ext}} \sqcup E_{\mathrm{int}}$, where $E_{\mathrm{out}} := \Big\{\big((v_1, v_2)_{\mathrm{ext}}, (v_1, v_2)_{\mathrm{out}}\big), \big((v_1, v_2)_{\mathrm{int}}, (v_1, v_2)_{\mathrm{out}}\big) : v_1 \in V(G_1), v_2 \in V(G_2)\Big\}$, $E_{\mathrm{ext}} := \Big\{\big((w_1, z_2)_{\mathrm{out}}, (v_1, v_2)_{\mathrm{ext}}\big) : v_1 \in V(G_1), v_2 \in V(G_2), w_1 \in \mathit{pred}_{G_1}(v_1)\Big\}$, and $E_{\mathrm{int}} := \Big\{\big((v_1, w_2)_{\mathrm{out}}, (v_1, v_2)_{\mathrm{int}}\big) : v_1 \in V(G_1), v_2 \in V(G_2), w_2 \in \mathit{pred}_{G_2}(v_2)\Big\}$.

Clearly, if G_1 and G_2 each has in-degree at most two and a unique sink, then so does the resulting graph $\mathcal{R}$. Note that the graph $\mathcal{R}$ partitions into $|V(G_1)|$ blocks, namely, for each $v_1 \in V(G_1)$, the v_1-block is the subgraph of $\mathcal{R}$ induced over the node set $\big\{(v_1, v_2)_{\mathrm{out}}, (v_1, v_2)_{\mathrm{ext}}, (v_1, v_2)_{\mathrm{int}} : v_2 \in V(G_2)\big\}$. Each such block further partitions into $|V(G_2)|$ cells, namely, in the v_1-block, for each $v_2 \in V(G_2)$, the (v_1, v_2)-cell is the subgraph of the v_1-block induced over the node set $\big\{(v_1, v_2)_{\mathrm{out}}, (v_1, v_2)_{\mathrm{ext}}, (v_1, v_2)_{\mathrm{int}}\big\}$.

Finally, given a configuration $\mathbb{P}'$ on $\mathcal{R}$, say the v_1-block (resp. the (v_1, v_2)-cell) is *surrounded* if any exterior node of the v_1-block (resp. the interior node of the (v_1, v_2)-cell) is surrounded in $\mathbb{P}'$. Note that the v_1-block is surrounded iff *every* exterior node of the v_1-block is surrounded.

Lemma E.8.3 (Upper Bound). $\mathit{RPeb}\big(\mathcal{R}(G_1, G_2)\big) \le \mathit{RPeb}(G_1) + \mathit{RPeb}(G_2) + 1$.

Proof. Fix a persistent pebbling $\mathcal{P}_1$ of G_1 using $\mathit{RPeb}(G_1)$ pebbles, and a persistent pebbling $\mathcal{P}_2$ of G_2 using $\mathit{RPeb}(G_2)$ pebbles. We will construct a persistent pebbling $\mathcal{P}'$ of $\mathcal{R}(G_1, G_2)$ using $\mathit{RPeb}(G_1) + \mathit{RPeb}(G_2) + 1$ pebbles.

We claim that the persistent price of each block of $\mathcal{R}$ is at most $\mathit{RPeb}(G_2) + 2$. For any $v_1 \in V(G_1)$, to persistently pebble the v_1-block (assuming the v_1-block is surrounded), simulate $\mathcal{P}_2$ as follows: whenever $\mathcal{P}_2$ pebbles a node $v_2 \in V(G_2)$, the simulating pebbling has a phase to persistently pebble the (v_1, v_2)-cell, and whenever $\mathcal{P}_2$ unpebbles a node $v_2 \in V(G_2)$, then the simulating pebbling has a phase to persistently unpebble the (v_1, v_2)-cell. If the current configuration in $\mathcal{P}_2$ is $\mathbb{P}$, and the configuration in the simulating pebbling at the end of a phase is $\mathbb{P}'$, then the simulating pebbling maintains the phase-invariant that $\mathbb{P}' = \big\{(v_1, v_2)_{\mathrm{out}} : v_2 \in \mathbb{P}\big\}$. Note that the simulating pebbling is legal: since the pebbling $\mathcal{P}_2$ is legal, when v_2 is pebbled or unpebbled it is surrounded in the current configuration $\mathbb{P}$, so the (v_1, v_2)-cell is surrounded in the simulating configuration $\mathbb{P}'$ and the interior node can be pebbled or unpebbled; and we assume that the v_1-block is surrounded, so the exterior node can be pebbled or unpebbled. The simulating pebbling uses at most two more pebbles (on the exterior node and the interior node of each cell), for at most $\mathit{RPeb}(G_2) + 2$ pebbles over the v_1-block.

Then the resulting graph $\mathcal{R}(G_1, G_2)$ can be persistently pebbled by simulating $\mathcal{P}_1$ as follows. Whenever $\mathcal{P}_1$ pebbles a node $v_1 \in V(G_1)$, the simulating pebbling $\mathcal{P}'$ has a stage to persistently pebble the v_1-block; whenever $\mathcal{P}_1$ unpebbles a node $v_1 \in V(G_1)$, the simulating pebbling $\mathcal{P}'$ has a stage to persistently unpebble the v_1-block. If the current configuration in $\mathcal{P}_1$ is $\mathbb{P}$, and the configuration in the simulating pebbling $\mathcal{P}'$ at the end of a stage is $\mathbb{P}'$, then the simulating pebbling maintains the stage-invariant that $\mathbb{P}' = \big\{(v_1, z_2)_{\mathrm{out}} : v_1 \in \mathbb{P}\big\}$. Note that the simulating pebbling $\mathcal{P}'$ is legal: since the pebbling $\mathcal{P}_1$ is legal, when v_1 is pebble or unpebbled it is surrounded in the current configuration $\mathbb{P}$, so the v_1-block is surrounded in the simulating configuration $\mathbb{P}'$, so the v_1-block can be persistently pebbled or unpebbled in a stage of $\mathcal{P}'$. When the v_1-block is pebbled or unpebbled in a stage of $\mathcal{P}'$, there are at most $\mathsf{RPeb}(G_1) - 1$ pebbles on other blocks of $\mathcal{R}$, and there are at most $\mathsf{RPeb}(G_2) + 2$ pebbles in the v_1-block, for a total of $\mathsf{RPeb}(G_1) + \mathsf{RPeb}(G_2) + 1$ pebbles. □

To prove the lower bound we extract simultaneous pebblings of G_1 and G_2 from any pebbling of $\mathcal{R}$ and use the known pebbling prices of G_1 and G_2 to argue that some configuration needs many pebbles. We do so by projecting a pebbling of $\mathcal{R}$ into G_1 and G_2 as the skeleton of a pebbling and then filling the gaps between configurations with a legal sequence of pebbling moves.

Lemma E.8.4 (Lower Bound). $\mathsf{RPeb}\big(\mathcal{R}(G_1, G_2)\big) \geq \mathsf{RPeb}(G_1) + \mathsf{RPeb}(G_2) + 1.$

Proof. Fix any persistent pebbling $\mathcal{P}' = (\mathbb{P}'_0, \mathbb{P}'_1, \ldots, \mathbb{P}'_\tau)$ of $\mathcal{R}(G_1, G_2)$. For every $v_1 \in V(G_1)$, we are going to simulate a pebbling $\mathcal{P}^{v_1}$ on G_2 based (essentially) on the configurations of $\mathcal{P}'$ over the v_1-block. From the family of pebblings $\{\mathcal{P}^{v_1}\}_{v_1 \in V(G_1)}$, we then simulate a persistent pebbling $\mathcal{P}$ on G_1.

In more detail, for each $v_1 \in V(G_1)$ we define a mapping $\mathit{Int}^{v_1} \colon \mathcal{R} \to G_2$ and we view the sequence of configurations $(\mathit{Int}^{v_1}(\mathbb{P}'_t))_{t \in [0,\tau]}$ as the skeleton of a pebbling of G_2. We fill the gaps between configurations according to the algorithm described below to obtain a legal pebbling $\mathcal{P}^{v_1}$. Similarly we define a mapping $\mathit{Ext} \colon \mathcal{R} \to G_1$ to construct the pebbling $\mathcal{P}$ on G_1.

To describe the mappings we need some definitions. Given a configuration $\mathbb{P}'$ in $\mathcal{P}'$ of $\mathcal{R}(G_1, G_2)$, its projection to the output (resp. exterior, interior) nodes of the v_1-block is $\mathit{proj}^{v_1}_{\mathrm{out}}(\mathbb{P}') := \big\{v_2 \in V(G_2) : (v_1, v_2)_{\mathrm{out}} \in \mathbb{P}'\big\}$ (resp. $\mathit{proj}^{v_1}_{\mathrm{ext}}(\mathbb{P}') := \big\{v_2 \in V(G_2) : (v_1, v_2)_{\mathrm{ext}} \in \mathbb{P}'\big\}$, $\mathit{proj}^{v_1}_{\mathrm{int}}(\mathbb{P}') := \big\{v_2 \in V(G_2) : (v_1, v_2)_{\mathrm{int}} \in \mathbb{P}'\big\}$).

The closure $\mathit{clos}(\mathbb{P}') \subseteq V\big(\mathcal{R}(G_1, G_2)\big)$ is the smallest set of nodes containing $\mathbb{P}'$ that is closed under pebble placements on interior or output nodes; equivalently, $\mathit{clos}(\mathbb{P}')$ can be generated by the following algorithm: Start with $\mathbb{P}'$, while there is a node $v \in V\big(\mathcal{R}(G_1, G_2)\big)$ which is an interior node $v = (v_1, v_2)_{\mathrm{int}}$ or an output node $v = (v_1, v_2)_{\mathrm{out}}$ such that v is surrounded by, but not in, the subset of nodes having pebbles, pebble v. Note that the closure of a v_1-block does not depend on other blocks as they are connected only through exterior nodes.

For brevity, given a graph G and a subset $U \subseteq V(G)$ of vertices, denote $\mathit{unsur}_G(U) := \{v \in V(G) : \mathit{pred}_G(v) \not\subseteq U\}$ as the subset of nodes in G not surrounded by U.

The block mapping is $I^{w_1}\big(\mathbb{P}'\big) := \mathit{proj}^{w_1}_{\mathrm{out}}(\mathbb{P}') \cup \Big(\mathit{proj}^{w_1}_{\mathrm{int}}(\mathbb{P}') \cap \mathit{unsur}_{G_2}\big(\mathit{proj}^{w_1}_{\mathrm{out}}(\mathbb{P}')\big)\Big)$

We define the interior mapping $Int^{w_1}(\mathbb{P}')$ to be $I^{w_1}\big(\mathbb{P}'\big)$ if the w_1-block is surrounded, and $I^{w_1}\big(clos(\mathbb{P}')\big)$ if the w_1-block is not surrounded.

Given a configuration $\mathbb{P}'$, its persistent projection (p-projection) is $proj^P(\mathbb{P}') := \big\{v_1 \in V(G_1) : Int^{v_1}(\mathbb{P}') \text{ is p-locked}\big\}$, and its visiting projection (v-projection) is $proj^V(t) := \big\{v_1 \in V(G_1) : Int^{v_1}(\mathbb{P}') \text{ is v-locked}\big\}$.

Finally the exterior mapping is $Ext(\mathbb{P}') := proj^P(\mathbb{P}') \cup \Big(proj^V(\mathbb{P}') \cap unsur_{G_1}\big(proj^P(\mathbb{P}')\big)\Big)$.

We abuse the notation for mappings from configurations in $\mathcal{P}'$ and write $f(t)$ to mean $f(\mathbb{P}'_t)$. In addition we define $\mathbb{P}^{v_1}(t) = Int^{v_1}(\mathbb{P}'_t)$ and $\mathbb{P}(t) = Ext(\mathbb{P}'_t)$. Note that there can be multiple pebble moves between, say, $\mathbb{P}(t-1)$ and $\mathbb{P}(t)$.

We construct the pebblings $\{\mathcal{P}^{v_1}\}_{v_1 \in V(G_1)}$ and $\mathcal{P}$ according to Algorithm E.8.8, which are legal by Claim E.8.9. Note that $\mathcal{P}$ is a persistent pebbling of G_1: $\mathbb{P}^{w_1}(\tau) = I^{w_1}\big(clos(\tau)\big)$, which is $\emptyset$ if w_1 is not the sink of G_1, and $\{z_2\}$ if w_1 is the sink of G_1, so $\mathbb{P}(\tau) = Ext(\tau) = \{z_1\}$, the sink of G_1. Then the lower bound follows from Claim E.8.13. □

The algorithm to construct the remaining configurations in the pebblings $\{\mathcal{P}^{v_1}\}_{v_1 \in V(G_1)}$ and $\mathcal{P}$, given $\mathcal{P}'$, is essentially to insert and remove missing pebbles in topological order whenever two configuration are different, and make such a pebbling go through a specific configuration in the exterior case. We give an explicit description below and in Claim E.8.9 we prove that it is equivalent to this implicit description.

Definition E.8.5 (Reasonable pebbling). Given a configuration $\mathbb{P}$, a set of vertices to pebble T_+ and a set of vertices to unpebble T_-, the *reasonable pebbling* is the following pebbling:

- start with $\mathbb{P}$;
- for each $v \in T_+$ in a topological order, pebble v;
- for each $v \in T_-$ in a reverse topological order, unpebble v.

Furthermore, the reasonable pebbling between two configurations $\mathbb{P}_1$, $\mathbb{P}_2$ is the reasonable pebbling with $\mathbb{P} = \mathbb{P}_1$, $T_+ = \mathbb{P}_2 \setminus \mathbb{P}_1$, and $T_- = \mathbb{P}_1 \setminus \mathbb{P}_2$.

Claim E.8.6 (Legality). *Assume $\mathbb{P}, T_+, T_- \subseteq V(G)$ are given. If for every $v \in T_+ \cup T_-$, we have $pred_G(v) \subseteq \mathbb{P} \cup T_+$, then the reasonable pebbling is legal.*

Proof. For any $v \in T_+$, right before v is pebbled, we know $pred_G(v)$ have pebbles since $pred_G(v) \subseteq \mathbb{P} \cup T_+$ and the pebble placement on T_+ proceeds in a topological order. So all pebble placements on T_+ are legal.

For any $v \in T_-$, right before v is unpebbled, we know $pred_G(v)$ have pebbles since $pred_G(v) \subseteq \mathbb{P} \cup T_+$ and the pebble placement on T_- proceeds in a reverse topological order. So all pebble removals on T_- are legal. □

Corollary E.8.7 (Legality). *Assume $\mathbb{P}_1, \mathbb{P}_2 \subseteq V(G)$ are given. Assume the sets $T_+ := \mathbb{P}_2 \setminus \mathbb{P}_1$ and $T_- := \mathbb{P}_1 \setminus \mathbb{P}_2$ satisfy that for every $v \in T_+ \cup T_- = \mathbb{P}_1 \triangle \mathbb{P}_2$, we have $pred_G(v) \subseteq \mathbb{P}_1 \cup T_+ = \mathbb{P}_1 \cup \mathbb{P}_2$, then the reasonable pebbling over $\mathbb{P}_1$, T_+, T_- is legal.*

Proof. Apply Claim E.8.6 on $\mathbb{P}_1, T_+, T_-$. □

Algorithm E.8.8. As Claim E.8.9 shows, we only need to consider the case when an output node $(v_1, v_2)_{\text{out}}$ is pebbled or unpebbled to get to $\mathbb{P}'_t$ in $\mathcal{P}'$. In this case we run the following steps in sequence:

(a) if the v_1-block is surrounded in $\mathbb{P}'_t$, we insert the reasonable pebbling between $I^{v_1}\big(t-1\big)$ and $I^{v_1}\big(t\big)$ into $\mathcal{P}^{v_1}$.

(b) if $v_2 = z_2$,

  (i) if $(v_1, v_2)_{\text{out}}$ is pebbled to get to $\mathbb{P}'_t$ in $\mathcal{P}'$, for each successor w_1 of v_1 such that the w_1-block is surrounded in $\mathbb{P}'_t$, we insert the reasonable pebbling between $I^{w_1}\big(clos(t-1)\big)$ and $I^{w_1}\big(t\big)$ into $\mathcal{P}^{w_1}$.

  (ii) if $(v_1, v_2)_{\text{out}}$ is unpebbled to get to $\mathbb{P}'_t$ in $\mathcal{P}'$, for each successor w_1 of v_1 such that the w_1-block is surrounded in $\mathbb{P}'_{t-1}$, we insert the reasonable pebbling between $I^{w_1}\big(t-1\big)$ and $I^{w_1}\big(clos(t)\big)$ into $\mathcal{P}^{w_1}$.

(c) if $Ext(t-1) \neq Ext(t)$, let $D := proj^V(t-1) \cup proj^V(t)$. We insert the reasonable pebbling with $T_+ := D \setminus Ext(t-1)$, and $T_- := D \setminus Ext(t)$ into $\mathcal{P}$.

Claim E.8.9 (Correctness). *The pebblings $\{\mathcal{P}^{v_1}\}_{v_1 \in V(G_1)}$ and $\mathcal{P}$ are legal.*

Proof. At the beginning $t = 0$, we know $\mathbb{P}'_0 = \emptyset$, so for any $w_1 \in V(G_1)$ we have $I^{w_1}\big(0\big) = \emptyset$ and $I^{w_1}\big(clos(0)\big) = \emptyset$, matching $\mathbb{P}^{w_1}(0) = \emptyset$. Also $proj^P(0) = \emptyset = proj^V(0)$, and $\mathbb{P}(0) = \emptyset = Ext(0)$.

For any two consecutive configurations we show that either they are the same or the algorithm inserted a reasonable pebbling between them, which is legal by Claim E.8.11.

When $t > 0$, if an exterior node or an interior node is pebbled or unpebbled to get to $\mathbb{P}'_t$ in $\mathcal{P}'$, fix any $w_1 \in V(G_1)$. Note that the w_1-block is surrounded in $\mathbb{P}'_{t-1}$ iff it is surrounded in $\mathbb{P}'_t$.

If the w_1-block is surrounded, then $proj^{w_1}_{\text{out}}(\cdot)$ is the same at $\mathbb{P}'_{t-1}$ and $\mathbb{P}'_t$, and $proj^{w_1}_{\text{int}}(\cdot) \cap unsur_{G_2}\big(proj^{w_1}_{\text{out}}(\cdot)\big)$ is also the same at $\mathbb{P}'_{t-1}$ and $\mathbb{P}'_t$ because a node being pebbled or unpebbled must be surrounded, and the predecessors of an interior node are output nodes. Hence $\mathbb{P}^{w_1}(t-1) = I^{w_1}\big(t-1\big) = I^{w_1}\big(t\big) = \mathbb{P}^{w_1}(t)$.

Otherwise the w_1-block is not surrounded, then the pebble move is not on any exterior node in the w_1-block, hence $clos(\cdot)$ is the same at $\mathbb{P}'_{t-1}$ and $\mathbb{P}'_t$ over the w_1-block, so $I^{w_1}\big(clos(\cdot)\big)$ is also the same at $\mathbb{P}'_{t-1}$ and $\mathbb{P}'_t$.

Since $Ext(\cdot)$ only depends on $\{\mathbb{P}^{v_1}(t)\}_{v_1 \in V(G_1)}$, it holds that $\mathbb{P}_{t-1} = \mathbb{P}_t$ as well.

Focus on the case when an output node $(v_1, v_2)_{\text{out}}$ is pebbled or unpebbled to get to $\mathbb{P}'_t$ in $\mathcal{P}'$, fix any $w_1 \in V(G_1)$.

- If $w_1 \neq v_1$ and either w_1 is not a successor of v_1 or $v_2 \neq z_2$, note that the w_1-block is surrounded in $\mathbb{P}'_{t-1}$ iff it is surrounded in $\mathbb{P}'_t$. Since $\mathbb{P}'_{t-1} = \mathbb{P}'_t$ over the w_1-block, we have $I^{w_1}\big(t-1\big) = I^{w_1}\big(t\big)$ and $I^{w_1}\big(clos(t-1)\big) = I^{w_1}\big(clos(t)\big)$.

- If $w_1 = v_1$, then the w_1-block is surrounded in $\mathbb{P}'_{t-1}$ iff it is surrounded in $\mathbb{P}'_t$. If the w_1-block is not surrounded, since the pebble move is not on any exterior node in the w_1-block, we have $clos(\cdot)$ is the same at $\mathbb{P}'_{t-1}$ and $\mathbb{P}'_t$ over the w_1-block, so $I^{w_1}\big(clos(\cdot)\big)$ is also the same at $\mathbb{P}'_{t-1}$ and $\mathbb{P}'_t$.

  Otherwise the w_1-block is surrounded and Step (a) inserts a reasonable pebbling into $\mathcal{P}^{w_1}$.

- If w_1 is a successor of v_1 and $v_2 = z_2$ the w_1-block cannot be surrounded in both $\mathbb{P}'_{t-1}$ and $\mathbb{P}'_t$.

  If the w_1-block is unsurrounded in both $\mathbb{P}'_{t-1}$ and $\mathbb{P}'_t$, since $\mathbb{P}'_{t-1} = \mathbb{P}'_t$ over the w_1-block, we have $I^{w_1}\big(clos(t-1)\big) = I^{w_1}\big(clos(t)\big)$.

  Otherwise w_1 is surrounded in exactly one of $\mathbb{P}'_{t-1}$ and $\mathbb{P}'_t$ and Step (b) inserts a reasonable pebbling into $\mathcal{P}^{w_1}$.

Finally, Step (c) inserts a reasonable pebbling into $\mathcal{P}$ as needed.

$\square$

Claim E.8.10 (Exterior Symmetry). *In Step (c), we have* $Ext(t-1) \cup T_+ = D = Ext(t) \cup T_-$.

Proof. Since $proj^P(t) \subseteq proj^V(t)$, by definition we have $Ext(t) = proj^P(t) \cup \Big(proj^V(t) \cap unsur_{G_1}\big(proj^P(t)\big)\Big) \subseteq proj^V(t) \subseteq D$ and likewise $Ext(t-1) \subseteq proj^V(t-1) \subseteq D$. $\square$

Claim E.8.11 (Legality). *All reasonable pebblings are legal.*

Proof. At Step (a) an output node $(v_1, v_2)_{\text{out}}$ is pebbled or unpebbled to get to $\mathbb{P}'_t$ in $\mathcal{P}'$ and the v_1-block is surrounded, hence $\mathbb{P}^{v_1}(t-1) = I^{v_1}\big(t-1\big)$. By Corollary E.8.7 it suffices to show that for $w_2 \in I^{v_1}\big(t\big) \triangle I^{v_1}\big(t-1\big)$, we have $pred_{G_2}(w_2) \subseteq I^{v_1}\big(t\big) \cup I^{v_1}\big(t-1\big)$. Recall $I^{v_1}\big(\cdot\big) = proj^{v_1}_{\text{out}}(\cdot) \cup \Big(proj^{v_1}_{\text{int}}(\cdot) \cap unsur_{G_2}\big(proj^{v_1}_{\text{out}}(\cdot)\big)\Big)$.

Assume $w_2 \in I^{v_1}\big(t\big) \setminus I^{v_1}\big(t-1\big)$, reversing the roles of $t-1$ and t otherwise. If $w_2 \in proj^{v_1}_{\text{out}}(t) \setminus I^{v_1}\big(t-1\big)$, then $w_2 \in proj^{v_1}_{\text{out}}(t) \setminus proj^{v_1}_{\text{out}}(t-1)$, i.e., $w_2 = v_2$ and $(v_1, w_2)_{\text{out}}$ is being pebbled to get to $\mathbb{P}'_t$ in $\mathcal{P}'$. Since $\mathcal{P}'$ is legal, $(v_1, w_2)_{\text{int}}$ has a pebble in $\mathbb{P}'_{t-1}$, i.e., $w_2 \in proj^{v_1}_{\text{int}}(t-1)$, so w_2 is surrounded by $proj^{v_1}_{\text{out}}(t-1) \subseteq I^{v_1}\big(t-1\big)$ as needed.

Otherwise $w_2 \in \Big(proj^{v_1}_{\text{int}}(t) \cap unsur_{G_2}\big(proj^{v_1}_{\text{out}}(t)\big)\Big) \setminus I^{v_1}\big(t-1\big)$, since $proj^{v_1}_{\text{int}}(t) = proj^{v_1}_{\text{int}}(t-1)$, we know $w_2 \in unsur_{G_2}\big(proj^{v_1}_{\text{out}}(t)\big)$ but $w_2 \notin unsur_{G_2}\big(proj^{v_1}_{\text{out}}(t-1)\big)$, so w_2 is a successor of v_2 and $(v_1, v_2)_{\text{out}}$ is being unpebbled to get to t in $\mathcal{P}'$ and $pred_{G_2}(w_2) \subseteq proj^{v_1}_{\text{out}}(t-1) \subseteq I^{v_1}\big(t-1\big)$ as needed.

At Step (b)(i) an output node $(v_1, v_2)_{\text{out}}$ is pebbled to get to $\mathbb{P}'_t$ in $\mathcal{P}'$, and w_1 is a successor of v_1 such that the w_1-block is surrounded in $\mathbb{P}'_t$. Note that the w_1-block is not surrounded in $\mathbb{P}'_{t-1}$, so $\mathbb{P}^{w_1}(t-1) = I^{w_1}\big(clos(t-1)\big)$. Since the pebble move to get to $\mathbb{P}'_t$ in $\mathcal{P}'$ is not in the w_1-block, we have $I^{w_1}\big(clos(t-1)\big) = I^{w_1}\big(clos(t)\big)$. By Corollary E.8.7 it suffices to show that for $w_2 \in I^{w_1}\big(t\big) \triangle I^{w_1}\big(clos(t)\big)$, we have $pred_{G_2}(w_2) \subseteq I^{w_1}\big(t\big) \cup I^{w_1}\big(clos(t)\big)$.

- Assume $w_2 \in I^{w_1}\big(t\big) \setminus I^{w_1}\big(clos(t)\big)$. Since $proj^{w_1}_{\text{out}}(t) \subseteq proj^{w_1}_{\text{out}}(clos(t)) \subseteq I^{w_1}\big(clos(t)\big)$, it follows that $w_2 \in \Big(proj^{w_1}_{\text{int}}(t) \cap unsur_{G_2}\big(proj^{w_1}_{\text{out}}(t)\big)\Big) \setminus I^{w_1}\big(clos(t)\big)$, and since $proj^{w_1}_{\text{int}}(t) \subseteq proj^{w_1}_{\text{int}}(clos(t))$, we know that w_2 is surrounded by $proj^{w_1}_{\text{out}}(clos(t)) \subseteq I^{w_1}\big(clos(t)\big)$ as needed.

- Assume $w_2 \in I^{w_1}\big(clos(t)\big) \setminus I^{w_1}\big(t\big)$. We claim that $w_2 \in proj^{w_1}_{\text{int}}(clos(t))$: if $w_2 \in \Big(proj^{w_1}_{\text{int}}(clos(t)) \cap unsur_{G_2}\big(proj^{w_1}_{\text{out}}(clos(t))\big)\Big) \setminus I^{w_1}\big(t\big)$, then we are done; otherwise $w_2 \in proj^{w_1}_{\text{out}}(clos(t)) \setminus I^{w_1}\big(t\big)$, then $w_2 \notin proj^{w_1}_{\text{out}}(t)$ and thus by definition of $clos(\cdot)$ we have $w_2 \in proj^{w_1}_{\text{int}}(clos(t))$ as claimed. Now if $w_2 \in proj^{w_1}_{\text{int}}(t)$, then w_2 is surrounded by $proj^{w_1}_{\text{out}}(t) \subseteq I^{w_1}\big(t\big)$ as needed; otherwise $w_2 \notin proj^{w_1}_{\text{int}}(t)$, then from $w_2 \in proj^{w_1}_{\text{int}}(clos(t))$ and by definition of $clos(\cdot)$ we have w_2 is surrounded by $proj^{w_1}_{\text{out}}(clos(t)) \subseteq I^{w_1}\big(clos(t)\big)$ as needed.

To see that Step (b)(ii) is legal, note that it is the reverse of Step (b)(i).

At Step (c) an output node $(v_1, v_2)_{\text{out}}$ is pebbled or unpebbled to get to $\mathbb{P}'_t$ in $\mathcal{P}'$. By Claim E.8.6 it suffices to show that for $w_1 \in \big(D \setminus Ext(t-1)\big) \cup \big(D \setminus Ext(t)\big)$, we have $pred_{G_1}(w_1) \subseteq Ext(t-1) \cup T_+ = D$, where the last equality is due to Claim E.8.10. Recall that $Ext(\cdot) = proj^P(\cdot) \cup \Big(proj^V(\cdot) \cap unsur_{G_1}\big(proj^P(\cdot)\big)\Big)$ and $D = proj^V(t-1) \cup proj^V(t)$.

Fix any $w_1 \in \big(D \setminus Ext(t-1)\big) \cup \big(D \setminus Ext(t)\big)$. Swap the roles of t and $t-1$ if necessary, we can assume $w_1 \in D \setminus Ext(t)$. If $w_1 \in proj^V(t) \setminus Ext(t)$, then w_1 is surrounded by $proj^P(t) \subseteq Ext(t) \subseteq D$ as needed.

Otherwise $w_1 \in proj^V(t-1) \setminus proj^V(t)$, in particular $\mathbb{P}^{w_1}(t-1) \neq \mathbb{P}^{w_1}(t)$, so $w_1 \in \{v_1\} \cup succ_{G_1}(v_1)$ by design of the algorithm. Note that it suffices to show that the w_1-block is surrounded in $\mathbb{P}'_{t-1}$ or in $\mathbb{P}'_t$: if the w_1-block is surrounded in $\mathbb{P}'$, for any $u_1 \in pred_{G_1}(w_1)$ the sink z_2 of the inner graph G_2 satisfies $z_2 \in proj^{u_1}_{\text{out}}(\mathbb{P}') \subseteq I^{u_1}\big(\mathbb{P}'\big) \subseteq Int^{u_1}(\mathbb{P}')$, so $u_1 \in proj^V(\mathbb{P}') \subseteq D$ as needed. If $w_1 = v_1$, then in Step (a) the w_1-block is surrounded in $\mathbb{P}'_t$ as needed. Otherwise $w_1 \in succ_{G_1}(v_1)$, then in Step (b) the w_1-block is surrounded in $\mathbb{P}'_t$ (in Step (b)(i)) or in $\mathbb{P}'_{t-1}$ (in Step (b)(ii)) as needed. □

Claim E.8.12 (No Spurious Projections). *For any $w_1 \in V(G_1)$ and $w_2 \in V(G_2)$, if w_2 has a pebble in any configuration of $\mathcal{P}^{w_1}$ created between time $t-1$ and time t, i.e., between $\mathbb{P}^{w_1}(t-1)$ and $\mathbb{P}^{w_1}(t)$, then the (w_1, w_2)-cell has a pebble both in $\mathbb{P}'_{t-1}$ and in $\mathbb{P}'_t$ in $\mathcal{P}'$.*

Proof. Let us reword the claim. Define $\mathit{proj}^{w_1}_{\text{any}}(\mathbb{P}') := \mathit{proj}^{w_1}_{\text{out}}(\mathbb{P}') \cup \mathit{proj}^{w_1}_{\text{ext}}(\mathbb{P}') \cup \mathit{proj}^{w_1}_{\text{int}}(\mathbb{P}')$, note that $\mathit{proj}^{w_1}_{\text{any}}(t-1) = \mathit{proj}^{w_1}_{\text{any}}(t)$, and we want to show $\mathbb{P}^{w_1}(t-1) \cup T^{w_1}_{+} \subseteq \mathit{proj}^{w_1}_{\text{any}}(t)$.

Note that for any $\mathbb{P}'$ and $w_1 \in V(G_1)$ we have $I^{w_1}(\mathbb{P}') \subseteq \mathit{proj}^{w_1}_{\text{any}}(\mathbb{P}')$. Moreover, we have $I^{w_1}(\mathit{clos}(\mathbb{P}')) \subseteq \mathit{proj}^{w_1}_{\text{any}}(\mathbb{P}')$: for any $w_2 \in I^{w_1}(\mathit{clos}(\mathbb{P}'))$, if $w_2 \in \mathit{proj}^{w_1}_{\text{out}}(\mathit{clos}(\mathbb{P}'))$, then either $w_2 \in \mathit{proj}^{w_1}_{\text{out}}(\mathbb{P}') \subseteq \mathit{proj}^{w_1}_{\text{any}}(\mathbb{P}')$ as needed, or $w_2 \notin \mathit{proj}^{w_1}_{\text{out}}(\mathbb{P}')$, then by definition of $\mathit{clos}(\cdot)$ we have $w_2 \in \mathit{proj}^{w_1}_{\text{ext}}(\mathbb{P}') \subseteq \mathit{proj}^{w_1}_{\text{any}}(\mathbb{P}')$ as needed. Otherwise $w_2 \in \mathit{proj}^{w_1}_{\text{int}}(\mathit{clos}(\mathbb{P}')) \cap \mathit{unsur}_{G_2}(\mathit{proj}^{w_1}_{\text{out}}(\mathit{clos}(\mathbb{P}')))$, then it follows that $w_2 \in \mathit{proj}^{w_1}_{\text{int}}(\mathbb{P}') \subseteq \mathit{proj}^{w_1}_{\text{any}}(\mathbb{P}')$ as needed; for otherwise $w_2 \notin \mathit{proj}^{w_1}_{\text{int}}(\mathbb{P}')$, thus by definition of $\mathit{clos}(\cdot)$ we have $\mathit{pred}_{G_2}(w_2) \subseteq \mathit{proj}^{w_1}_{\text{out}}(\mathit{clos}(\mathbb{P}'))$, which contradicts $w_2 \in \mathit{unsur}_{w_1}(\mathit{proj}^{w_1}_{\text{out}}(\mathit{clos}(\mathbb{P}')))$.

To prove the claim it is enough to recall that the set $\mathbb{P}^{w_1}(t-1) \cup T^{w_1}_{+}$ equals $I^{w_1}(\mathbb{P}_{t-1}) \cup I^{w_1}(\mathbb{P}_t)$ in Step (a) because the w_1-block is surrounded, it equals $I^{w_1}(\mathit{clos}(\mathbb{P}_{t-1})) \cup I^{w_1}(\mathbb{P}_t)$ in Step (b)(i)), and it equals $I^{w_1}(\mathbb{P}_{t-1}) \cup I^{w_1}(\mathit{clos}(\mathbb{P}_t))$ in Step (b)(ii). All three sets are contained in $\mathit{proj}^{w_1}_{\text{any}}(t)$ by the previous paragraph. □

Claim E.8.13 (Legal implies Lower Bound). *Assume that the pebblings $\{\mathcal{P}^{v_1}\}_{v_1 \in V(G_1)}$ and $\mathcal{P}$ are legal, and $\mathcal{P}$ is a persistent pebbling of G_1. Then there is a configuration $\mathbb{P}'_\beta$ in $\mathcal{P}'$ of $\mathcal{R}(G_1, G_2)$ having at least $\mathsf{RPeb}(G_1) + \mathsf{RPeb}(G_2) + 1$ pebbles.*

Proof. Since $\mathcal{P}$ is a legal persistent pebbling of G_1, there is a configuration $\mathbb{P}_b$ having at least $\mathsf{RPeb}(G_1)$ pebbles. By definition of the algorithm $\mathbb{P}_b$ is created in Step (c) when an output node $(v_1, v_2)_{\text{out}}$ is pebbled or unpebbled to get to $\mathbb{P}'_t$ in $\mathcal{P}'$, so the configuration $\mathbb{P}_b$ is between $\mathbb{P}(t-1)$ and $\mathbb{P}(t)$. Let $\mathbb{P}'_\beta$ be $\mathbb{P}'_t$ if the output node $(v_1, v_2)_{\text{out}}$ is pebbled, and $\mathbb{P}'_{t-1}$ if the output node $(v_1, v_2)_{\text{out}}$ is unpebbled, then $\mathbb{P}'_\beta = \mathbb{P}'_{t-1} \cup \mathbb{P}'_t$.

Note that for any $u_1 \in \mathbb{P}_b$ we know that between $\mathbb{P}^{u_1}(t-1)$ and $\mathbb{P}^{u_1}(t)$, some configuration in $\mathcal{P}^{u_1}$ is not empty, because $u_1 \in \mathit{Ext}(t-1) \cup \mathit{Ext}(t) \subseteq D = \mathit{proj}^V(t-1) \cup \mathit{proj}^V(t)$. It follows from Claim E.8.12 that there is at least one pebble in every u_1-block of $\mathbb{P}'_\beta$.

If $\mathit{proj}^P(t-1) \neq \mathit{proj}^P(t)$, that is if there is $w_1 \in \mathit{proj}^P(t-1) \triangle \mathit{proj}^P(t)$, then we are done. Since exactly one of $\mathbb{P}^{w_1}(t-1)$ and $\mathbb{P}^{w_1}(t)$ is p-locked, there is a configuration between them having at least $\mathsf{RPeb}(G_2)$ pebbles, and by Claim E.8.12 at least $\mathsf{RPeb}(G_2)$ cells of the w_1-block have pebbles. Summing up, in $\mathbb{P}'_\beta$, there are

$$\underbrace{\mathsf{RPeb}(G_2)}_{w_1\text{ block}} + \underbrace{\mathsf{RPeb}(G_1) - 1}_{\text{other blocks}}$$

cells having pebbles, and among them the (v_1, v_2)-cell has three pebbles, for a total of $\mathsf{RPeb}(G_1) + \mathsf{RPeb}(G_2) + 1$ pebbles as needed.

Assume that $\mathit{proj}^P(t-1) = \mathit{proj}^P(t)$ instead. Recall $\mathit{Ext}(\cdot) = \mathit{proj}^P(\cdot) \cup \Big(\mathit{proj}^V(\cdot) \cap \mathit{unsur}_{G_1}(\mathit{proj}^P(\cdot))\Big)$. In Step (c) we know $\mathit{Ext}(t-1) \neq \mathit{Ext}(t)$, therefore at least one of $D_+ := \mathit{Ext}(t) \setminus \mathit{Ext}(t-1)$ and $D_- := \mathit{Ext}(t-1) \setminus \mathit{Ext}(t)$ is not empty. Swap t and

$t-1$ if necessary, by their symmetry in the rest of this lemma, assume $D_- \neq \emptyset$. Fix $w_1 \in D_- = \mathit{Ext}(t-1) \setminus \mathit{Ext}(t)$.

Since $\mathit{proj}^P(t-1) \setminus \mathit{Ext}(t) \subseteq \mathit{proj}^P(t-1) \setminus \mathit{proj}^P(t) = \emptyset$, we can assume that $w_1 \in \big(\mathit{proj}^V(t-1) \cap \mathit{unsur}_{G_1}(\mathit{proj}^P(t-1))\big) \setminus \mathit{Ext}(t)$.

In particular $w_1 \in \big(\mathit{proj}^V(t-1) \setminus \mathit{proj}^V(t)\big) \cap \mathit{unsur}_{G_1}(\mathit{proj}^P(t))$. Let $u_1 \in \mathit{pred}_{G_1}(w_1)$ be such that $u_1 \notin \mathit{proj}^P(t-1) = \mathit{proj}^P(t)$.

Now $w_1 \in \mathit{proj}^V(t-1) \setminus \mathit{proj}^V(t)$, and as in the proof of Claim E.8.11, we know that the w_1-block is surrounded in $\mathbb{P}'_{t-1}$ or in $\mathbb{P}'_t$. This implies that the sink of the u_1-block $(u_1, z_2)_{\text{out}}$ has a pebble, and so $z_2 \in \mathbb{P}^{u_1}(t-1)$ or $z_2 \in \mathbb{P}^{u_1}(t)$. But u_1 is not in the p-projection at either time, so there is some other pebble other than z_2 in $\mathbb{P}^{u_1}(t-1)$ and $\mathbb{P}^{u_1}(t)$. By Claim E.8.12, at least 2 cells of the u_1-block have pebbles.

Since $w_1 \in \mathit{proj}^V(t-1) \setminus \mathit{proj}^V(t)$, exactly one of $\mathbb{P}^{w_1}(t-1)$ and $\mathbb{P}^{w_1}(t)$ is v-locked so there is a configuration of $\mathcal{P}^{w_1}$ between them having at least $\mathsf{RPeb}^V(G_2)$ pebbles, and by Claim E.8.12 at least $\mathsf{RPeb}^V(G_2)$ cells of the w_1-block have pebbles. Summing up, in $\mathbb{P}'_\beta$, there are

$$\underbrace{\mathsf{RPeb}^V(G_2)+}_{w_1\text{ block}} \underbrace{\;2\;}_{u_1\text{ block}} \underbrace{+\mathsf{RPeb}(G_1)-2}_{\text{other blocks}} \geq \mathsf{RPeb}(G_1)+\mathsf{RPeb}(G_2)-1$$

cells having pebbles, and among them the (v_1, v_2)-cell has three pebbles, for a total of $\mathsf{RPeb}(G_1)+\mathsf{RPeb}(G_2)+1$ pebbles as needed. □

The product construction does not work for the standard black pebbling: there is no single constant C such that for any two graphs G_1 and G_2, $\mathsf{Peb}(\mathcal{R}(G_1, G_2)) = \mathsf{Peb}(G_1)+\mathsf{Peb}(G_2)+C$.

Indeed, if we take G_1 and G_2 to be the singleton graph, which has pebbling price 1, the product construction is the pyramid of height 1, which has pebbling price 3. This gives a value of 1 for C.

If we take G_1 and G_2 to be the path of length 1, which has pebbling price 2, the product construction has pebbling price 4. This gives a value of 0 for C.

If we take G_1 and G_2 to be the pyramid of height 1, which has pebbling price 3, the product construction has pebbling price 5. An optimal strategy is to pebble the sinks of the two G_2 copies corresponding to sources in G_1, then pebble all the exterior nodes of the remaining copy of G_2, unpebble the sinks and finish the pebbling. This gives a value of -1 for C.

E.9 Product Construction for Standard Pebbling

The part of the proof of Theorem E.3.4 that deals with standard pebbling uses as a black box the construction in Theorem E.3.5 for standard pebbling. Now we state it again and we give its full proof.

Theorem E.9.1. *Given two graphs G_1 and G_2 of standard pebbling price at least* 3*, there is a polynomial-time constructible graph $\mathcal{S}(G_1, G_2)$ of size $|G_1|(2|G_1| + |G_2|)$ with standard pebbling price $\mathsf{Peb}(\mathcal{S}(G_1, G_2)) = \mathsf{Peb}(G_1) + \mathsf{Peb}(G_2) - 1$.*

For the rest of the section we fix G_1 and G_2 to be two single sink directed acyclic graphs, with sinks z_1 and z_2, respectively. We set $p_1 := \mathsf{Peb}(G_1)$ and $p_2 := \mathsf{Peb}(G_2)$, and we assume that p_2 is at least 3.

Construction E.9.2 (Product for standard pebbling). A *centipede* of length ℓ is a path of length ℓ where all nodes but the source have an extra predecessor. Formally, it is the graph with vertices $\{r_0, \dots, r_\ell\}$, $\{s_1, \dots, s_\ell\}$ and edges $\{(r_{i-1}, r_i) : i \in [\ell]\} \cup \{(s_i, r_i) : i \in [\ell]\}$.

As the first step we define the graph $\hat{G_2}$ from G_2 as follows: $\hat{G_2}$ is the union of G_2 and of a centipede of length $|G_1|$, where we identify the sink of G_2 with the vertex r_0 of the centipede. Observe that the pebbling price of $\hat{G_2}$ is $\max(3, p_2) = p_2$.

The graph $\mathcal{S}(G_1, G_2)$ is as follows. For every vertex v_1 of G_1 we make a copy of $\hat{G_2}$, which we call the v_1-block. Then, for every edge (u_1, v_1) in G_1, we add edges from the sink of the u_1-block to all the sources of the centipede in the v_1-block. Formally, $\mathcal{S}(G_1, G_2)$ is the graph with vertices $\{(v_1, v_2) : v_1 \in V(G_1), v_2 \in V(\hat{G_2})\}$ and edges $\{((v_1, u_2), (v_1, v_2)) : v_1 \in V(G_1), (u_2, v_2) \in E(G_2)\}$, $\{((u_1, r_{|G_1|}), (v_1, s_i)) : (u_1, v_1) \in E(G_1), i \in [|G_1|]\}$.

Lemma E.9.3. *The standard pebbling price of $\mathcal{S}(G_1, G_2)$ is at most $p_1 + p_2 - 1$.*

Proof. To pebble $\mathcal{S}(G_1, G_2)$ we simulate a strategy for pebbling G_1 with p_1 pebbles. When a pebble is placed in v_1, we pebble the sink of the v_1-block using a strategy for $\hat{G_2}$ in p_2 pebbles. When a pebble is removed from v_1, we remove the pebble from the sink of the v_1-block. When we put a pebble in some v_1-block there are at most $p_1 - 1$ other non empty blocks and they all have exactly one pebble, therefore this strategy for $\mathcal{S}(G_1, G_2)$ is within the pebbling limit. □

Lemma E.9.4. *The standard pebbling price of $\mathcal{S}(G_1, G_2)$ is at least $p_1 + p_2 - 1$.*

Proof. Given a pebbling $\mathcal{P}^{\mathcal{S}} = (\mathbb{P}^{\mathcal{S}}_0, \mathbb{P}^{\mathcal{S}}_1, \dots, \mathbb{P}^{\mathcal{S}}_\tau)$ for the graph $\mathcal{S}(G_1, G_2)$ we construct a pebbling $\mathcal{P}$ for G_1 in such a way that if the space of $\mathcal{P}^{\mathcal{S}}$ is less than $p_1 + p_2 - 1$ then $\mathcal{P}$ has space less than p_1, which is impossible.

In particular for any configuration $\mathbb{P}^{\mathcal{S}}_t$ in $\mathcal{P}^{\mathcal{S}}$, we build a sequence $\mathcal{P}^{[t]}$ of pebbling configurations for G_1, such that the final pebbling $\mathcal{P}$ of G_1 is the concatenation of $\mathcal{P}^{[0]}, \mathcal{P}^{[1]}, \dots, \mathcal{P}^{[\tau]}$. While we build these sequences of configurations, we say that a vertex $v_1 \in V(G_1)$ is *active* if in the last configuration built so far v_1 does not have a pebble while all of its predecessors do.

If $\mathbb{P}^{\mathcal{S}}_t$ follows from a pebbling removal after which some v_1-block of $\mathcal{S}(G_1, G_2)$ becomes empty then $\mathcal{P}^{[t]}$ is the pebbling step that removes the pebble present on v_1, if any, otherwise $\mathcal{P}^{[t]}$ is the empty sequence.

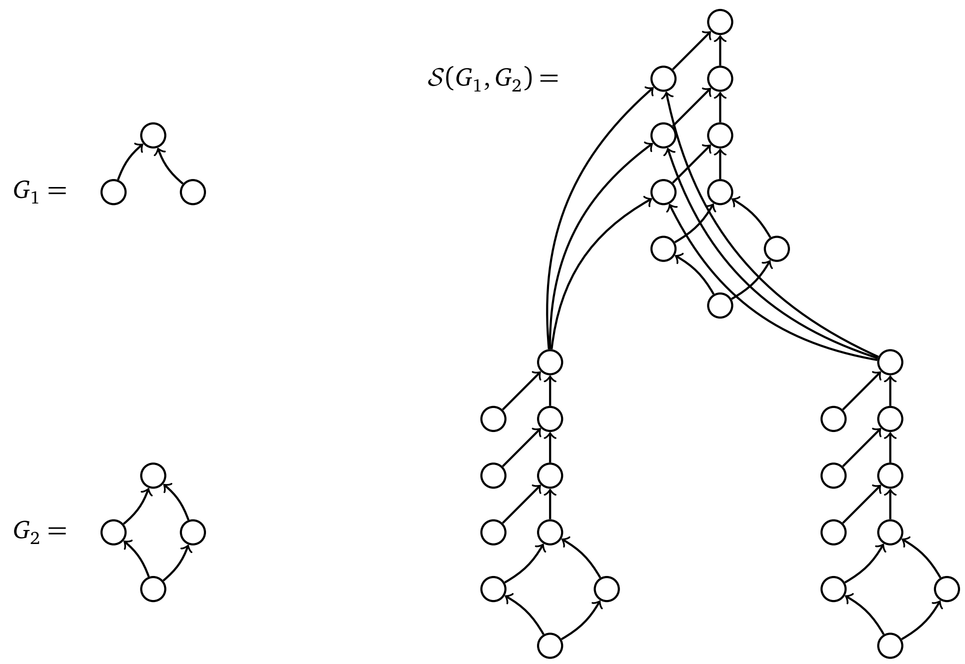


Figure E.20: Example of Construction E.9.2: product of a pyramid of height 1 and a rhombus.

If $\mathbb{P}_t^{\mathcal{S}}$ is the result of a pebble placement after which some v_1-block of $\mathcal{S}(G_1, G_2)$ contains p_2 pebbles, then $\mathcal{P}^{[t]}$ performs the following pebbling steps: place a pebble on each empty vertex $w_1 \in V(G_1)$ such that

1. the w_1-block of $\mathcal{S}(G_1, G_2)$ contains a pebble in $\mathbb{P}_t^{\mathcal{S}}$; and

2. w_1 is active.

This process is repeated until there is no vertex in $V(G_1)$ that meets both conditions. In particular a pebbling placement in the pebbling $\mathcal{P}^{\mathcal{S}}$ can cause a long chain of pebbling placements in $\mathcal{P}$.

Pebbling $\mathcal{P}$ is a legal standard pebbling of G_1 since removals are always legal, and pebbling placements are only done on active vertices by construction.

Claim E.9.5. *Assume the pebbling $\mathcal{P}^{\mathcal{S}}$ uses at most $p_1 + p_2 - 1$ pebbles. If there is a pebble on $(v_1, r_{|G_1|})$ in $\mathbb{P}_t^{\mathcal{S}}$, then there is a pebble on v_1 at the end of $\mathcal{P}^{[t]}$.*

Proof. We prove the claim by induction over a topological order of G_1. Consider the earliest time t_1 such that $\mathcal{P}^{\mathcal{S}}$ has p_2 pebbles in the v_1-block and there is a pebble in the

v_1-block during the whole interval $[t_1, t]$. Such a time exists because the v_1-block is a copy of $\hat{G}_2$, and p_2 pebbles are necessary to pebble its sink.

If v_1 is active at any point during the construction of $\mathcal{P}^{[t_1]}$, then v_1 is pebbled in that sequence and it is not removed afterwards. This is always the case if v_1 is a source of G_1.

If v_1 is not active we assume that Claim E.9.5 holds for all its predecessors. Time t_1 is the first time when there are p_2 pebbles in the v_1-block since it has been empty. Therefore none of the successors of (v_1, z_2) in the v_1-block has a pebble. Also, since at most $p_1 - 1 < |G_1|$ pebbles are outside the G_2 part of the v_1-block, some vertex (v_1, s_i) in the centipede part of the v_1-block has no pebble.

So far we discovered that at time t_1 there is a path $(v_1, s_i), (v_1, r_i), (v_1, r_{i+1}), \ldots, (v_1, r_{|G_1|})$ with no pebbles, and that at time t vertex $(v_1, r_{|G_1|})$ has a pebble. Then it must be the case that vertex (v_1, s_i) is pebbled at some time t_2 where $t_1 < t_2 < t$, and furthermore at time t_2 there must be pebbles on $(u_1, r_{|G_1|})$ and $(w_1, r_{|G_1|})$, where u_1 and w_1 are the predecessors of v_1 in graph G_1. By induction hypothesis u_1 and w_1 have a pebble at the end of $\mathcal{P}^{[t_2]}$, so v_1 is active at that point and, since the v_1-block is not empty, it gets a pebble. Such pebble stays in place at least until the end of $\mathcal{P}^{[t]}$. □

We can finally prove Lemma E.9.4. Assume for the sake of contradiction that $\mathcal{P}^{S}$ uses strictly less than $p_1 + p_2 - 1$ pebbles. Pebbling $\mathcal{P}$ is a legal pebbling of G_1 which pebbles the sink z_1, because of Claim E.9.5 and the fact that $\mathcal{P}^{S}$ pebbles vertex $(z_1, r_{|G_1|})$. Consider a configuration in which $\mathcal{P}$ reaches its maximum number of pebbles. This configuration is at the end of a sequence $\mathcal{P}^{[t]}$ corresponding to a pebble placement in $\mathcal{P}^{S}$ that causes some v_1-block to have p_2 pebbles, since this is the only case in which a sequence $\mathcal{P}^{[t]}$ adds pebbles.

The corresponding configuration $\mathbb{P}^{S}_{t}$ has p_2 pebbles in the v_1-block and at most $p_1 - 2$ other non empty blocks by assumption. Empty blocks in $\mathbb{P}^{S}_{t}$ corresponds to empty vertices in $\mathcal{P}^{[t]}$ by construction, so there are at most $p_1 - 1$ pebbles in all configurations in $\mathcal{P}^{[t]}$. This contradicts the fact that $\mathsf{Peb}(G_1) = p_1$. □

Observe that the only point where we used the fact that the length of a centipede is $|G_1|$ is to claim that there is one source without a pebble, so any length $u \geq \mathsf{Peb}(G_1)$ would suffice. Since in general it is PSPACE-hard to compute $\mathsf{Peb}(G_1)$, we settle for the trivial upper bound $|G_1|$.

E.10 Concluding Remarks

In this paper, we study the pebble game first introduced in [149] as well as the more restricted reversible pebble game in [31], where by [54] the latter game is also equivalent to the Dymond–Tompa game [74] and the Raz–McKenzie game [158].

We establish that it is PSPACE-hard to approximate standard and reversible pebbling price up to any additive constant. To the best of our knowledge, these are the first hardness of approximation results for such pebble games, even for polynomial time. It would be very interesting to show stronger inapproximability results for pebbling price under stronger assumptions. On the one hand, we are only able to show additive hardness, but on the other hand our results hold for arbitrary algorithms using a polynomial amount of memory. It seems reasonable to believe that the problem should become much harder for algorithms restricted to polynomial time, but showing this seems like a challenging task—in some sense, it appears that pebbling might be so hard a problem that it is even hard to prove that it is hard.

Another challenging problem is to prove approximation hardness, or even just PSPACE-completeness, for the *black-white pebble game* [67] modelling nondeterministic computation. This game is a strict generalization of the standard (black) pebble game, and so intuitively it should be at least as hard, but the added option of placing nondeterministic white pebbles anywhere in the graph completely destroys locality and makes the reduction in [88] break down. Hertel and Pitassi [102] showed a PSPACE-completeness result in the nonstandard setting when unbounded (and very large) fan-in is allowed. Essentially, the large fan-in makes it possible to lock down almost all pebbles in one place at a time (namely on the predecessors of a large fan-in vertex to be pebbled) and to completely rule out any use of white pebbles, reducing the whole problem to black pebbling (although this reduction, it should be stressed, is far from trivial). This approach does not work for bounded fan-in graphs, however, which is the standard setting studied in the 1970s and 80s and the setting that could potentially have interesting applications in, for instance, proof complexity.

We also show in this paper that standard black pebbling is asymptotically stronger than reversible pebbling by exhibiting families of DAGs over n vertices which have standard pebblings in space s but for which the reversible pebbling price is $\Omega(s \log n)$. Since any DAG on n vertices with standard pebbling price s can be reversibly pebbled in space $O(s^2 \log n)$, our separation is at most a linear factor (in $s \leq n$) off from the optimal. It would be interesting to determine how large the separation can be. We do not rule out the possibility that the separation we give might in fact be asymptotically optimal.

Acknowledgements

We are grateful to Anna Gál, Yuval Filmus, Toniann Pitassi, and Robert Robere for stimulating discussions on the topic of pebble games. A special thanks goes to Mladen Mikša, who participated in the initial stages of this work but somehow managed to avoid the pebbling addiction that seized the rest of us...

The first author performed part of this work while at Princeton University. The second, third and fourth authors were funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 279611. The third author was also supported by Swedish Research Council grants 621-2010-4797 and 621-2012-5645.

Bibliography

[1] Miklós Ajtai. The complexity of the pigeonhole principle. *Combinatorica*, 14(4): 417–433, 1994. Preliminary version in *FOCS '88*.

[2] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal on Computing*, 31(4): 1184–1211, 2002. Preliminary version in *STOC '00*.

[3] Michael Alekhnovich and Alexander A. Razborov. Lower bounds for polynomial calculus: Non-binomial case. *Proceedings of the Steklov Institute of Mathematics*, 242:18–35, 2003. Available at `http://people.cs.uchicago.edu/~razborov/files/misha.pdf`. Preliminary version in *FOCS '01*.

[4] Michael Alekhnovich and Alexander A. Razborov. Resolution is not automatizable unless W[P] is tractable. *SIAM Journal on Computing*, 38(4):1347–1363, October 2008. Preliminary version in *FOCS '01*.

[5] Noga Alon and Michael Capalbo. Smaller explicit superconcentrators. *Internet Mathematics*, 1(2):151–163, 2003.

[6] Noga Alon and Joel Spencer. *The probabilistic method*. Wiley-Interscience, 2nd edition, 2000.

[7] Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In *Proceedings of the 36th Annual International Cryptology Conference (CRYPTO '16)*, pages 241–271, August 2016.

[8] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC '15)*, pages 595–603, June 2015.

[9] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. Technical Report 2016/875, Cryptology ePrint Archive, September 2016.

[10] Ian Anderson. *Combinatorics of Finite Sets*. Oxford University Press, 1987.

[11] A. E. Andreev. On a method for obtaining lower bounds for the complexity of individual monotone functions. *Soviet Mathematics Doklady*, 31(3):530–534, 1985. English translation of a paper in *Doklady Akademii Nauk SSSR*.

[12] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, May 2008. Preliminary version in *CCC ’03*.

[13] Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *Journal of Artificial Intelligence Research*, 40:353–373, January 2011. Preliminary version in *SAT ’09*.

[14] Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint propagation as a proof system. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP ’04)*, volume 3258 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2004.

[15] Albert Atserias, Massimo Lauria, and Jakob Nordström. Narrow proofs may be maximally long. *ACM Transactions on Computational Logic*, 17:19:1–19:30, May 2016. Preliminary version in *CCC ’14*.

[16] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI ’09)*, pages 399–404, July 2009.

[17] Tomáš Balyo, Marijn J.H. Heule, and Matti Järvisalo. SAT competition 2016: Recent developments. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI ’17)*, pages 5061–5063, February 2017.

[18] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI ’97)*, pages 203–208, July 1997.

[19] Paul Beame, Chris Beck, and Russell Impagliazzo. Time-space tradeoffs in resolution: Superpolynomial lower bounds for superlinear space. *SIAM Journal on Computing*, 45(4):1612–1645, August 2016. Preliminary version in *STOC ’12*.

[20] Paul Beame, Trinh Huynh, and Toniann Pitassi. Hardness amplification in proof complexity. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC ’10)*, pages 87–96, June 2010.

[21] Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, December 2004. Preliminary version in *IJCAI ’03*.

[22] Paul Beame and Ashish Sabharwal. Non-restarting SAT solvers with simple preprocessing can efficiently simulate resolution. In *Proceedings of the 28th National Conference on Artificial Intelligence (AAAI '14)*, pages 2608–2615. AAAI Press, July 2014.

[23] Chris Beck, Jakob Nordström, and Bangsheng Tang. Some trade-off results for polynomial calculus. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 813–822, May 2013.

[24] Eli Ben-Sasson. Size-space tradeoffs for resolution. *SIAM Journal on Computing*, 38(6):2511–2525, May 2009. Preliminary version in *STOC '02*.

[25] Eli Ben-Sasson and Nicola Galesi. Space complexity of random formulae in resolution. *Random Structures and Algorithms*, 23(1):92–109, August 2003. Preliminary version in *CCC '01*.

[26] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585–603, September 2004.

[27] Eli Ben-Sasson and Jakob Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS '08)*, pages 709–718, October 2008.

[28] Eli Ben-Sasson and Jakob Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In *Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS '11)*, pages 401–416, January 2011.

[29] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, March 2001. Preliminary version in *STOC '99*.

[30] Charles H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, November 1973.

[31] Charles H Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, August 1989.

[32] Patrick Bennett, Ilario Bonacina, Nicola Galesi, Tony Huynh, Mike Molloy, and Paul Wollan. Space proof complexity for random 3-CNFs. Technical Report 1503.01613, arXiv.org, April 2015.

[33] Christoph Berkholz and Jakob Nordström. Supercritical space-width trade-offs for resolution. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP '16)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 57:1–57:14, July 2016.

[34] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[35] Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

[36] Béla Bollobás. The isoperimetric number of random regular graphs. *European Journal of Combinatorics*, 9:241–244, 1988.

[37] Béla Bollobás and Imre Leader. Edge-isoperimetric inequalities in the grid. *Combinatorica*, 11:299–314, 1991.

[38] Ilario Bonacina. Total space in resolution is at least width squared. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP '16)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:13, July 2016.

[39] Ilario Bonacina and Nicola Galesi. A framework for space complexity in algebraic proof systems. *Journal of the ACM*, 62(3):23:1–23:20, June 2015. Preliminary version in *ITCS '13*.

[40] Ilario Bonacina, Nicola Galesi, and Neil Thapen. Total space in resolution. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS '14)*, pages 641–650, October 2014.

[41] Ilario Bonacina, Nicola Galesi, and Neil Thapen. Total space in resolution. *SIAM Journal on Computing*, 45(5):1894–1909, October 2016. Preliminary version in *FOCS '14*.

[42] María Bonet, Toniann Pitassi, and Ran Raz. Lower bounds for cutting planes proofs with small coefficients. *Journal of Symbolic Logic*, 62(3):708–728, September 1997. Preliminary version in *STOC '95*.

[43] María Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM Journal on Computing*, 30(5):1462–1484, 2000. Preliminary version in *FOCS '98*.

[44] María Luisa Bonet and Nicola Galesi. Optimality of size-width tradeoffs for resolution. *Computational Complexity*, 10(4):261–276, December 2001. Preliminary version in *FOCS '99*.

[45] María Luisa Bonet, Sam Buss, and Jan Johannsen. Improved separations of regular resolution from clause learning proof systems. *Journal of Artificial Intelligence Research*, 49:669–703, 2014.

[46] R. B. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume A: Algorithms and Complexity*, pages 757–804. MIT Press, 1990.

[47] Michael Brickenstein and Alexander Dreyer. PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. *Journal of Symbolic Computation*, 44(9):1326–1345, September 2009.

[48] Harry Buhrman, John Tromp, and Paul Vitányi. Time and space bounds for reversible simulation. *Journal of physics A: Mathematical and general*, 34:6821–6830, 2001. Preliminary version in *ICALP '01*.

[49] Samuel R. Buss and Peter Clote. Cutting planes, connectivity and threshold logic. *Archive for Mathematical Logic*, 35:33–63, 1996.

[50] Samuel R. Buss, Jan Hoffmann, and Jan Johannsen. Resolution trees with lemmas: Resolution refinements that characterize DLL-algorithms with clause learning. *Logical Methods in Computer Science*, 4(4:13), December 2008.

[51] Samuel R. Buss and Leszek Kołodziejczyk. Small stone in pool. *Logical Methods in Computer Science*, 10:16:1–16:22, June 2014.

[52] David A. Carlson and John E. Savage. Graph pebbling with many free pebbles can be difficult. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC '80)*, pages 326–332, 1980.

[53] David A. Carlson and John E. Savage. Extreme time-space tradeoffs for graphs with small space requirements. *Information Processing Letters*, 14(5):223–227, 1982.

[54] Siu Man Chan. Just a pebble game. In *Proceedings of the 28th Annual IEEE Conference on Computational Complexity (CCC '13)*, pages 133–143, June 2013.

[55] Siu Man Chan. *Pebble Games and Complexity*. PhD thesis, University of California at Berkeley, 2013.

[56] Siu Man Chan, Massimo Lauria, Jakob Nordström, and Marc Vinyals. Hardness of approximation in PSPACE and separation results for pebble games (Extended abstract). In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS '15)*, pages 466–485, October 2015.

[57] Siu Man Chan and Aaron Potechin. Tight bounds for monotone switching networks via Fourier analysis. *Theory of Computing*, 10:389–419, October 2014. Preliminary version in *STOC '12*.

[58] Ashok K. Chandra. Efficient compilation of linear recursive programs. In *Proceedings of the 14th Annual Symposium on Switching and Automata Theory (SWAT '73)*, pages 16–25, 1973.

[59] Arkadev Chattopadhyay, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay. Composition and simulation theorems via pseudo-random properties. Technical Report TR17-014, Electronic Colloquium on Computational Complexity (ECCC), January 2017.

[60] Vašek Chvátal. Edmond polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(1):305–337, 1973.

[61] Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, October 1988.

[62] Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96)*, pages 174–183, May 1996.

[63] Anne Condon, Joan Feigenbaum, Carsten Lund, and Peter W Shor. Probabilistically checkable proof systems and nonapproximability of PSPACE-hard functions. *Chicago Journal of Theoretical Computer Science*, 1995, October 1995. Preliminary version in *STOC '93*.

[64] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC '71)*, pages 151–158, 1971.

[65] Stephen A. Cook. An observation on time-storage trade off. *Journal of Computer and System Sciences*, 9(3):308–316, 1974. Preliminary version in *STOC '73*.

[66] Stephen A. Cook and Robert Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, March 1979.

[67] Stephen A. Cook and Ravi Sethi. Storage requirements for deterministic polynomial time recognizable languages. *Journal of Computer and System Sciences*, 13 (1):25–37, 1976. Preliminary version in *STOC '74*.

[68] William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

[69] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[70] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[71] Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. How limited interaction hinders real communication (and what it means for proof and circuit complexity). In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS '16)*, pages 295–304, October 2016.

[72] Scott Diehl, Dieter van Melkebeek, and Ryan Williams. An improved time-space lower bound for tautologies. *Journal of Combinatorial Optimization*, 22(3): 325–338, October 2011. Preliminary version in *COCOON '09*.

[73] Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In *Proceedings of the 25th Annual International Cryptology Conference (CRYPTO '05)*, volume 3621 of *Lecture Notes in Computer Science*, pages 37–54. Springer, August 2005.

[74] Patrick W. Dymond and Martin Tompa. Speedups of deterministic machines by synchronous parallel machines. *Journal of Computer and System Sciences*, 30(2): 149–161, April 1985. Preliminary version in *STOC '83*.

[75] Jan Elffers, Jakob Nordström, Karem Sakallah, and Laurent Simon. Seeking practical CDCL insights from theoretical SAT benchmarks. Pragmatics of SAT Workshop, 2016.

[76] Paul Erdős, Ronald L. Graham, and Endre Szemerédi. On sparse graphs with dense long paths. Technical report, Stanford University, 1975.

[77] Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171(1):84–97, 2001. Preliminary versions of these results appeared in *STACS '99* and *CSL '99*.

[78] Yuval Filmus, Massimo Lauria, Mladen Mikša, Jakob Nordström, and Marc Vinyals. Towards an understanding of polynomial calculus: New separations and lower bounds (Extended abstract). In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP '13)*, volume 7965 of *Lecture Notes in Computer Science*, pages 437–448. Springer, July 2013.

[79] Yuval Filmus, Massimo Lauria, Jakob Nordström, Noga Ron-Zewi, and Neil Thapen. Space complexity in polynomial calculus. *SIAM Journal on Computing*, 44(4):1119–1153, August 2015. Preliminary version in *CCC '12*.

[80] Yuval Filmus, Jakob Nordström, Toniann Pitassi, and Yu Wu. Unpublished note, 2010.

[81] Yuval Filmus, Toniann Pitassi, Robert Robere, and Stephen A Cook. Average case lower bounds for monotone switching networks. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS '13)*, pages 598–607, November 2013.

[82] Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. A better-than-3n lower bound for the circuit complexity of an explicit function. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS '16)*, pages 89–98, October 2016.

[83] Noah Fleming, Denis Pankratov, Toniann Pitassi, and Robert Robere. Random CNFs are hard for cutting planes. Technical Report TR17-045, Electronic Colloquium on Computational Complexity (ECCC), March 2017.

[84] Lance Fortnow. Time-space tradeoffs for satisfiability. *Journal of Computer and System Sciences*, 60(2):337–353, April 2000. Preliminary version in *CCC '97*.

[85] Péter Frankl. A new short proof for the Kruskal–Katona theorem. *Discrete Mathematics*, 48(2):327–329, February 1984.

[86] Nicola Galesi and Massimo Lauria. Optimality of size-degree trade-offs for polynomial calculus. *ACM Transactions on Computational Logic*, 12:4:1–4:22, November 2010.

[87] Nicola Galesi, Pavel Pudlák, and Neil Thapen. The space complexity of cutting planes refutations. In *Proceedings of the 30th Annual Computational Complexity Conference (CCC '15)*, volume 33 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 433–447, June 2015.

[88] John R. Gilbert, Thomas Lengauer, and Robert Endre Tarjan. The pebbling problem is complete in polynomial space. *SIAM Journal on Computing*, 9(3): 513–524, August 1980. Preliminary version in *STOC '79*.

[89] John R. Gilbert and Robert Endre Tarjan. Variations of a pebble game on graphs. Technical Report STAN-CS-78-661, Stanford University, 1978. Available at `http://infolab.stanford.edu/TR/CS-TR-78-661.html`.

[90] Andreas Goerdt. The cutting plane proof system with bounded degree of falsity. In *Proceedings of the 5th Workshop on Computer Science Logic (CSL '91)*, pages 119–133, October 1991.

[91] Ralph E. Gomory. An algorithm for integer solutions of linear programs. In R.L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

[92] Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC '15)*, pages 257–266, June 2015.

[93] Mika Göös, T.S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication vs. partition number. Technical Report TR15-169, Electronic Colloquium on Computational Complexity (ECCC), October 2015.

[94] Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. *SIAM Journal on Computing*, 45(5): 1835–1869, 2016. Preliminary version in *STOC '15*.

[95] Mika Göös and Toniann Pitassi. Communication lower bounds via critical block sensitivity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC '14)*, pages 847–856, May 2014.

[96] Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS '15)*, pages 1077–1088, October 2015.

[97] Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for BPP. Technical Report TR17-053, Electronic Colloquium on Computational Complexity (ECCC), March 2017.

[98] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39 (2-3):297–308, August 1985.

[99] Armin Haken and Stephen A. Cook. An exponential lower bound for the size of monotone real circuits. *Journal of Computer and System Sciences*, 58(2):326–335, 1999.

[100] Hamed Hatami, Kaave Hosseini, and Shachar Lovett. Structure of protocols for XOR functions. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS '16)*, pages 282–288, October 2016.

[101] Philipp Hertel, Fahiem Bacchus, Toniann Pitassi, and Allen Van Gelder. Clause learning can effectively P-simulate general propositional resolution. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI '08)*, pages 283–290, July 2008.

[102] Philipp Hertel and Toniann Pitassi. The PSPACE-completeness of black-white pebbling. *SIAM Journal on Computing*, 39(6):2622–2682, April 2010. Preliminary version in *FOCS '07*.

[103] Edward A. Hirsch, Arist Kojevnikov, Alexander S. Kulikov, and Sergey I. Nikolenko. Complexity of semialgebraic proofs with restricted degree of falsity. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:53–69, 2008. Preliminary version in *SAT '05* and *SAT '06*.

[104] John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM*, 24(2):332–337, April 1977. Preliminary version in *FOCS '75*.

[105] Pavel Hrubeš and Pavel Pudlák. Random formulas, monotone circuits, and interpolation. Technical Report TR17-042, Electronic Colloquium on Computational Complexity (ECCC), March 2017.

[106] Trinh Huynh and Jakob Nordström. On the virtue of succinct proofs: Amplifying communication complexity hardness to time-space trade-offs in proof complexity (Extended abstract). In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC '12)*, pages 233–248, May 2012.

[107] Harry B. Hunt III, Madhav V. Marathe, and Richard Edwin Stearns. Generalized CNF satisfiability problems and non-efficient approximability. In *Proceedings of the 9th Annual IEEE Conference on Structure in Complexity Theory (Structures '94)*, pages 356–366, June 1994.

[108] Russell Impagliazzo and Valentine Kabanets. Constructive proofs of concentration bounds. In *Proceedings of the 13th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 14th International Workshop on Randomization and Computation (APPROX-RANDOM '10)*, pages 617–631, 2010.

[109] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, March 2001. Preliminary version in *CCC '99*.

[110] Russell Impagliazzo, Toniann Pitassi, and Alasdair Urquhart. Upper and lower bounds for tree-like cutting planes proofs. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science (LICS '94)*, pages 220–228, July 1994.

[111] Russell Impagliazzo, Pavel Pudlák, and Jiří Sgall. Lower bounds for the polynomial calculus and the Gröbner basis algorithm. *Computational Complexity*, 8(2): 127–144, 1999.

[112] Svante Janson, Tomasz Łuczak, and Andrzej Ruciński. *Random graphs*. Wiley-Interscience, 2000.

[113] Jan Johannsen. Lower bounds for monotone real circuit depth and formula size and tree-like cutting planes. *Information Processing Letters*, 67(1):37–41, July 1998.

[114] Jan Johannsen. Depth lower bounds for monotone semi-unbounded fan-in circuits. *RAIRO-Theoretical Informatics and Applications*, 35(3):277–286, 2001.

[115] Stasys Jukna. *Extremal Combinatorics with Applications in Computer Science*. Springer, 2nd edition, 2011.

[116] Matti Järvisalo, Arie Matsliah, Jakob Nordström, and Stanislav Živný. Relating proof complexity measures and practical hardness of SAT. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 316–331. Springer, October 2012.

[117] Balasubramanian Kalyanasundaram and George Schnitger. On the power of white pebbles. *Combinatorica*, 11(2):157–171, June 1991. Preliminary version in *STOC '88*.

[118] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM Journal on Discrete Mathematics*, 3(2):255–265, 1990. Preliminary version in *STOC '88*.

[119] Gyula O. H. Katona. A theorem of finite sets. In *Theory of Graphs*, pages 187–207. Akadémiai Kiadó, 1968.

[120] George Katsirelos, Ashish Sabharwal, Horst Samulowitz, and Laurent Simon. Resolution and parallelizability: Barriers to the efficient parallelization of SAT solvers. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI '13)*, July 2013.

[121] Jeong Han Kim and Nicholas C. Wormald. Random matchings which induce Hamilton cycles, and hamiltonian decompositions of random regular graphs. *Journal of Combinatorial Theory B*, 81:20–44, 2001.

[122] Maria Klawe, Wolfgang J. Paul, Nicholas Pippenger, and Mihalis Yannakakis. On monotone formulae with restricted depth. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing (STOC '84)*, pages 480–487, 1984.

[123] Maria M. Klawe. A tight bound for black and white pebbles on the pyramid. *Journal of the ACM*, 32(1):218–228, January 1985. Preliminary version in *FOCS '83*.

[124] Jan Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62(2): 457–486, June 1997.

[125] Jan Krajíček. Interpolation by a game. *Mathematical Logic Quarterly*, 44:450–458, 1998.

[126] Richard Královič. Time and space complexity of reversible pebbling. *RAIRO – Theoretical Informatics and Applications*, 38(02):137–161, April 2004.

[127] Joseph B. Kruskal. The number of simplices in a complex. In Richard Bellman, editor, *Mathematical Optimization Techniques*, pages 251–278. University of California Press, 1963.

[128] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

[129] Klaus-Jörn Lange, Pierre McKenzie, and Alain Tapp. Reversible space equals deterministic space. *Journal of Computer and System Sciences*, 60(2):354–367, April 2000.

[130] Thomas Lengauer and Robert Endre Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *Journal of the ACM*, 29(4):1087–1130, October 1982. Preliminary version in *STOC '79*.

[131] Leonid A. Levin. Universal'nye zadachi perebora. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973. In Russian.

[132] Ming Li, John Tromp, and Paul Vitányi. Reversible simulation of irreversible computation. *Physica D: Nonlinear Phenomena*, 120:168–176, September 1998.

[133] Ming Li and Paul Vitányi. Reversibility and adiabatic computation: Trading time and space for energy. *Proceedings of the Royal Society of London, Series A*, 452 (1947):769–789, April 1996.

[134] Andrzej Lingas. A PSPACE-complete problem related to a pebble game. In *Proceedings of the 5th Colloquium on Automata, Languages and Programming (ICALP '78)*, pages 300–321, 1978.

[135] László Lovász, Moni Naor, Ilan Newman, and Avi Wigderson. Search problems in the decision tree model. *SIAM Journal on Discrete Mathematics*, 8(1):119–132, 1995. Preliminary version in *FOCS '91*.

[136] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.

[137] Friedhelm Meyer auf der Heide. A comparison of two variations of a pebble game on graphs. *Theoretical Computer Science*, 13(3):315–322, 1981.

[138] M. Morgenstern. Existence and explicit constructions of $q+1$ regular Ramanujan graphs for every prime power q. *Journal of Combinatorial Theory, Series B*, 62(1): 44–62, 1994.

[139] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

[140] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

[141] Noam Nisan and Avi Wigderson. Rounds in communication complexity revisited. *SIAM Journal on Computing*, 22(1):211–219, February 1993. Preliminary version in *STOC '91*.

[142] Jakob Nordström. Narrow proofs may be spacious: Separating space and width in resolution. *SIAM Journal on Computing*, 39(1):59–121, May 2009. Preliminary version in *STOC '06*.

[143] Jakob Nordström. A simplified way of proving trade-off results for resolution. *Information Processing Letters*, 109(18):1030–1035, August 2009. Preliminary version in ECCC report TR07-114, 2007.

[144] Jakob Nordström. On the relative strength of pebbling and resolution. *ACM Transactions on Computational Logic*, 13(2):16:1–16:43, April 2012. Preliminary version in *CCC '10*.

[145] Jakob Nordström. Pebble games, proof complexity and time-space trade-offs. *Logical Methods in Computer Science*, 9:15:1–15:63, September 2013.

[146] Jakob Nordström. New wine into old wineskins: A survey of some pebbling classics with supplemental results. Manuscript in preparation. To appear in *Foundations and Trends in Theoretical Computer Science*. Current draft version available at `http://www.csc.kth.se/~jakobn/research/`, 2017.

[147] Jakob Nordström and Johan Håstad. Towards an optimal separation of space and length in resolution. *Theory of Computing*, 9:471–557, May 2013. Preliminary version in *STOC '08*.

[148] Alessandro Panconesi and Aravind Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. *SIAM Journal on Computing*, 26(2):350–368, 1997.

[149] Michael S. Paterson and Carl E. Hewitt. Comparative schematology. In *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*, pages 119–127, 1970.

[150] Wolfgang J. Paul and Rüdiger Reischuk. On alternation II. A graph theoretic approach to determinism versus non-determinism. *Acta Informatica*, 14(4): 391–403, 1980. Preliminary version in *GITCS '79*.

[151] Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. *Mathematical Systems Theory*, 10:239–251, 1977.

[152] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175:512–525, February 2011. Preliminary version in *CP '09*.

[153] Nicholas Pippenger. Pebbling. Technical Report RC8258, IBM Watson Research Center, 1980. in Proceedings of the 5th IBM Symposium on Mathematical Foundations of Computer Science, Japan.

[154] Nicholas Pippenger and Leslie G. Valiant. Shifting graphs and their applications. *Journal of the ACM*, 23:423–432, July 1976.

[155] Aaron Potechin. Bounds on monotone switching networks for directed connectivity. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS '10)*, pages 553–562, October 2010.

[156] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, September 1997.

[157] Anup Rao and Amir Yehudayoff. Communication complexity. Manuscript in preparation, 2016.

[158] Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, March 1999. Preliminary version in *FOCS '97*.

[159] Alexander A. Razborov. Lower bounds for the monotone complexity of some Boolean functions. *Soviet Mathematics Doklady*, 31(2):354–357, 1985. English translation of a paper in *Doklady Akademii Nauk SSSR*.

[160] Alexander A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, December 1998.

[161] Alexander A. Razborov. A new kind of tradeoffs in propositional proof complexity. *Journal of the ACM*, 63:16:1–16:14, April 2016.

[162] Alexander A. Razborov. On the width of semi-algebraic proofs and algorithms. Technical Report TR16-010, Electronic Colloquium on Computational Complexity (ECCC), January 2016.

[163] Alexander A. Razborov. Proof complexity and beyond. *ACM SIGACT News*, 47(2): 66–86, June 2016.

[164] Søren Riis. *Independence in Bounded Arithmetic*. PhD thesis, University of Oxford, 1993.

[165] Arnold Rosenbloom. Monotone real circuits are more powerful than monotone boolean circuits. *Information Processing Letters*, 61(3):161–164, February 1997.

[166] Rahul Santhanam. Lower bounds on the complexity of recognizing SAT by Turing machines. *Information Processing Letters*, 79(5):243–247, September 2001.

[167] John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, 1998. Available at `http://www.modelsofcomputation.org`.

[168] John E. Savage and Sowmitri Swamy. Space-time tradeoffs for oblivious interger multiplications. In *Proceedings of the 6th International Colloquium on Automata, Languages and Programming (ICALP '79)*, pages 498–504, 1979.

[169] Georg Schnitger. On depth-reduction and grates. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science (FOCS '83)*, pages 323–328, November 1983.

[170] Nathan Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):417–481, December 2007.

[171] Ravi Sethi. Complete register allocation problems. *SIAM Journal on Computing*, 4(3):226–248, September 1975.

[172] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC '73)*, pages 1–9, 1973.

[173] Sowmitri Swamy and John E. Savage. Space-time trade-offs on the FFT-algorithm. Technical Report CS-31, Brown University, 1977.

[174] Sowmitri Swamy and John E. Savage. Space-time tradeoffs for linear recursion. *Mathematical Systems Theory*, 16(1):9–27, 1983.

[175] Neil Thapen. A trade-off between length and width in resolution. Technical Report TR14-137, Electronic Colloquium on Computational Complexity (ECCC), October 2014.

[176] Martin Tompa. Time-space tradeoffs for computing functions, using connectivity properties of their circuits. In *Proceedings of the 10th annual ACM symposium on Theory of computing (STOC '78)*, pages 196–204, 1978.

[177] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1): 209–219, January 1987.

[178] Leslie G. Valiant. Parallelism in comparison problems. *SIAM Journal on Computing*, 4(3):348–355, March 1975.

[179] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *Proceedings of the 6th International Symposium on Mathematical Foundations of Computer Science (MFCS '77)*, pages 162–176, September 1977.

[180] Allen Van Gelder. Pool resolution and its relation to regular resolution and DPLL with clause learning. In *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '05)*, volume 3835 of *Lecture Notes in Computer Science*, pages 580–594. Springer, 2005.

[181] Dieter van Melkebeek. A survey of lower bounds for satisfiability and related problems. *Foundations and Trends in Theoretical Computer Science*, 2(3):197–303, October 2007.

[182] H. Venkateswaran and Martin Tompa. A new pebble game that characterizes parallel complexity classes. *SIAM Journal on Computing*, 18(3):533–549, June 1989. Preliminary version in *FOCS '86*.

[183] Ryan Williams. Space-efficient reversible simulations. Technical report, Cornell University, 2000. Available at `http://web.stanford.edu/~rrwill/spacesim9_22.pdf`.

[184] Ryan Williams. Time-space tradeoffs for counting NP solutions modulo integers. *Computational Complexity*, 17(2):179–219, May 2008. Preliminary version in *CCC '07*.

[185] Nicholas C. Wormald. Models of random regular graphs. In *Surveys in Combinatorics*, pages 239–298. Cambridge University Press, 1999.

[186] Yu Wu, Per Austrin, Toniann Pitassi, and David Liu. Inapproximability of treewidth and related problems. *Journal of Artificial Intelligence Research*, 49:569–600, April 2014.

[187] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in Boolean satisfiability solver. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '01)*, pages 279–285, November 2001.