Red-Blue and Standard Pebble Games: Complexity and Applications in the Sequential and Parallel Models

by

Quanquan Liu

S.B., Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2017

Author .....
Department of Electrical Engineering and Computer Science
February 3, 2017

Certified by .....
Erik D. Demaine
Professor, Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Dr. Christopher Terman
Chairman, Masters of Engineering Thesis Committee

Abstract

Pebble games are games played on directed acyclic graphs involving placing and moving pebbles on nodes of the graph according to a certain set of rules. The standard pebble game (also known as the black pebble game) is the first of such games played on DAGs. The game itself involves three simple rules: a pebble can be placed on any leaf node (a node without any predecessors), a pebble can be moved or placed on a non-leaf node if all of its predecessors are pebbled, and a pebble can be deleted from any node at any time. Generally, the standard pebble game is used to model space-bounded computation. Each node represents a result of a computation and placing a pebble on a node represents performing a deterministic computation of a result using previously computed results.

The standard pebble game has been used in a variety of applications including register allocation, VLSI design, compilers, and, more recently, propositional proof complexity and memory-hard functions. Much previous research has been done in analyzing the computational complexity of the standard pebble game in a variety of settings. It has been shown previously that computing an optimal strategy using the standard pebbling game on any given DAG is PSPACE-hard [GLT79]. Furthermore, it was more recently shown that the standard pebble game is hard to approximate to any constant *additive* factor [CLNV15]. In this thesis, we present a simpler proof of the result presented in [CLNV15] and strengthen the result to include any *polynomial* additive factor. In particular, we strengthen the result to show that it is PSPACE-hard to determine the minimum number of pebbles to any $n^{1-\varepsilon}$ additive factor for any ε.

The red-blue pebble game was introduced by [JWK81] as a model of I/O complexity and is also played on DAGs. Despite its importance in applications such as data access complexity and more recently to I/O-complexity in multi-level memory hierarchies [CRSS16], little is known about the computational complexity of determining the minimum number of red pebbles and transitions used when pebbling a given DAG using the rules of the game. In this thesis, we show that the red-blue pebble game is PSPACE-hard and that the red-blue pebble game with no deletions and allowing overwrites is NP-Complete. We also show that the red-blue pebble game parameterized by the number of transitions, k, is W[1]-hard.

In addition to the stated hardness results, we also introduce a graph family that takes $\Omega(n^k)$ time to pebble given k constant number of pebbles. This graph family partially answers an open question posed in [Nor15] regarding whether such a family that meets the $O(n^k)$ time upper bound exists for constant k pebbles. Furthermore, the graph family can be generalized for any $k < \sqrt{n}$. Given k pebbles where $k = \omega(1)$, the pebbling time is $\Omega(\frac{n^k}{k})$ for any graphs with n nodes in this family.

Finally, we present a new complexity measure, called *decremental complexity*, based on the sequential and parallel pebbling models. This complexity measure is concerned with the decrease in pebbling time when switching from the sequential to the parallel model of pebbling any given graph with n nodes using the minimum number of pebbles. Graphs with low decremental complexity have potential applications in proofs of work/space and memory-hard functions. In this thesis, we determine the decremental complexity of several common families of graphs as well as composite graphs consisting of many copies of instances of these families.

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Erik Demaine, for having provided me with invaluable aid throughout the course of this thesis. Prof. Demaine first introduced the topic of this thesis to me and have been immediately available throughout the course of the past year to discuss ideas and provide pointers on new directions to explore, new problems to look at, and comments on writing and presenting my ideas in a clear and precise manner–all points of which I am very grateful.

Secondly, I would like to thank Sunoo Park for helpful discussions on the cryptographic applications of this topic and for working with me through the surprisingly large number of papers published on this topic within the last year and a half. I also thank the following people for the help provided in various ways such as suggesting and providing copies of research papers related to the topic of pebble games or for nice intellectual discussions related to the topic of this thesis: Jeremiah Blocki, Tadge Dryja, Jayson Lynch, Jakob Nordström, Aaron Potechin, Ling Ren, Alessandra Scafuro, and Gary Wang. Special thanks to Jakob Nordström for providing a copy of his *New Wine into Old Wineskins: A Survey of Some Pebbling Classics with Supplemental Results* manuscript which has been an invaluable resource for discovering past research on this topic.

Finally, I would like to thank my wonderful family and friends who supported me through this thesis and listened and comforted me during the frantic rush in the weeks/days/hours before deadlines.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Pebble games were originally introduced to study compiler operations and programming languages. For example, a DAG represents the computational dependency of an operation on a set of previous operations and pebbles represent register allocation. Minimizing the amount of resources allocated to perform a computation is accomplished by minimizing the number of pebbles placed on the graph [Set75] and the time of computation, specifically parallel computation, is modeled by the Dymond and Tompa pebble game played by two players [DT85]. In addition to the standard pebble game (also known as the black pebble game in the literature) and the Dymond and Tompa pebble game, there are a number of other pebble games that are useful for studying computation. The *red-blue pebble game* is used to study I/O complexity [JWK81], the *reversible pebble game* is used to model reversible computation [Ben89], and the *black-white pebble game* is used to model non-deterministic straight-line programs [CS74]. A quick summary of these pebble games, their applications, and current hardness results are given in Section 1.2.

Previous research has shown upper bounds for the number of pebbles needed to pebble a graph in any of the one-player pebble games mentioned above except the red-blue pebble game since the minimization constraint is also different there. Hopcroft et al. [HPV77a] showed that any graph with bounded in-degree and containing n vertices requires at most $O(n/\log n)$ pebbles in the standard pebble game, meaning a computation that takes time n requires space $O(n/\log n)$. Furthermore, [DT85] showed that a graph with n nodes and bounded-indegree takes $O(n/\log n)$ time

Pebble Game	Upper Bound	Lower Bound
Standard	$O(n/\log n)$	$\Omega(n/\log n)$
Black-White	$O(n/\log n)$	$\Omega(n/\log n)$
Reversible	$O(n/\log n)$	$\Omega(n/\log n)$
Dymond-Tompa	$O(n/\log n)$	$\Omega(n/\log n)$

Table 1.1: Upper and lower bounds on pebbling costs for 4 types of pebble games [HPV77a, DT85, Cha13, PTC76, GT78]

to pebble in the two-player Dymond and Tompa pebble game. This means that any deterministic computation carried out in n time can be carried out in $O(n/\log n)$ alternating time (alternating time can be measured on an alternating machine which is useful for estimating the time required for parallel computation). Similarly, the upper bound on pebbling price for the black-white pebbling of a DAG is $O(n/\log n)$ since we can ignore the use of white pebbles. The reversible pebble game on a bounded in-degree graph has cost $O(n/\log n)$ [Cha13]. A summary of these upper bounds can be seen in Table 1.1.

In addition to upper bounds, [PTC76] showed that there exists an n-node DAG with bounded in-degree such that the pebbling cost of the graph is $\Omega(n/\log n)$ and consequently, the price of the Dymond and Tompa game on the same graph is $\Omega(n/\log n)$ and the cost of the reversible pebbling game is also $\Omega(n/\log n)$. Furthermore, [GT78] showed that there exists a family of DAGs such that the cost of black-white pebbling is $\Omega(n/\log n)$. These lower bounds are also summarized in Table 1.1.

For the red-blue pebble game, upper and lower bounds are derived for the number of transitions needed to pebble the graph given that the maximum number of red pebbles that can be used is r. The minimum number of transitions needed to pebble a FFT digraph given n nodes and r red pebbles is lower bounded by $\Omega(n\log n/\log r)$ [JWK81, AV88] and also upper bounded by $O(n\log n/\log r)$. Hong and Kung et al. [JWK81] also proved bounds on the minimum number of transitions for several other types of computations outlined in their paper. These time bounds are shown in Table 1.2. However, to the best of the author's knowledge, an upper bound on the minimum number of transitions for general DAGs with n nodes is not yet known.

Despite somewhat extensive research on the upper and lower bounds of optimally pebbling a

Type of Computation (i.e. Type of Graph)	Bound on Minimum Number of Transitions
FFT digraph	$\Theta(n \log_r n)$ [JWK81]
Multiplication of Two Square Matrices	$\Theta(n^3/\sqrt{r})$ [JWK81]
Multiplication of Matrix by Vector	$\Theta(n^2/r)$ [JWK81]
Odd-even Transposition Sort	$\Theta(n^2/r)$ [JWK81]
General DAGs	$\Omega(n^3/\sqrt{r})$ [JWK81]

Table 1.2: Upper and lower bounds on transition costs for red-blue pebble games on certain types of graphs [JWK81, AV88].

DAG in these games, the complexity of finding a minimum solution has fewer results. In fact, it is not yet known whether it is hard to find the minimum number of pebbles within even a non-constant *additive* factor [CLNV15]. It turns out that finding a strategy to optimally pebble a graph in the standard pebble game is computationally difficult even when each vertex is allowed to be pebbled only once. Specifically, finding the minimum number of black pebbles needed to pebble a DAG in the standard pebble game is PSPACE-complete [GLT79] and finding the minimum number of black pebbles needed in the one-shot case is NP-complete [Set75]. In addition, finding the minimum number of pebbles in both the black-white and reversible pebble games have been very recently shown to be both PSPACE-complete [CLNV15, HP10]. A summary of these results is shown in Table 1.3.

Despite having some results in the hardness of finding the minimum number of pebbles needed to pebble these DAGs using the rules of the aforementioned pebble games, there are still some number of gaps in our knowledge of these games. There are currently no results on whether the solutions could be approximated to within a constant factor of the optimal or whether a fixed-parameter tractable algorithm can be found for these games. In this thesis, we discuss some new results in the fixed parameter tractability of finding the minimum number of red pebbles and transitions needed in the red-blue pebble games. We also introduce a new red-blue pebble game paradigm that we prove to be NP-complete to solve for the minimum number of red pebbles and transitions. Furthermore, we present some hardness results in approximating the minimum number of pebbles in the standard pebble game, in particular, improving upon a result presented in [CLNV15]. We also make some observations in the complexity of approximating other pebble games such as the

Type of Game	Unbounded In-degree	Bounded In-degree	Approximation (Additive)	FPT
Standard (one-shot)	NP-complete [Set75]	?	?	FPT [Thm 7.1]
Standard	PSPACE-complete	PSPACE-complete [GLT79]	$n^{1-\varepsilon}$ [Thm 5.1]	?
Black-white	PSPACE-complete [HP10]	?	?	?
Reversible	PSPACE-complete	PSPACE-complete [CLNV15]	PSPACE-complete (Constant Factor) [CLNV15]	?
Red-Blue	PSPACE-complete [Thm 3.2]	PSPACE-complete [Thm 3.2]	?	W[1]-hard k Transitions [Thm 4.1]
Red-Blue (no deletion)	NP-complete [Thm 3.1]	NP-complete [Thm 3.1]	?	?

Table 1.3: Hardness of various pebble games with minimizing parameters such as the minimum pebbling number or minimum number of pebbling moves/transitions.

one-shot standard pebble game by utilizing a variety of previous results available in the literature.

1.2 Pebble Games and Previous Hardness Results

1.2.1 Standard Pebble Game

The standard pebble game played on DAGs has two variations [PH70]. For the one-shot pebble game, pebbles can be placed on a vertex at most once (hence the graph has to be pebbled in one-shot). In the standard pebble game, a vertex can be pebbled multiple times. The rules of the standard pebble game are as follows:

1. A pebble can be placed on any leaf (i.e. a vertex that has no predecessors).

2. A pebble can be removed from any vertex.

3. A pebble can be placed on a non-leaf vertex if and only if its direct predecessors are pebbled.

4. A pebble can be moved from a predecessor to a direct successor if and only if all the successor's direct predecessors are pebbled.

The goal of both variations is to place a pebble on a set of target vertices. As mentioned above, the standard pebble game can be used to model computational resource usage. The final vertices that need to be pebbled thus represent the results that must be obtained from the computation.

Sethi [Set75] proved that the one-shot pebbling game is NP-complete given a DAG with unbounded in-degree. Gilbert et al. [GLT79] proved that the pebbling game is PSPACE-complete even when it is played on graphs with bounded in-degree. More recently, Chan et al. [CLNV15] showed that estimating the optimal pebbling price to within a *constant additive* factor is PSPACE-complete.

1.2.2 Red-Blue Pebble Game

The red-blue pebble game was introduced to model I/O complexity [JWK81, AV88]. The rules of the game are as follows:

1. A red pebble can be placed on any leaf node if and only if the leaf node contains a blue pebble.

2. A red pebble can be placed on any non-leaf node if and only if all of its predecessors are pebbled with red pebbles.

3. A blue pebble can be placed on any node that contains a red pebble.

4. A red pebble can be placed on any node that contains a blue pebble.

5. A pebble can be deleted from a node at any time.

6. All source nodes (nodes without incoming edges) contain blue pebbles at the begining of the computation (i.e. at time $t = 0$).

Red pebbles represent space in *fast* memory (i.e. cache) and blue pebbles represent space in *slow* memory (i.e. disk). Suppose, as in real systems, that there is a limited amount of fast memory

and an unlimited amount of slow memory. In terms of the red-blue pebble game, this means that the number of red pebbles is limited by some upper bound, r. One would like to minimize the number of transitions, k, of data between fast and slow memory. The goal is to pebble a certain set of vertices using red pebbles. In the I/O interpretation of the game, pebbling with red pebbles represent performing a computation and saving the result in fast memory while pebbling all the vertices means computing all the results and maintaining some record of the results either in fast or slow memory.

While much has been done in showing upper and lower bounds in pebbling price in terms of number of red pebbles and number of transitions of pebbling certain types of DAGs, the computational complexity of finding the exact number of minimum red pebbles used and the minimum number of transitions has not been studied in the past to the best of the authors' knowledge.

More recently, the model of one-shot red-blue pebble games was introduced in [CRSS16]. This pebble game is used to model I/O-complexity *without recomputating any calculations in cache.* They also show how to extend this model to the multi-level memory hierarchy case.

1.2.3 Reversible Pebble Game

The reversible pebble game was introduced by [Ben89] to study energy efficient algorithms such that any computation that is performed could be reversed. The rules of the reversible pebble game are the following.

1. A pebble can be placed on a vertex if and only if all of its direct predecessors are pebbled or if the vertex is a leaf.

2. A pebble can only be removed from a vertex if and only if all of its direct predecessors are pebbled.

The rules of the reversible pebble game is the same as the standard pebble game except that the rules of adding and deleting pebbles from a vertex are now symmetric. Solving this game and approximating the game to a constant additive factor has been shown recently to be PSPACE-complete [CLNV15] even in DAGs with bounded in-degree 2.

1.2.4 Black-White Pebble Game

The black-white pebble game is used to model non-deterministic straight-line programs [CS74]. The problem this game seeks to answer is whether non-determinism can save space (i.e. reduce the number of registers used to perform a computation). As in the standard pebble game, a black pebble represents a deterministically computed value. Each white pebble represents a non-deterministically determined value that is verified later on. The rules of the black-white pebble game are the following:

1. A white pebble can be placed on a vertex at any time.
2. A white pebble on a vertex can be turned black if all of its immediate predecessors are pebbled.
3. A black pebble can be removed from a vertex at any time.
4. All vertices are pebble-free at the end of the computation.

The goal of this game is to pebble and remove the vertices at least once from all nodes in the given DAG. Finding an optimal pebbling of the vertices was determined to be PSPACE-complete for DAGs with unbounded in-degree [HP10]. It is currently not known whether the problem is hard for DAGs with bounded in-degree.

While many results in previous literature hold for both the standard (black) pebble game and the black-white pebble game, in this thesis, we only state results for the standard pebble game unless we state explicitly that the result also holds for the black-white pebble game.

1.3 Time-Space Tradeoffs and Graph Families

The time space tradeoffs of graph families have always been an area of interest in pebble games. The key question in studying such tradeoffs is given S pebbles and a graph $G_{n,\delta}$ with n nodes and indegree δ, how much time does it take to pebble $G_{n,\delta}$ using S pebbles. If S increased, does computing the graph take a shorter amount of time?

A number of previous results have proven time-space tradeoff bounds for various families of graphs. Paul and Tarjan [PT78] were the first to show an exponential time space trade for some constants c_1 and c_2 where $c_2 < c_1$ such that using $c_1\sqrt{n}$ pebbles requires n pebbling moves whereas

using $c_2\sqrt{n}$ pebbles requires $2^{\Theta(\sqrt{n})}$ moves. It subsequently became of research interest to find graph families that exhibit the greatest tradeoff between time and space when $\Theta(n/\log n)$ pebbles were used. This is of greater interest because $\Theta(n/\log n)$ pebbles is shown to be asymptotically tight for all DAGs [HPV77b]. In a monumental paper by Lengauer and Tarjan [LT82], it was shown that the tradeoff between time and space is exponential even when $S = \Theta(n/\log n)$ pebbles are used for a certain graph families consisting of graphs they called superconcentrators.

While much attention has been focused on the pebbling graphs with a large number of pebbles (such as $\Theta(n/\log n)$ pebbles), not much work has been done in terms of smaller number of pebbles (such as $\Theta(1)$ pebbles). The only line of research in this direction is a graph family that has a tradeoff of $T(S) = \Omega(n^2/S)$ where S is the number of pebbles used and $T(S)$ is the pebbling time as a function of S [LT82].

In this thesis, we introduce a graph family that has steep time-space tradeoffs when the number of pebbles used is $S < \sqrt{n}$, thereby lessening the gap in our knowledge with respect to graphs pebbled by $o(n/\log n)$ pebbles.

1.4 Parallel Pebbling Model, Previous Complexity Measures and Decremental Complexity

A series of recent papers in cryptography introduces the concept of the parallel Random Oracle Model (pROM) in the context of memory-hard functions [AB16a, ACK+16, AB16b, AGK+16, ABP16, AS15]. The concept behind the usage of the parallel pebbling model is that adversaries may break cryptographic memory-hard functions by computing such functions faster than an honest party. The idea behind such attacks is the design of special ASIC circuits that are specialized to break such functions while utilizing parallel time. The honest party, on the other hand, is assumed to only have access to a generic CPU that comes in any standard commercial computer or laptop.

In this section, we introduce the parallel pebbling model as well as some previous measures of complexity in the parallel pebbling model. We conclude with a brief description of decremental complexity which we define in more detail in Chapter 3.4.

Definition 1.1 (Parallel Pebbling Model [AS15])**.** *Let $G_{n,\delta} = (V, E)$ be a graph with n nodes and indegree δ and $T, S \subseteq V$ be sets of nodes. There is a legal sequence of pebble placements on node*

sets, $P = (P_0, \ldots, P_t)$, starting with pebbles on all nodes in S and ending with pebbles on all nodes in T that follow the following rules:

1. *$P_0 \subseteq S$.*

2. *Pebbles are added to a set of nodes P_i only if all of their predecessors are pebbled at time $i-1$:*

$$\forall i \in [t],\ \ \forall v \in P_i \setminus P_{i-1},\ \ \mathrm{pred}(v) \in P_{i-1}.$$

3. *Pebbles can be deleted from any set of nodes at any time.*

4. *At some point, every target node is pebbled (though not necessarily simultaneously):*

$$\forall x \in T,\ \ \exists z \leq t,\ \ x \in P_z.$$

Definition 1.2 (Parallel Complexity Measures)**.** *Let $G_{n,\delta}$ be a DAG with n vertices and δ indegree and let $P = (P_0, \ldots, P_t)$ be a pebbling of $G_{n,\delta}$ that takes t parallel pebbling steps. Let Π be the set of all complete parallel pebblings of G. Then, the cumulative cost of P, $c - cost(P)$, and the cumulative complexity (CC) of $G_{n,\delta}$, $cc(G_{n,\delta})$, are defined as follows:*

$$\text{c-cost}(P) \coloneqq \sum_{i=0}^{t} |P_i|$$

and

$$cc(G_{n,\delta}) \coloneqq \min \{\text{c-cost}(P) : P \in \Pi\}.$$

We further define the space complexity (SC), $\mathrm{sc}(G_{n,\delta})$, and the space/complexity complexity (STC), $\mathrm{stc}(G_{n,\delta})$, as the following:

$$\text{s-cost}(P) \coloneqq \max\{P_i : i \in \{0, \ldots, t\}\}$$
$$\text{et-cost}(P) \coloneqq k \cdot \max\{P_i : i \in \{0, \ldots, t\}\}$$
$$\text{sc}(G) \coloneqq \min\{\text{s-cost}(P) : P \in \Pi\}$$
$$\text{stc}(G) \coloneqq \min\{\text{st-cost}(P) : P \in \Pi\}.$$

A recent paper proved that it is NP-complete to compute the minimum cumulative complexity of a given DAG [BZ16] since the cumulative complexity of any DAG with n nodes cannot exceed n^2. However, it is conjectured but not proven in [BZ16] that approximating the cumulative complexity of a given graph is also hard. An efficient and reasonably accurate approximation algorithm would be useful in the sense that password hash functions could be analyzed as to the cumulative complexity hardness of computation of the hash function.

While cumulative complexity measures the amortized space-time cost of computing a function, allowing honest parties to trade space for time whereas adversaries must use less space but greater time, we believe an important measure in determining the memory-hardness of a function is the amount of pebbling time that is *gained* when switching from the parallel model of computation to the sequential model of computation. Thus, we introduce the concept of *decremental complexity* as the ratio of the minimum pebbling time in the sequential model over the minimum pebbling time in the parallel model given S (minimum pebbling) number of pebbles. Intuitively, this complexity measure tells us the relative disadvantage that an honest prover who does not have access to parallel computation must incur in terms of pebbling (computation) time. Graphs that have a decremental complexity ratio of 1 does not require the honest prover to use any more resources than the adversary–a very powerful statement to show. We show in Chapter 6 some examples of graphs that have low decremental complexity and pose as an open question: what is the maximum amount of space S you can show that results in a decremental complexity ratio of 1?

1.5 Organization of the Thesis and Summary of Contributions

Despite some results in the hardness of finding the minimum number of pebbles needed to pebble DAGs using the rules of the aforementioned pebble games, there are still a number of gaps in

our knowledge of these games. There are currently no results on whether the solutions could be approximated to within a constant multiplicative factor of the optimal or whether a fixed-parameter tractable algorithm can be found for these games. In this thesis, we discuss the following new results:

1. We present some hardness results in approximating the minimum number of pebbles in the standard pebble game, in particular, improving upon a result presented in [CLNV15].

2. We show that it is W[1]-hard in the number of transitions to find the minimum number of red pebbles and transitions needed in the red-blue pebble games.

3. We also introduce a new red-blue pebble game paradigm that we prove to be NP-complete to solve for the minimum number of red pebbles and transitions. Recently, the one-shot variation of the red-blue pebble game (i.e. without recomputation of values in cache) has been shown to be approximable to a polylogarithmic factor (same as the best known approximation factor for the one-shot standard pebbling game [APW12]) and can be shown to be applicable to modeling multi-level memory hierarchies [CRSS16]. Thus, it seems natural to consider the case when recomputation is allowed but some version of every computed value is kept in memory.

4. We provide a family of DAGs that exhibit pebbling time $\Omega(n^k)$ time given constant k pebbles in the black pebbling case, thereby partially resolving a longstanding open problem as can be seen in [Nor15]. We conjecture that this graph family also exhibits this pebbling time lower bound in the black-white pebbling model.

5. We also make some observations in the complexity of approximating other pebble games such as the one-shot standard pebble game by utilizing a variety of previous results available in the literature.

6. We finally present our decremental complexity definition and analyze this complexity measure on common graph families.

The organization of the thesis is as follows. In Chapter 3, we present a short argument on the complexity of finding the minimum number of red pebbles and transitions needed to pebble a given DAG. We also introduce our new paradigm for the red-blue pebble game without deletion and show that this game is NP-complete. In Chapter 4, we prove that the red-blue pebble game is

W[1]-hard to determine whether a given number of red pebbles and parameterized by the number of transitions, k. In Chapter 5, we present our inapproximability result for the standard pebble game to an additive factor of $n^{1-\varepsilon}$ for any ε. In Chapter 6, we present our decremental complexity measure and analysis of several common graph families with regard to this measure. In Chapter 7, we talk about other results in pebbling including other hardness results obtained through some scouring of the current literature. Finally, in Chapter 8, we discuss some open problems resulting from this thesis.

Chapter 2

Definitions and Terminology

In this section, we define some definitions and terminology that we will use throughout the thesis. Since graph pebbling by definition only occurs on directed acyclic graphs, we define $\mathbb{G}_{n,\delta}$ to be the family of all DAGs with n vertices and indegree δ. (The *indegree* of a DAG is the maximum number of incoming edges to any node in the DAG.)

Let $G_{n,\delta} = (V, E) \in \mathbb{G}_{n,\delta}$ be a DAG with n nodes and indegree δ. A pebbling of $\mathbb{G}$ is a sequence of pebble configurations $\mathbb{P} = \{P_0, \ldots, P_t\}$ such that $P_0 = \emptyset$ and $P_t = S$ where S is the set of *sinks* (i.e. nodes with no outgoing edges). We define $[i]$ to be the ordered set of values $[0, \ldots, i-1]$. Then, for all $i \in [t]$, P_{i+1} follows from P_i by the rules of the pebble games given in Chapter 1.

For any given node $v \in G_{n,\delta}$, we define $\mathsf{succ}(v)$ to be the set of all successors of v and $\mathsf{pred}(v)$ to be the set of all predecessors of v. Furthermore, let $G_v^{\triangledown}$ be the set of all ancestors of v and $G_{\triangle}^v$ be the set of all descendants of v. Here we define a *predecessor* of node v to be any node u that has a directed edge directed into v (i.e. $(u, v) \in E$). A *successor* of v is any node w that has a directed edge from v into w. The *ancestors* of v are all nodes u' where there exists a directed path from u to v and the *descendants* of v are all nodes w' where there exists a directed path from v to w. Let $G_{\triangle}^{\not v}$ and $G_{\not v}^{\triangledown}$ be the corresponding sets not including v. We define nodes u and w to be siblings if $u, w \in \mathsf{pred}(v)$. The *sources* of $G_{n,\delta}$ are all nodes $s \in S(G_{n,\delta})$ that have indegree 0 and the *sinks* are all nodes $z \in Z(G_{n,\delta})$ that have outdegree 0.

The *depth* of $G_{n,\delta}$ is the longest path from a source in $S(G_{n,\delta})$ to a sink in $Z(G_{n,\delta})$.

2.1 Sequential Pebbling Definitions

Given the terminology we defined in the previous section we now define the measures by which we determine the pebbling cost of a sequential pebbling strategy. First we define a sequential pebbling strategy, $\mathbb{P}$, to be the following:

Definition 2.1. *A sequential pebbling strategy $\mathbb{P} = \{P_0, \ldots, P_t\}$ is a set of valid pebbling configurations (that follow the rules defined in Chapter 1) where we let $|P_i|$ be the number of pebbles used in configuration P_i. Then, one of the following must be true:*

1. $|P_i| = |P_{i-1}| + 1$,
2. $|P_i| = |P_{i-1}| - 1$, *or*
3. $|P_i| = |P_{i-1}|$.

Let $\mathsf{Peb}(G_{n,\delta})$ be the sequential pebbling price of pebbling $G_{n,\delta}$ where $\mathsf{Peb}(G_{n,\delta})$ is defined as follows:

Definition 2.2. *The sequential black pebbling cost of pebbling $G_{n,\delta}$ by strategy $\mathbb{P}$ is defined as the maximum number of pebbles used, where $|P_i|$ is the number of pebbles used in configuration P_i:*

$$\mathsf{Peb}(G_{n,\delta}, \mathbb{P}) = \max_{P_i \in \mathbb{P}}(|P_i|).$$

We define $\mathsf{Peb}(G_{n,\delta})$ to be the minimum number of pebbles used by all strategies $\mathbb{P} \in \mathcal{P}$:

Definition 2.3.

$$\mathsf{Peb}(G_{n,\delta}) = \min_{\mathbb{P} \in \mathcal{P}}(\mathsf{Peb}(G_{n,\delta}, \mathbb{P})).$$

Furthermore, we define the function, $\mathsf{Time}(G_{n,\delta}, S)$ to be the sequential time of pebbling $G_{n,\delta}$ using S pebbles:

Definition 2.4. *Let t be the number of sequential steps to pebble graph $G_{n,\delta}$ using strategy $\mathbb{P} = \{P_0, \ldots, P_t\}$. And let $\mathsf{Time}(G_{n,\delta}, \mathbb{P}) = t$. Then,*

$$\mathsf{Time}(G_{n,\delta}) = \min_{\mathbb{P}\in\mathcal{P}}(\mathsf{Time}(G_{n,\delta},\mathbb{P})).$$

2.2 Parallel Pebbling Definitions

We now define the parallel pebbling cost explored in [AS15, CRSS16] and further results will be mentioned in Chapter 6. We first define a parallel pebbling strategy, $\mathbb{P}^{||}$ to be the following:

Definition 2.5. *A parallel pebbling strategy* $\mathbb{P}^{||} = \left\{P_0^{||}, \ldots, P_t^{||}\right\}$ *is a set of valid pebbling configurations (that follow the rules defined in Chapter 1) where we let* $|P_i^{||}|$ *(the number of pebbles used in configuration* $P_i^{||}$*) be any size.*

Furthermore, we define the parallel pebbling cost of a parallel pebbling strategy, $\mathbb{P}^{||}$ to be the following:

Definition 2.6. *The parallel black pebbling cost of pebbling* $G_{n,\delta}$ *by strategy* $\mathbb{P}^{||}$ *is defined as the maximum number of pebbles used, where* $|P_i^{||}|$ *is the number of pebbles used in configuration* $P_i^{||}$*:*

$$\mathsf{Peb}^{||}(G_{n,\delta},\mathbb{P}^{||}) = \max_{P_i^{||}\in\mathbb{P}^{||}}(|P_i^{||}|).$$

Then, the parallel pebbling cost is the following:

Definition 2.7.

$$\mathsf{Peb}^{||}(G_{n,\delta}) = \min_{\mathbb{P}^{||}\in\mathcal{P}^{||}}(\mathsf{Peb}^{||}(G_{n,\delta},\mathbb{P}^{||})).$$

Finally, we define the time cost of parallel pebbling $G_{n,\delta}$ to be:

Definition 2.8. *Let* $t^{||}$ *be the number of parallel steps to pebble graph* $G_{n,\delta}$ *using strategy* $\mathbb{P}^{||} = \left\{P_0^{||}, \ldots, P_t^{||}\right\}$*. And let* $\mathsf{Time}^{||}(G_{n,\delta},\mathbb{P}^{||}) = t^{||}$*. Then,*

$$\mathsf{Time}^{||}(G_{n,\delta},S) = \min_{\mathbb{P}^{||}\in\mathcal{P}^{||}}(\mathsf{Time}^{||}(G_{n,\delta},\mathbb{P}^{||})).$$

In addition to the previously mentioned parallel definitions, we also introduce the following definition for measuring the relative robustness of a graph in resisting decrease in time when switching from a sequential strategy to a parallel pebbling strategy. As we will mention in Chapter 6, this measure has important applications in cryptography such as proofs of work and space and memory-hard functions.

We define $\mathsf{Dec}(G_{n,\delta}, S)$ to be the ratio of the time cost of sequential pebbling and the time cost of parallel pebbling of $G_{n,\delta}$ using S pebbles.

Definition 2.9.

$$\mathsf{Dec}(G_{n,\delta}, S, \mathbb{P}, \mathbb{P}^{||}) = \frac{\mathsf{Time}(G_{n,\delta})}{\mathsf{Time}^{||}(G_{n,\delta})}$$

where $\mathsf{Peb}(G_{n,\delta}, P) \leq S$ *and* $\mathsf{Peb}^{||}(G_{n,\delta}, \mathbb{P}^{||}) \leq S$.

We define $\mathsf{Dec}(G_{n,\delta}, S)$ to be the minimum ratio between the time of pebbling sequentially and the time of pebbling in parallel. We define this complexity measure as the *decremental complexity* of the graph.

Definition 2.10.

$$\mathsf{Dec}(G_{n,\delta}, S) = \min_{\mathbb{P} \in \mathcal{P}_S, \mathbb{P}^{||} \in \mathcal{P}_S^{||}} \frac{\mathsf{Time}(G_{n,\delta})}{\mathsf{Time}^{||}(G_{n,\delta})}$$

where $\mathcal{P}_S^{||}$ *and* $\mathcal{P}_S$ *are the set of strategies in their respective models that uses at most* S *pebbles.*

Note that $\mathsf{Dec}(G_{n,\delta}, S) \geq 1$ by simple argument.

Chapter 3

Red-Blue No-Deletion Pebble Game is NP-Complete

We begin this section with a short proof that the red-blue pebble game *with deletion* is PSPACE-complete. We do not expand too much into the proof since it relies heavily on the proof given in [GLT79] (and is almost identical to the proof provided there). Therefore we include this result before our main result on red-blue pebbling *without deletions* which is the main proof we expand upon in detail.

First, we define the *red-blue start-in-disk* game to be the version of the red-blue pebble game as defined in Section 1.2.2 (i.e. all source nodes contain blue pebbles at the beginning of the computation, i.e. at $t = 0$, and all inputs if deleted from cache must be redrawn from disk) and the *red-blue start-in-cache* game to be the version of the red-blue pebble game where we remove the condition that all source nodes contain blue pebbles at the beginning of the computation (i.e. essentially, all inputs start in cache) and red pebbles can be placed at any time on the source nodes–without the need of blue pebbles being on the source nodes first (i.e. inputs always stay in cache).

Before we dive into the proofs, we first show that any red-blue pebbling of a DAG G using the rules of the red-blue start-in-cache game that has minimum pebbling space usage r and number of transitions k can be converted to a DAG G' with minimum pebbling space usage $r+1$ and number of transitions $k+1$ using the rules of the red-blue start-in-disk game.[1]

[1]Note, we did not explicitly make this distinction between the red-blue start-in-cache and the red-blue start-in-disk

Proposition 3.1 (Red-Blue Disk to Cache)**.** *Given a DAG, $G = (V, E)$, with bounded indegree 2 that uses a minimum of r red pebbles and k transitions to pebble using the rules of the red-blue start-in-cache game, we can convert it into a DAG, $G' = (V', E')$, that uses a minimum of $r + 1$ red pebbles and $k + 1$ transitions to pebble using the rules of the red-blue start-in-disk game.*

Proof. First, create a node u and let $V' = V \cup \{u\}$. Then, create a set of directed edges U where we add an edge (u, v) to U for all $v \in V$. Let $E' = E \cup U$. Graph G' now potentially has vertices with indegree up to 3. In the final step of creating G', we replace all vertices with indegree 3 with pyramids of height 3. Note that a pyramid of height 3 functions in the same manner as a node with indegree 3 by normality of pebbling strategies proven in [GLT79, Nor15].

Let $\mathbb{P}$ be the optimal strategy used to pebble G using the rules of the red-blue start-in-cache game that results in a minimum of r red pebbles and k transitions.

Now, we prove that a minimum of $r + 1$ pebbles and $k + 1$ transitions are necessary to pebble G' using the rules of the red-blue start-in-disk game. By construction, G' has one source (leaf) node where one blue pebble is placed on it at the beginning of the pebbling (at $t = 0$). Before any other pebbles can be placed on G', we must use exactly 1 transition to turn the blue pebble on the source to a red pebble. The red pebble remains on the source, and we use strategy $\mathbb{P}$ to pebble the remaining nodes of G'.

We now show that in order to use a minimum of $k+1$ transitions, the red pebble must remain on the source during the entire computation of G'. Suppose for contradiction, the red pebble is turned into a blue pebble (recall that all leaves must contain a pebble at all times–otherwise they can never be pebbled again using the rules of the red-blue start-in-disk game), then, in any future pebbling of any other nodes in G', the blue pebble on the source must be turned into a red pebble, resulting in 2 additional transitions (a total of $k + 3$ transitions) which exceeds the minimum allowed $k + 1$ transitions (since $\mathbb{P}$ uses a minimum of k transitions). Thus, given that the red pebble remains on the source during the entire pebbling of G' (i.e. the red pebble on the source is present at time $t = \max$ where the maximum number of red pebbles are on the graph using strategy $\mathbb{P}$) the number of red pebbles necessary to pebble G is then increased by 1, so a minimum of $r + 1$ red pebbles are necessary to pebble G' given that a minimum of $k + 1$ transitions are used. □

In the remaining sections of the paper, we prove all results with respect to the rules of the

models in the first version of this thesis.

red-blue start-in-cache game, even if we do not explicitly state that we do so. Note that using Proposition 3.1, we can transform any graph G we use in our hardness reductions into a graph G' that can be used to show the corresponding hardness results for the red-blue start-in-disk game.

3.1 Red-Blue Pebble Game is PSPACE-complete

The red-blue pebble game as defined in Section 1.2.2 is PSPACE-hard as a simple extension of the proof given in [GLT79]. The formal definition of the problem is given below.

Definition 3.1 (Red-Blue Pebble Game)**.** *Given a DAG, $G(V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, find a pebbling of G following the red-blue pebbling rules provided in Section 1.2.2 such that at most r red pebbles are present on G at any time and the number of red-blue transitions, k, is minimized.*

The proof structure and the gadgets to show that the red-blue pebble game is PSPACE-hard can be constructed in the same way as the gadgets in the proof of the PSPACE-hardness of the standard pebble game as defined in [GLT79]. The reduction would specify the number of red pebbles necessary to be one greater than the number of pebbles necessary in the proof presented by Gilbert et al. [GLT79] and the number of transitions to be 0. We, thus, only need to show that the number of red pebbles necessary to pebble the gadgets in the construction is indeed one greater than the number necessary to pebble the construction provided in [GLT79]. If the construction can be pebbled with one greater pebble in the red-blue pebble game using 0 transitions if and only if the construction in [GLT79] can be pebbled using the rules of the standard pebble game, then we have shown that the red-blue pebble game is PSPACE-complete.

Lemma 3.1. *The proof construction provided in [GLT79] can be pebbled using s pebbles in the standard pebble game if and only if it can be pebbled using $s+1$ red pebbles and 0 transitions in the red-blue pebble game.*

Proof. We first show that if the construction given in [GLT79] can be pebbled using s pebbles in the standard pebble game, then it can be pebbled using $s + 1$ red pebbles and 0 transitions in the red-blue pebble game. The only difference between the rules of the standard pebble game and the red-blue pebble game is that in the standard pebble game, a pebble can be moved from a node to

a successor from a predecessor. However, in the red-blue pebble game, we are no longer allowed moving a pebble from a predecessor to a successor. However, the process of moving a pebble from predecessor to successor can be modeled using 2 red pebbles. Suppose that a pebble is moved from a predecessor to a successor in the standard pebble game. Let v be node in the graph and p be the predecessor from which a black pebble was moved to v. Let $pred(v)$ indicate the set of nodes that are the predecessors of v. Since a pebble was moved from p to v, it means that all nodes in $pred(v)$ are pebbled with black pebbles. If the pebble movement from p to v is modeled using red pebbles, then at the time the black pebble was moved from p to v, all nodes in $pred(v)$ would be pebbled with red pebbles. Thus, one additional red pebble can be placed on v at the same time. In the construction provided by [GLT79], if the constructed DAG can be pebbled using s pebbles, then one additional pebble will be able to simulate all pebble movements from predecessor nodes to successor nodes. Thus, if the original construction can be pebbled using s pebbles in the standard pebbling game, the construction can be pebbled using $s+1$ pebbles in the red-blue pebbling game.

Now we show that if the construction provided in [GLT79] can be be pebbled using $s+1$ red pebbles and 0 transitions in the red-blue pebble game, then it can be pebbled using s black pebbles in the standard pebble game. We need to show that having one additional pebble in the red-blue pebble game does not provide an advantage in any situations where a pebble is moved from a predecessor to a successor in the standard pebble game. Suppose for the purposes of contradiction that there is an advantage if an extra pebble is provided in the red-blue pebble game for situations where a pebble is moved from a predecessor to successor node in the standard pebble game. Then, some node can be pebbled in the red-blue pebble game with one extra pebble that cannot be pebbled in the standard pebble game. Suppose that node v cannot be pebbled in the standard pebble game. This means that some node in $pred(v)$ is not pebbled. Otherwise, a pebble can be moved from a node in $pred(v)$ to v. With one extra pebble, the node in $pred(v)$ can be pebbled. However, in the red-blue pebble game, no red pebbles can be moved from a node in $pred(v)$ to v. Therefore, v cannot be pebbled even with one extra pebble, a contradiction.

In the proof construction provided in [GLT79], every pebbling of a node requires moving a pebble from a predecessor to a successor *except* pebbling the leaf nodes of the clause gadgets. Thus, having an extra pebble in all cases but the cause of pebbling the leaf nodes of the clause gadgets does not provide an advantage. Now, we will show that having an extra pebble in the red-

blue pebble game also does not provide an advantage in this case. In the proof provided by [GLT79], the clause gadgets can be pebbled using 3 additional pebbles. Suppose that in the red-blue pebble game construction, the clauses now have 4 additional pebbles available. In this case, the red-blue pebble game only provides an advantage if 4 pebbles can be used to pebble a clause gadget even if all variable gadgets connecting to it are in the false configuration. In this case, 4 pebbles must be used to pebble all the leaves of the clause gadget. Then, since the non-leaf nodes of the clause gadget requires a pebble to be moved from a predecessor to successor, they cannot be pebbled in the red-blue pebble game using 4 pebbles if all literals connecting to the clause are false. A simple case by case analysis shows that no other strategy can cause the clause gadget to be completely pebbled. □

Theorem 3.1. *Determining the minimum pebbling cost and number of transitions is PSPACE-complete to compute in the red-blue pebbling game.*

Proof. Containment in PSPACE is trivial and PSPACE-hardness of the red-blue pebble game follows immediately from Lemma 3.1. □

3.2 Red-Blue Pebble Game with No Deletion

In this section, we introduce our model of the red-blue pebble game with *no deletion* and prove that it is NP-complete to determine the minimum number of red pebbles and transitions needed to pebble a given DAG under the rules of this game. The red-blue pebble game with no deletion is defined as follows:

1. A red pebble can be placed on any vertex that has a blue pebble. (Transition move.)
2. A blue pebble can be placed on any vertex that has a red pebble. (Transition move.)
3. A red pebble can be placed on a vertex where all predecessors of the vertex contain red pebbles. (The red pebble can override preexisting pebble placements, i.e. the vertex already contains a blue pebble.)
4. No pebbles can be deleted from a vertex.

Red pebbles represent fast memory and blue pebbles represent slow memory. As usual, we assume that we have infinitely large slow memory, but only a bounded fast memory. The goal of the game is to pebble all vertices in G while minimizing the number of *transition* moves. Transition moves are moves that convert a pebble of one color into a pebble of another color. The motivation of this game is to determine the added computational complexity of allowing deletions to occur in the RAM. Suppose that one would like to limit the number of deletions or to minimize the number of transitions as well as deletions. Another motivation is to always maintain computed data in memory. For example, for certain persistent data structures, one always want to keep some form of computed values in memory at all times. This paper analyzes the computational complexity of such a model.

The formal statement of the game is almost identical to the definition of the red-blue pebble game and is the following:

Definition 3.2 (Red-Blue Pebble Game with No Deletions)**.** *Given a DAG, $G(V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, find a pebbling of G following the red-blue with no deletion pebbling rules (given above) such that at most r red pebbles are present on G at any time and the number of red-blue transitions, k, is minimized.*

In the next few sections, we show that the Red-Blue Pebble Game is NP-complete.

3.3 Proof Overview

We provide a reduction broadly similar in concept to [GLT79] except we reduce from Positive 1-in-3 SAT to show that our problem is NP-complete. The definition of Positive 1-in-3 SAT is given below:

Definition 3.3 (Positive 1-in-3 SAT [GJ90])**.** *Given a set U of variables and a collection C of clauses over U such that each clause $c \in C$ has size $|c| = 3$ and all literals in c are positive, does there exist a truth assignment for U such that each clause has exactly one true literal?*

The proof of NP-completeness of the red-blue pebble game with no deletions proceeds as follows. We create a set of variable gadgets that are pebbled with a set of red pebbles that determine whether the variable is set to true or false. The variable gadgets are then connected to clause and anti-clause gadgets that enforce the 1-in-3 condition on the variable settings. The variable gadgets

are also connected to a *pebble sink path* that ensures that all variables are pebbled and set to a truth configuration before the clause gadgets are pebbled. Finally, the clause gadgets and variable gadgets are connected to a *pebble hold path* that ensures that all red pebbles are removed from these gadgets and are used to fill up the pebble hold path. Specifics about the gadgets and details of the proof construction will be given in the next few sections of this thesis.

3.4 Gadgets

In this section, we introduce some gadget components that will be used in the proof that the red-blue pebble game with no deletions is NP-complete.

We define a *variable* gadget for every $x_i \in U$. The purpose of the variable gadget is to force a selection of variable assignments. In order to construct the variable gadget, we use a *pyramid* gadget introduced by previous work [GLT79] that is used to "trap" a certain minimum number of pebbles that must be used to pebbled the gadget. Henceforth, for every gadget, g, we introduce, we will specify the minimum number of red pebbles, r_g, that can remain on the gadget after it has been pebbled once and t_g, the minimum number of red-blue transitions that must be performed on the gadget after it is pebbled each time.

The pyramid graph has been proven to use h pebbles where h is the height (where a single node has height 1) of the pyramid graph using standard pebbling (with sliding pebbles) [Coo73]. Let Π_h be a pyramid graph with height h. It was proven in [Nor15] that the standard pebbling price with no sliding is $h+1$ for a pyramid with height h. Here we prove that using the red-blue pebbling strategy with no deletions, the minimum pebbling price of a pyramid with height h is $r_{\Pi_h} = h+1$ and $t_{\Pi_h} = \frac{h(h+1)}{2}$. Let $PebRBD(\Pi_h)$ be the minimum pebbling price of a pyramid graph using the red-blue strategy with no deletions. The ending state of the pyramid has no red pebbles. We use this property of the pyramid graph in our proof in Section 3.5.

Lemma 3.2. *Given a pyramid graph of height h, the $PebRBD(\Pi_h)$ is $r_{\Pi_h} = h+1$ and $t_{\Pi_h} = \frac{h(h+1)}{2}$.*

Proof. The standard pebbling lower bound (no sliding) for pyramids is h given a pyramid of height h. The number of red pebbles necessary to pebble a pyramid of height h, however, is $h+1$ since the key component of the bound on standard pebbling of pyramids relies on the ability to slide pebbles, whereas in our pebbling game, no pebble slides are allowed. Otherwise, the rules for red

pebble placement is the same as the rule for standard pebbling with no sliding. Therefore, by the proof of Theorem 4.8 in [Nor15], the lower bound for the number of red pebbles necessary to pebble Π_h is $r_{\Pi_h} = h + 1$.

As stated in the proof of Theorem 4.8 in [Nor15], the strategy for achieving this pebbling is to pebble the bottom row (the sources) of the pyramid and use one extra pebble to move the pebbles up the levels of the pyramid.

To prove the transitions bound, since the only way to remove all the pebbles from the pyramid is to use transitions, the number of transitions is $\geq \frac{h(h+1)}{2}$. Since the pebbling strategy stated above for pyramids is linear (each vertex is pebbled by a red pebble at most once), each node is visited once and a red pebble is removed from each node one resulting in $t_{\Pi_h} = \frac{h(h+1)}{2}$. When a pebble is "moved up" a level of the pyramid, the pebble on the previous node is turned to blue. □

Fig. 3-1 shows the construction of the pyramid gadget and its associated symbol that will be used to denote it in all subsequent proofs in Section 3.5. One can see that the number of pebbles required to fill the pyramid gadget is also the number of leaves on the bottom layer plus one and the number of transitions is $\sum_{i=1}^{l} i$ where l is the number of leaves in the gadget.

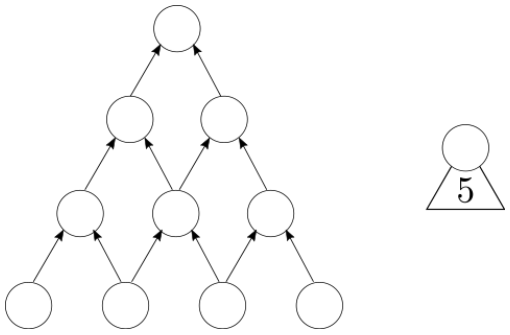

Figure 3-1: Example of a pyramid gadget with $r_{\Pi_4} = 5$ and $t_{\Pi_4} = 10$.

Using the pyramid gadget we can construct the variable gadget as the following:

Lemma 3.3. *The pebbling price of the variable gadget (not including s_i, s'_i, s''_i, g_i, g'_i, g''_i or q_i) for variable x_i is $r_{x_i} = a_i$ and $t_{x_i} = \sum_{j=1}^{3} \frac{(a_i-j+1)(a_i-j)}{2} + 4$. The transitions cost includes the cost of pebbling and removing red pebbles from all nodes as shown in Fig. 3-2 except s_i, s'_i, s''_i, g_i, g'_i, g''_i and q_i. During Phase 1, the pebbling cost is $r_{x_i} = a_i$ and $\sum_{j=1}^{3} \frac{(a_i-j+1)(a_i-j)}{2} - 3 \leq t \leq \sum_{j=1}^{3} \frac{(a_i-j+1)(a_i-j)}{2} - 1$. Furthermore, during Phase 2 of the pebbling, the pebbling cost is $r_{x_i} = 4$ and $5 \leq t_{x_i} \leq 7$.*

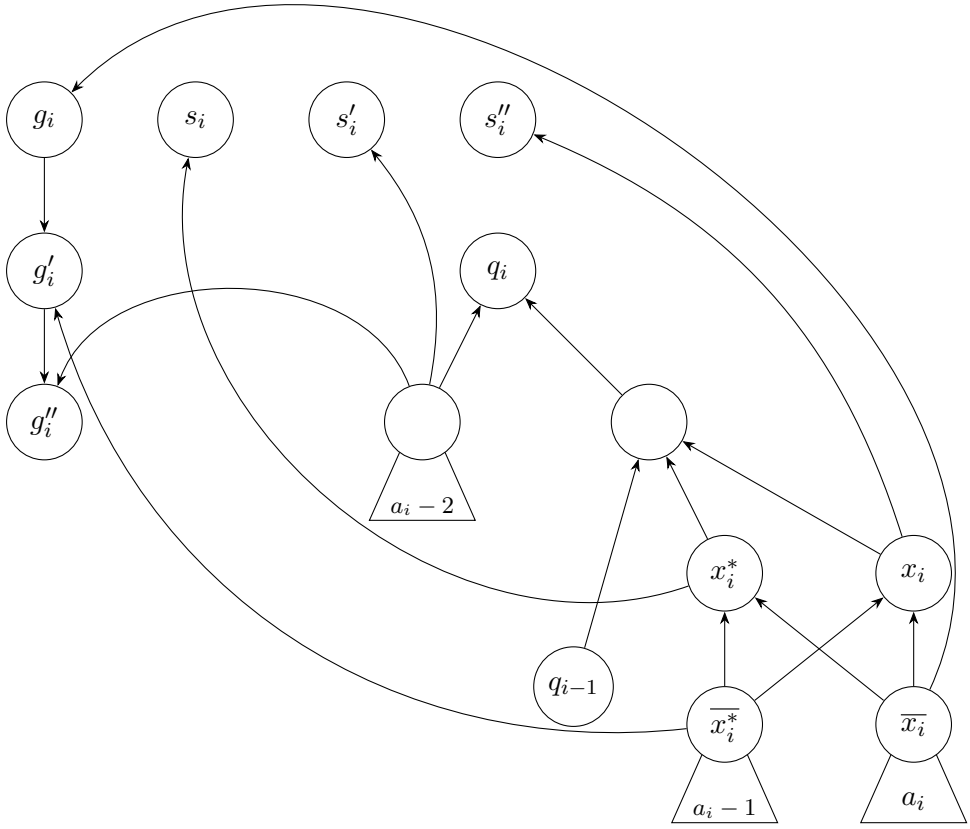

Figure 3-2: Example of a variable gadget, x_i, with pyramid costs a_i, pebble sink path connections g_i, g'_i, and g''_i. The corresponding pebble sinks that correspond with this gadget are s_i, s'_i, and s''_i.

Proof. Each of the three pyramids must be pebbled using the number of pebbles shown in Fig. 3-2 by Lemma 3.2. The number of nodes in the gadget is $\sum_{j=1}^{3} \frac{(a_i-j+1)(a_i-j)}{2} + 4$ (excluding s_i, s'_i, s''_i, g_i, g'_i, g''_i and q_i). If we are limited to a total of $t_{x_i} = \sum_{j=1}^{3} \frac{(a_i-j+1)(a_i-j)}{2} + 4$ transitions in both Phase 1 and Phase 2, then each vertex of the gadget can only be pebbled once among the two phases. In Phase 1, we must use a_i red pebbles to pebble all three pyramids. Recall that g_i, g'_i, and g''_i must be pebbled at the end of Phase 1 in order to begin pebbling the clauses. In order to pebble g_i, g'_i, and g''_i, all the pyramids in all variable gadgets x_i must be pebbled by the end of Phase 1.

In addition to pebbling the pyramids, one pair of x_i^* and x_i or $\overline{x_i^*}$ and $\overline{x_i}$ must be pebbled in Phase 1 in order to be able to pebble them during the clause verification phase. To ensure that each vertex in the gadget is pebbled once during Phase 1 and Phase 2, the variables must be selected in the indicated pairs. Suppose for contradiction that x_i and $\overline{x_i^*}$ are selected to contain a pebble. This means that $\overline{x_i}$ must have been pebbled at some point in Phase 1 (and a transition was used to turn the pebble to blue). Now, in Phase 2, $\overline{x_i}$ must be pebbled again (and using one more transition to turn the pebble to blue) to pebble x_i^*, breaking our invariant. The same holds if the pair x_i^* and $\overline{x_i}$ was chosen initially.

In Phase 2, the following set of pebblings must occur. If $\overline{x_i}$ and $\overline{x_i^*}$ have red pebbles remaining from Phase 1, then x_i^* and x_i must be pebbled in Phase 2. Since, q_{i-1} will be pebbled by the end of the clause verification phase, the remaining parts of the gadgets can be pebbled using in 7 more transitions. If, instead, x_i and x_i^* have red pebbles remaining from Phase 1, then the remaining nodes of the gadget can be pebbled using 5 more transitions. Since s_i, s_i', s_i'', and q_i must be pebbled with red pebbles at the end of pebbling x_i in Phase 2, the number of pebbles necessary to pebble x_i in Phase 2 is 4. □

We define a_i for all x_i shortly. Each x_i variable gadget requires a_i pebbles since in order to choose an assignment for x_i, a_i pebbles must be used to pebble the pyramid gadget attached to the variable. q_{i+1} is part of the next variable gadget. The gadget must be pebbled in the following way.

During Phase 1 of pebbling the variable gadget, x_i, each pyramid gadget is pebbled. Then, the corresponding nodes in the pebble sink path can be pebbled in the order g_i, g_i', and, then, g_i''. All nodes along the pebble sink path must be pebbled in order to pebble the clauses and proceed to the clause verification phase. First, a_i pebbles must be used to pebble both $\overline{x_i}$ and $\overline{x_i^*}$ with red pebbles, converting all other pebbled vertices in each pyramid to contain blue pebbles. This then leaves $a_i - 2$ pebbles to pebble the other pyramid gadget, leaving one pebble at the apex of the gadget and converting all other pebbled vertices in the pyramid to contain blue pebbles. Then, either x_i and x_i^* are pebbled with red pebbles and $\overline{x_i}$ and $\overline{x_i^*}$ are converted to blue pebbles or $\overline{x_i}$ and $\overline{x_i^*}$ contain red pebbles and x_i and x_i^* are not pebbled. At most 3 red pebbles can remain on each variable gadget.

In Phase 2, q_{i-1} will be pebbled once all clauses are pebbled. Therefore, all other nodes of the

variable gadget must be pebbled using the pebbles that remain on each pyramid gadget. If the red pebbles from Phase 1 are placed on $\overline{x_i^*}$ and $\overline{x_i}$, then x_i and x_i^* need to be pebbled in Phase 2. Furthermore, s_i, s_i' and s_i'' need to be pebbled with red pebbles in Phase 2 by moving the red pebbles from each corresponding pyramid. The red pebbles will remain on these nodes since no transitions are allowed to be spent on these nodes.

Lemma 3.4. *If n' variables does not have at least one of the two pairs, x_i^* and x_i or $\overline{x_i}$ and $\overline{x_i^*}$, pebbled at the end of the clause verification phase, then a total of at least $\left(\sum_{i=1}^{n}\sum_{j=1}^{3}\frac{(a_i-j+1)(a_i-j)}{2}+4\right)+n'$ transitions are needed to pebble all variable gadgets in Phase 1 and Phase 2.*

Proof. Suppose that without loss of generality, node $\overline{x_i}$ is pebbled and node $\overline{x_i^*}$ is not at time t in Phase 2. Suppose also that in accordance with the lemma, either node x_i^* or x_i is also not pebbled with a red pebble. Since all pyramids must be pebbled at the end of Phase 1, $\overline{x_i^*}$ must have been pebbled with a red pebble at some time $t' < t$ during Phase 1. Since $\overline{x_i^*}$ was pebbled with a red pebble at time t' and holds a blue pebble at time t in Phase 2, 1 transition must have been used to convert the red pebble to a blue pebble during Phase 1 (since $a_n - 3n$ pebbles are needed to pebble the top of the pebble sink path). In order to pebble either x_i^* or x_i in Phase 2, one transition must be used to convert the blue pebble on $\overline{x_i^*}$ to a red pebble. Therefore, one additional transition per each of the n' variables. Therefore, the number of transitions necessary to pebble the variable gadgets in Phase 1 and Phase 2 is $\left(\sum_{i=1}^{n}\sum_{j=1}^{3}\frac{(a_i-j+1)(a_i-j)}{2}+4\right)+n'$.

□

Each variable gadget is connected to a *pebble sink path* as shown in Fig. 3-3. The purpose of the pebble sink path is to ensure that all pyramids are pebbled by the end of Phase 1 of variable pebbling and that each variable only contains at most 3 red pebbles. The pebble sink path can be pebbled using $a_n - 3n$ red pebbles and the number of transitions needed to pebble this path is $3n + \frac{(a_n-3n+1)(a_n-3n)}{2}$. The number of transitions indicate that each node of the pebble sink path can only be pebbled once. Once the end of the pebble sink path is pebbled, the clause gadgets can be pebbled in the *Clause Verification* phase.

A *clause gadget* is created for each $c_j \in C$. The clause gadget is connected to every positive x_i literal that is present in its respective clause c_j. An example clause gadget with $r_{c_i} = 6$ and $t_{c_i} = 29$ (here r_{c_i} *does not* include the red pebble on the one true literal and the red pebble on p_{i-1} and t_{c_i} does not include the transition used to turn the pebble on p_i to blue) is shown below.

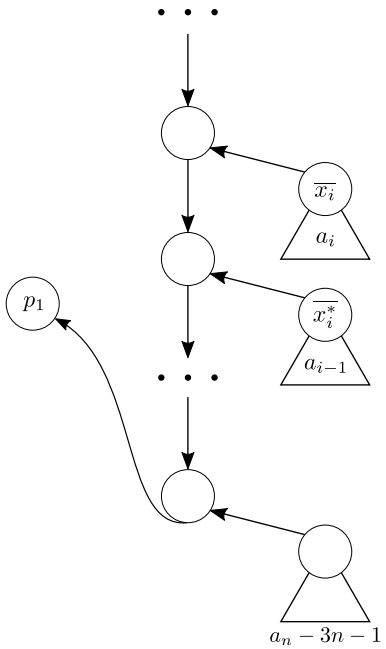

Figure 3-3: Example pebble sink path. Each node is connected to the root of each pyramid in each variable gadget.

The clause gadget must be pebbled by first pebbling the pyramid gadget that requires 6 red pebbles to pebble. Then, with the remaining 5 red pebbles the bottom layer of the gadget (i.e. the nodes representing the literals) are pebbled using 4 red pebbles plus the two pebbles that are present on the true positive literal (false literals are pebbled to become true). The final red pebble plus the red pebble from p_{i-1} are used to pebble the next layer. The rest of the pebbling follows directly from this initial pebbling. Once the gadget has been pebbled, one red pebble is left at the apex of the gadget, p_i, and the true literal still contains its red pebbles. Four transitions are used to convert the two pebbled false literals back to false and 25 transitions are used to convert all other vertices except p_i and the positive literal to blue.

Lemma 3.5. *Given $r_{c_i} = 6$ and $t_{c_i} = 29$, the clause gadget c_i cannot be pebbled if all three variable gadgets incident on c_i are in the false configuration.*

Proof. The amount of transitions needed to pebble the clause gadget if all incident literals are in the false position is 31 if no red pebbles remain on the gadget after it is pebbled. Suppose that two red pebbles remain on the gadget, then the number of red pebbles available to the next clause is 4 which is not enough to ensure that the clause is successfully pebbled. Suppose without loss of

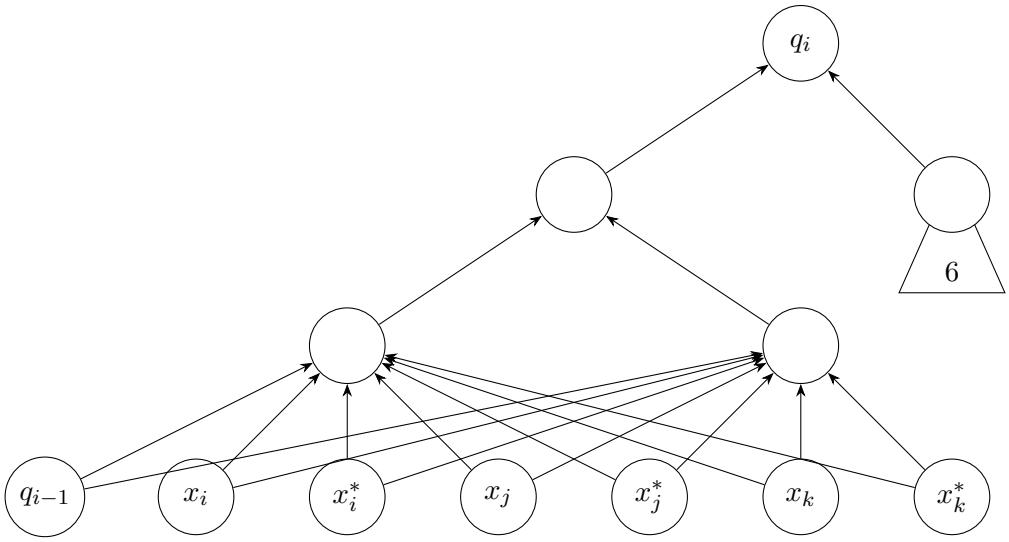

Figure 3-4: Example of a clause gadget with $r_{c_i} = 2$ and $t_{c_i} = 8$ for clause $c_i = (x_i \vee x_j \vee x_k)$. The number of red pebbles that is needed to fill this gadget is 6 (excluding the two red pebbles that are present on the true literal and the red pebble on p_{i-1}).

generality that two red pebbles remain on x_i and x_i^*, then, two red pebbles need to be removed from $\overline{x_i}$ and $\overline{x_i^*}$ resulting in two extra transitions as before. Therefore, a clause gadget cannot be pebbled under the conditions stated in the lemma unless at least one literal is true. □

The clause gadget is accompanied by two *anti-clause* gadgets, $\overline{c_i}$ and $\overline{c_i}'$, that are used to enforce the exact 1-in-3SAT condition. The anti-clause gadgets should also be pebbled with $r_{\overline{c_i}} = 6$ and $t_{\overline{c_i}} = 64$. $\overline{c_i}$ contains all $\overline{x_i}$ literals and $\overline{c_i}'$ contains all $\overline{x_i^*}$ literals. First, the pyramids must be pebbled along the path leading up to p_i. Then, the one negative literal that is not pebbled must be pebbled with a red pebble by using 2 transitions. Finally, the remaining nodes of the gadget are pebbled once using 62 transitions resulting in p_i being pebbled with a red pebble. An anti-clause gadget is shown in Fig. 3-5.

Lemma 3.6. *Given $r_{\overline{c_i}} = 6$ and $t_{\overline{c_i}} = 64$, the anti-clause cannot be pebbled if less than 2 negative literals are true.*

Proof. Without loss of generality, we will assume we are pebbling the anti-clauses containing the $\overline{x_i}$ variables. Each $\overline{x_i}$ variable needs 2 transitions to pebble and unpebble. The remaining vertices need $t = 43$ transitions to unpebble. Therefore, the total number of transitions and red pebbles needed is $r = 9$ and $t = 45$. □

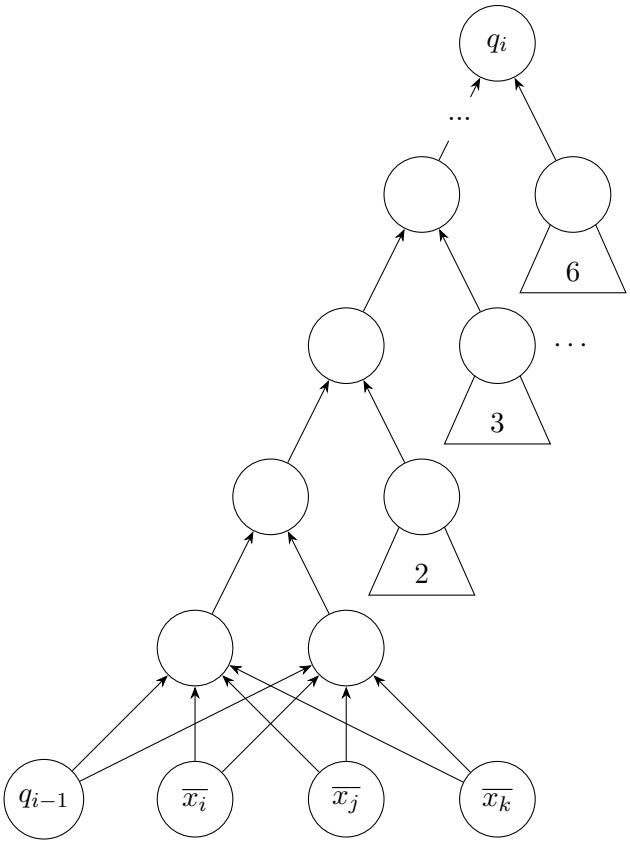

Figure 3-5: Example of a anti-clause gadget with $r = 3$ and $t = 5$. The number of red pebbles that is needed to fill this gadget is 6 (including the two red pebbles that are present on the true negative literals).

Lemma 3.7. *Each clause gadget must contain exactly one true literal and each anti-clause gadget must contain exactly two true literals at the end of Phase 1 before the clause verification phase.*

Proof. The following are the different possible ways red pebbles can reside on the nodes for each variable gadget:

1. A clause contains 0 gadgets set in the true configuration at time t during the clause verification phase. If a clause contains 0 true literals, then we must pebble the clause using ≥ 6 red pebbles. Given that all variables gadgets must contain 3 red pebbles by the end of Phase 1, we must obtain the 2 extra pebbles from another variable gadget. Obtaining the 2 extra pebbles from the variable gadget results in 2 extra transitions during Phase 1 or a previous time $t' < t$ in the clause verification phase. Furthermore, these two pebbles must be deleted and reinserted back into the other variable gadget resulting in two more transitions.

2. A clause contains 2 variable gadgets set in the true configuration. In this case, the clause gadget does not save any transitions since recomputation in memory is free. However, this also means that both corresponding anti-clause gadgets need one more red pebble placement to pebble them. This results in at least 2 additional transitions to turn the blue pebbles on the negative literals to red plus 2 additional transition from Phase 1 or some time t' before the current time to obtain the extra necessary red pebbles. This results in a net gain of 4 additional transitions.

3. A clause contains 3 true variables. This is the same case as 2 with net transitions change (i.e. number of transitions needed to pebble the anti-clauses minus the number of transitions saved) of 8 instead.

4. Without loss of generality, a clause contains the pair x_i and $\overline{x_i^*}$ that are pebbled with red pebbles and $\overline{x_i}$ is pebbled with a blue pebble and x_i^* is not pebbled. In order to pebble the corresponding anti-clause, one pebble must be removed from x_i using one transition and the blue pebble on $\overline{x_i}$ must be turned into a red pebble using an additional transition. There will be a net increase of at least one additional transition with each variable gadget that is set in this configuration.

5. A variable gadget contains more than 3 pebbles. (For instance, a variable gadget could contain red pebbles on $\overline{x_i}$, $\overline{x_i^*}$, and x_i.) Recall that at the end of Phase 1, all variables gadgets are each pebbled with 3 red pebbles. Therefore, in order for a literal to be pebbled with more than 2 red pebbles, at least one transition is used to delete the pebble from a variable gadget by the end of Phase 1 (assuming that the pebble is not one of the 6 red pebbles used to pebble the clause gadgets). If the red pebble is moved onto a positive literal node then one transition is used to delete the pebble from its previous node. If it is moved onto a negative literal node, then 2 transitions are used to delete the pebble from its previous node and place the pebble on its new node. If the pebble movement is unnecessary for satisfying a clause or anti-clause, then it would not occur. If the movement is necessary, then at least one other transition per variable that contains more than 3 pebbles is necessary resulting in more transitions than that allowed.

6. The variables switch value after some clauses are satisfied. If the variables switch from the

true configuration to the false configuration to satisfy some anti-clause, then the switch would result in at least 4 transitions per switch exceeding our allowed bound on transitions. If the switch was from a false configuration to a true configuration, then an extra 2 transitions per switch is necessary. Suppose that some of these extra transitions are credited to the transitions necessary for pebbling a clause. In the case of the false to true switch, none of the switches can be credited to satisfying the corresponding previous clause since the switch needs to occur before the first clause or anti-clause is unsatisfiable. Therefore, no extra transitions can be saved from previous clauses or anti-clauses since all clauses were satisfied (and none of the previously mentioned cases occurred). The problem occurs when one of the previously false variables could be turned to true and a true variable can be turned to false during clause verification. However, this results in 2 extra transitions that cannot be shared in the next clause verification since two of the true literal red pebbles must be removed from the variable that was turned to false since all variables must be set by the conclusion of the clause verification phase. In the other case, if a variable needs to be switched from true to false, then one can only charge the transitions necessary for deleting the pebbles on the positive literals to satisfying the previous clause. However, the 2 transitions for turning the negative literals to true cannot be charged to clause satisfaction.

The lemma follows from the set of cases mentioned above.

□

Given these gadgets, we are ready to proceed with the reduction from Positive 1-in-3 SAT.

3.5 Reduction from Positive 1-in-3 SAT

Given a Positive 1-in-3 SAT expression, ϕ, we create a variable gadget for each of the n variables and a clause and two anti-clause gadgets (one for $\overline{x_i}$ and one for $\overline{x_i^*}$) for each of the m clauses. The gadgets are linked together as shown in Fig. 3-6.

Each variable gadget is connected to the next by the set of vertices Q consisting of nodes $q_i \in Q$. Each variable gadget is also connected to the pebble sink path consisting of vertices $g_i \in G$ and to the pebble hold nodes s_i, s_i', and s_i''. For each clause gadget, we connect it with its corresponding anti-clause gadgets via the nodes in the set $p_i \in P$. The final anti-clause gadget in the chain of

clause and anti-clause gadgets is connected to the bottom of the chain of variable gadgets. Finally, all variable gadgets are connected to pebble hold nodes, $s_i, s_i', s_i'' \in S$ that are also along a path and ensure that all red pebbles end on these set of nodes. There are no transitions allocated for these nodes; therefore, any red pebbles that are used to pebble these nodes must remain.

We let $a_n = 3n + 6$ and $a_i = a_{i-1} + 3$. Therefore, we set $r = 3n + 6$ and $t = 93m + 3n + 22 + \sum_{i=1}^{n} \sum_{j=1}^{3} \frac{(a_i - j + 1)(a_i - j)}{2}$ for the entirety of the construction.

We now provide an argument that red-blue pebbling with no deletions is in NP.

Lemma 3.8. *Given a DAG $G(V, E)$ where $n = |V|$, and parameters r and t and a pebbling strategy, we can check whether the strategy works in time $O(n^2)$.*

Proof. We can solve any red-blue pebbling game using $O(n^2)$ transitions given a reasonable number of red pebbles. We can achieve this by performing the pebbling greedily. If $r < \max(indegree(v))$ for some vertex v, then the pebbling cannot be done. Otherwise, all other pebblings can be completed using $(2d + 1)n$ transitions where $d = \max(indegree(v))$ for all vertices v in a graph G with n vertices. On the other hand, a pebbling can never be performed if $t < n - r$. Therefore, we seek to show that $n - r \leq t \leq (2d + 1)n$ and $r \geq d$ are necessary conditions to valid strategies. To show that all pebblings can be completed using $(2d + 1)n$ transitions, first, topologically sort the vertices in the DAG, then perform pebbling according to the topological sort.

No transitions are used to pebble the predecessors of the first node in the topological sort and only 1 transition is used to remove a pebble from the node after it has been pebbled. Now for each of the outgoing edges of this node, the number of times this vertex will need to be pebbled and removed is at most the number of outgoing edges from the node. Since all nodes that are predecessors of the current node in the topological sort must be pebbled before the current node, the current node can be pebbled by using d transitions to make all predecessors contain red pebbles, pebble the current node, and then use $d + 1$ transitions to convert all red pebbles to blue pebbles. There are n nodes that need to be pebbled in this way, therefore, at most $(2d + 1)n$ transitions are needed to pebble the graph. Therefore, the maximum number of transitions needed to pebble a graph is $2dn = O(n^2)$.

Using only $O(n^2)$ transitions results in a time of pebbling that is $O(n^2)$. Therefore, checking the pebbling strategy should not take more than polynomial amount of time (if $t = \omega(n^2)$ and $r \geq d$, then the graph can always be pebbled). □

Theorem 3.2. *Generalized Red-Blue No-Deletion pebble game on a DAG with maximum in degree 7 is NP-Complete by reduction from Positive 1-in-3 SAT.*

Proof. We first show that given a solution to ϕ, we can construct a solution to our construction using the amount of red pebbles and transitions as described above. We first set the variables according to the assignment provided by the satisfying assignments to ϕ. Then, we pebble the remainder of the construction according to the steps provided in the previous section.

Now we prove that given a satisfying pebbling strategy to our construction, we also have a satisfying assignment for ϕ. First, during Phase 1 of pebbling the construction, the variable gadgets as described in Section 3.4 are pebbled using the number of red pebbles and transitions described in Lemma 3.3. By Lemma 3.4, at the end of Phase 1, the variable gadgets all have at least one pair of x_i and x_i^* or $\overline{x_i}$ and $\overline{x_i^*}$ pebbled with red pebbles. Each pyramid of the variable gadget needs to be pebbled in order to pebble the pebble sink path which is necessary to pebble before the clause verification phase. No more than 3 red pebbles can remain in each variable gadget since the next variable gadget uses the number of red pebbles minus the 3 that remain in the previous variable gadget. The remaining 6 pebbles after all variable gadgets are pebbled will be used to pebble the clause and anti-clause gadgets. In order for the clause and anti-clause gadgets to be pebbled, exactly 1 literal of each clause must be true as proven by Lemma 3.7.

Each clause and anti-clause gadget uses the number of red pebbles and transitions as given in Lemma 3.5 and Lemma 3.6 in order to be pebbled. After all clauses and anti-clauses have been pebbled, we can proceed with Phase 2 of pebbling the variables. The cost of pebbling the variables during Phase 2 is given in Lemma 3.3.

Since the number of transitions thus far cover the pebbling of the variable gadgets, the pebble sink path, and the clause/anti-clause gadgets, there does not remain any transitions for the pebble hold nodes. Therefore, all the red pebbles will be used to pebble these pebble hold nodes. By Lemmas 3.4 and 3.7, if there exists a valid strategy to pebble the configuration, then there exists a valid variable assignment for ϕ.

The problem is in NP by Lemma 3.8. Therefore, the problem is NP-Complete by reduction from Positive 1-in-3SAT. □

Corollary 3.1. *Generalized Red-Blue No-Deletion pebble game on a DAG with maximum in degree 2 is NP-Complete by reduction from Positive 1-in-3 SAT.*

Proof. This follows as an immediate corollary of Theorem 3.3 by using the gadgets described in [GLT79]. □

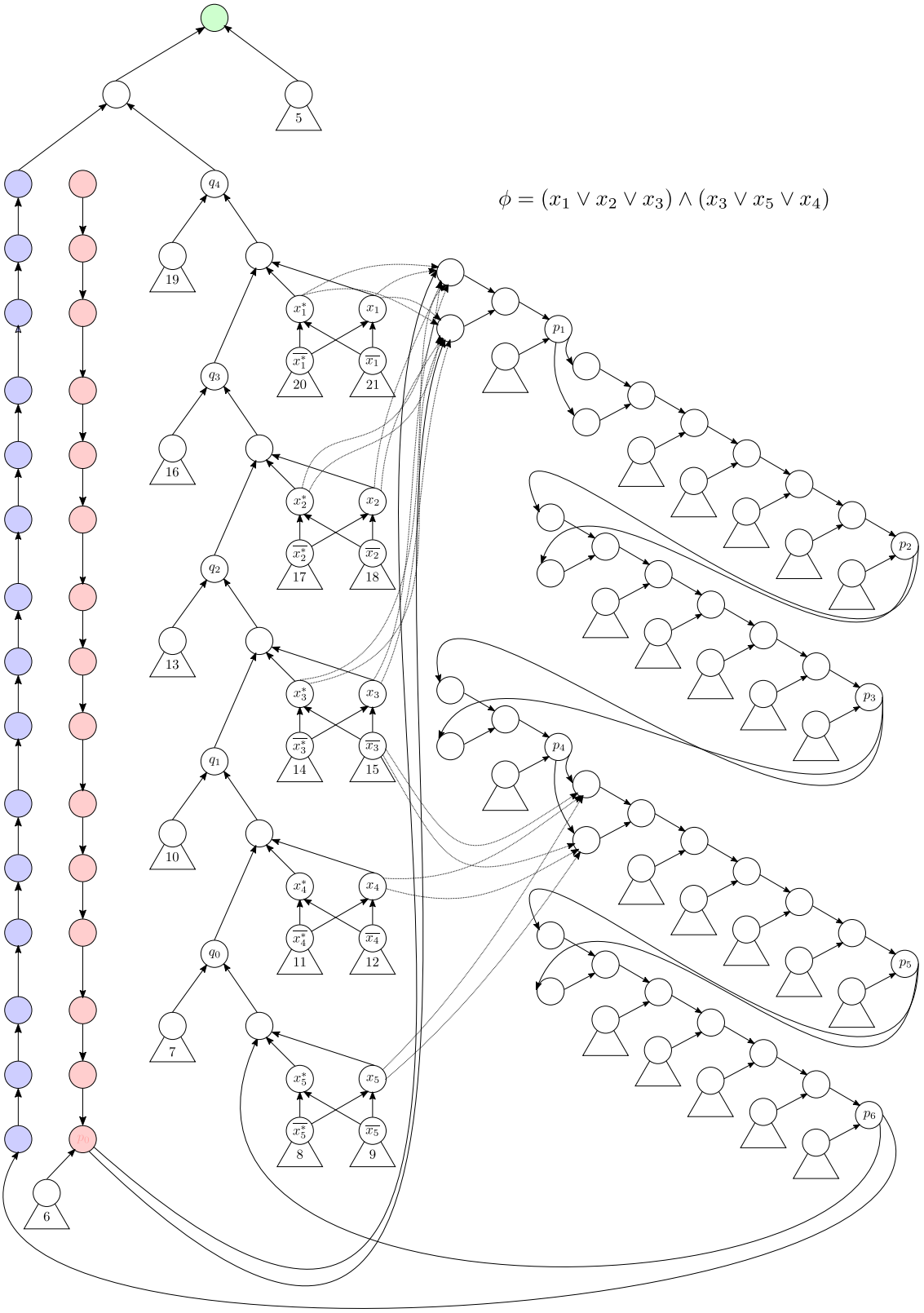

Figure 3-6: Example construction given $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_5 \vee x_4)$. Blue nodes represent the pebble hold nodes and red nodes represent the pebble sink path. The green node is the target node that needs to be pebbled in the end. Note that many of the edges for variable nodes have been omitted for clarity.

Chapter 4

Red-Blue Pebble Game Parameterized by Number of Transitions is W[1]-hard

In this section, we prove that the red-blue pebble game with deletion and target vertices parameterized by k, the number of red-to-blue or blue-to-red transitions is W[1]-hard by reduction from the W[1]-complete problem, Weighted q-CNF Satisfiability. It was previously shown by [DF95] that Weighted q-CNF Satisfiability is W[1]-complete for any fixed $q \geq 2$. In order to maximize the similarity to our previous reductions, we will be reducing from Weighted 3-CNF SAT via a parameterized reduction.

It has been noted that this result seems superfluous given the NP-hardness result for 0 transitions given in Chapter 3. However, we note that an NP-hardness result does not necessarily supersede a parameterized complexity result since they are different complexity domains. Furthermore, the techniques presented in this chapter are techniques that could be important for future proofs of hardness or graph family constructions.

Definition 4.1 (Weighted q-CNF Satisfiability [DF95])**.** *Given a CNF formula, ϕ, a set U of variables where $n = |U|$, and a set C of clauses and $m = |C|$ where the number of literals per clause is at most q, determine whether there is a satisfying assignment for ϕ of truth values to the variables in U such that the number of variables that are true is k.*

Given a 3-CNF formula, we first create two clauses for each of the variables: for all $x_i \in U$ we add the clauses $(x_i \vee x_i \vee \overline{x_i}) \wedge (x_i \vee \overline{x_i} \vee \overline{x_i})$. Note that if a truth value is assigned to x_i, then the clauses must be true. The presence of these clauses is to ensure that each of the variables are assigned a truth value.

The reduction transforms an instance of Weighted 3-CNF SAT with parameter k to an instance of red-blue pebble game *with deletion* (note that this is a different model from the one presented in Chapter 3) such that the reduced instance is allowed $r = 7n - 4k + 1$ pebbles and $2k$ red-blue transitions.

We first provide an overview of our proof techniques. Then, we describe the gadgets used in our proof. Finally, we provide the proof that the red-blue pebble game defined in Section 1.2.2 is W[1]-hard.

4.1 Proof Overview

Given a Weighted 3-CNF SAT expression, ϕ, we first duplicate all the variables and clauses until n' the number of new variables including duplicates follows the rule $\frac{3n'}{4} > k$. From here onwards, we refer to ϕ to be the new 3-CNF expression (with the duplications) and n to be the number of new variables.

As in the proof given in Section 3, we create a set of variable gadgets that are connected to a set of clause gadgets that check whether each clause is satisfied according to the truth settings of the variables. The k true variables conditions is enforced by the *All-False* and *k-True-Variables* gadgets which first force all variable to be set to false and then picks exactly k variables to set to true. The problem is parameterized by the number of transitions, k, and the number of red pebbles is limited by some number that is polynomial in the number of variables in ϕ. All of the transitions will be used before the clause gadgets are pebbled. Therefore, all pebblings of all gadgets after the variable gadgets, the All-False gadget, and the k-True-Variables gadget are pebbled using only red pebbles and no transitions.

We will now describe the gadgets that are used in the reduction.

4.2 Gadgets

4.2.1 Variable Gadget

The variable gadgets are used to represent the variables that are in U and are present in ϕ (i.e. we do not create a variable gadget for variables that are not present in ϕ). We again categorize the complete pebbling of the variable gadgets into three phases: Phase 1, Phase 2, and Phase 3. During Phase 1, each variable must be pebbled in the following way. The pyramid gadgets within each variable gadget are first pebbled with red pebbles and one red pebble remains on the apex of each pyramid gadget. See Fig. 4-1.

After all variable gadgets have been pebbled once (i.e. both $\overline{x'_i}$ and $\overline{x_i}$ are pebbled), the x'_i nodes must be pebbled with red pebbles. In order to pebble the x'_i nodes, the remaining red pebbles will be used as well as the red pebbles on $\overline{x'_i}$. The corresponding red pebble on $\overline{x'_i}$ is either turned to blue or removed. At most k of these red pebbles may be turned to blue since we are given only $2k$ transitions and each of the blue pebbles must be reverted back to red at some point in the future (proof will be provided later). The three vertices representing x_i remain un-pebbled and a red pebble remains on each $\overline{x_i}$. Each vertex of x'_i as well as $\overline{x_i}$ are connected to the All False gadget (described below). The All False gadget must be pebbled after the variable gadgets since all other subsequent pebbling depends on the set of $2k + 1$ nodes that were pebbled during the pebbling of the All False gadget.

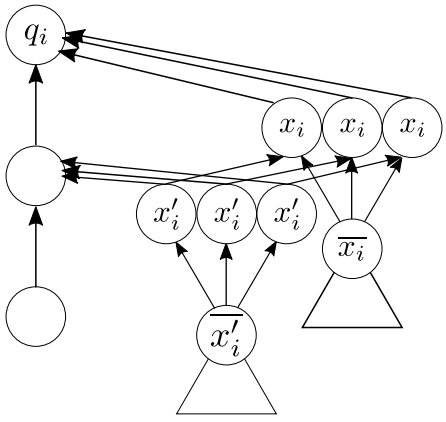

Figure 4-1: Variable gadget.

Lemma 4.1. *All variable gadgets must be in the false configuration after Phase 1.*

Proof. During Phase 1, the All False Gadget 4-2 must be pebbled. We are allowed at most a_n pebbles and the All False gadget consists of a set of $2k+1$ nodes each of which costs a_n, $a_n - 1$, $\ldots$, $a_n - 2k$ pebbles to pebble (i.e. have indegree a_n, $a_n - 1$, $\ldots$, $a_n - 2k$). Thus, the only possible pebbling configuration is the configuration that leads all variable gadgets to be in the false configuration and the remaining pebbles are used to pebble the other nodes that each of the $2k+1$ are dependent. □

During Phase 3 of pebbling the variable gadgets, the other nodes within the gadget are pebbled using the red pebbles that are left on the gadget from Phase 2. This phase requires no transitions since the extra red pebbles from the clause and pebble sink path gadgets can be used to pebble the variable gadgets during this phase.

4.2.1.1 All False Gadget

The All False gadget is used to check that all variables are initially set to false. See Fig. 4-2. It consists of $2k+1$ vertices with unbounded indegree with predecessors x'_i and $\overline{x_i}$ for all i. Furthermore, its predecessors also contain $a_n - 4n - i$ nodes for $i = \{0, \ldots, 2k\}$ to use up the other $a_n - 4n - i$ extra pebbles. The $2k+1$ nodes from the All False gadget are then connected to the k-True-Variables gadget and all the clause gadgets.

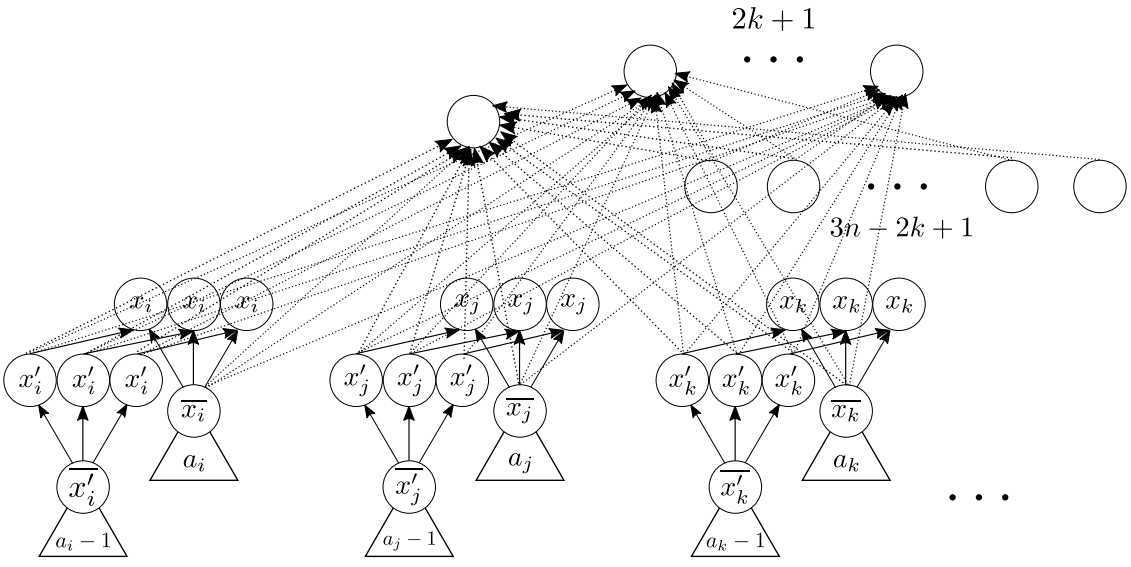

Figure 4-2: The All False gadget consists of $2k+1$ nodes that all have x'_i and $\overline{x_i}$ as predecessors. Each of these $2k+1$ nodes are connected to the k-True-Variables gadget and the clause gadgets.

4.2.1.2 k-True-Variables Gadget

Phase 2 of the variable pebbling phases consists of resetting a set of k variables to true. The k-True-Variables gadget is present to constrain the number of true variables to exactly k. The k-True-Variables gadget consists of a single unbounded indegree vertex with x_i as predecessors for all i. After passing through the All False gadget, k variable gadgets must be switched from the False position to the True position by moving $3k$ pebbles from x'_i to x_i and using k transitions to move k red pebbles to $\overline{x'_i}$. As we will show in the next few gadgets, k transitions must be used to pebble $\overline{x'_i}$ nodes with red pebbles. See Fig. 4-3 for an example.

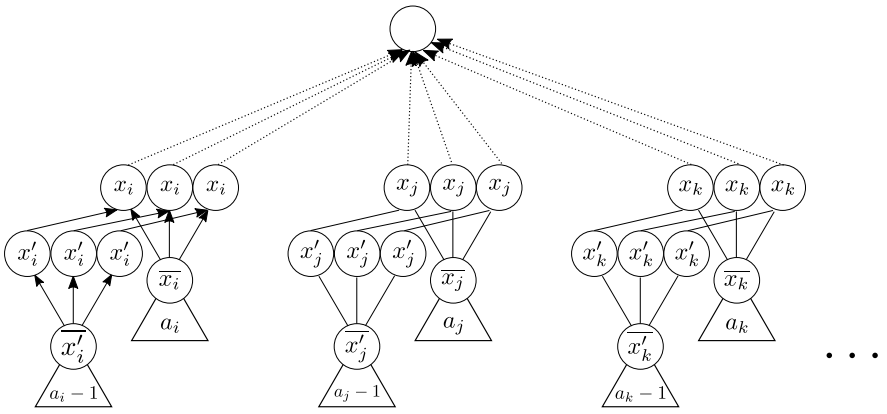

Figure 4-3: k-True gadget connects to all x_i for all i.

Lemma 4.2. *At the end of Phase 2, exactly k variables are set in the true configuration.*

Proof. In order to prove this lemma, we first prove the following two claims.

Claim 4.1. *At least $3k$ pebbles must be removed from x'_i and used to pebble x_i with red pebbles and k pebbles must be removed from $\overline{x_i}$.*

Proof. In order to pebble the k-True-Variables gadget, all the nodes that are incident to the central node must be pebbled. Given that we only have $3n - 4k$ pebbles remaining after pebbling all the variable gadgets in Phase 1 and the All-False gadget, we must use $4k$ pebbles from the variable gadgets in order to pebble the remaining $4k$ slots. The pebbling strategy proceeds as follows.

First, pebble $3n-4k$ x_i's. Of these, remove $4k$ pebbles from the corresponding x'_i and $\overline{x_i}$. Using these removed pebbles, pebble the remaining k variables (i.e. the remaining $4k$ x_i's).

Since the $2k+1$ pebbles from the All-False gadget are needed to pebble the k-True gadget, we cannot remove these pebbles. Thus, the smallest number of pebbles we need to remove is $4k$ from the variable gadgets. □

Claim 4.2. *No pyramid gadgets in the variable gadgets may be repebbled using $\leq 2k$ transitions in Phase 2.*

Proof. In order to pebble the k-True-Variables gadget, the $2k+1$ nodes pebbled during the All-False phase must remain pebbled with red pebbles. Even if all red pebbles were removed from all x'_i nodes, the number of red pebbles available is not enough to pebble the pyramid with the smallest cost among all pyramids in variable gadgets. In order to obtain the $2k+1$ red pebbles, at least $2k+1$ transitions need to be used to turn the red pebbles on the All-False gadget to blue which exceeds our limit of $2k$ transitions. □

Claim 4.3. *At least k pebbles must be turned from blue to red on $\overline{x'_i}$ using k transitions.*

Proof. The Pebble Sink Path gadget uses $3n-4k-6$ pebbles and the clause gadgets use 5 pebbles each; therefore, $3n-4k$ of the pebbles that are used to pebble the k-True-Variables gadget (or the variable gadgets) must be removed from the gadgets and used to pebble the pebble sink path. Therefore a total of $4n$ pebbles remain on the variable gadgets. By Lemma 4-4, each variable gadget must be set in either the true or false configuration (in other words, each variable gadget either has all x_i nodes pebbled with red pebbles or $\overline{x'_i}$ pebbled with a red pebble *and* all x'_i nodes pebbled with red pebbles or $\overline{x_i}$ pebbled with a red pebble). In order to transform a variable gadget from the false configuration to the true configuration, at least one transition must be used to pebble $\overline{x'_i}$ with a red pebble provided that the pyramid under $\overline{x'_i}$ cannot be repebbled. We showed this in Claim 4.2 that in order to repebble this pyramid, more than $2k$ transitions are needed.

Therefore, the only way to change the truth value of a variable gadget is to use 1 transition per gadget. Suppose that a red pebble is removed from a x'_i node. Then, the node $\overline{x'_i}$ must be pebbled by Lemma 4-4. Furthermore, if a node is removed from $\overline{x_i}$, then all x_i nodes must be pebbled which implies that $\overline{x'_i}$ must also be pebbled. Therefore, the minimum number of variables that need to be switched from false to true is k since $4k$ pebbles can be removed from k variables and switched from false to true using the strategy provided by Claim 4.1. The k transitions are used to turn blue pebbles on k $\overline{x'_i}$ nodes to red.

□

Claim 4.4. *The minimum number of transitions that are needed for Phase 2 is k.*

Proof. This follows directly from Claim 4.3. □

Therefore, since all variables either have x'_i and $\overline{x_i}$ or x_i and $\overline{x'_i}$ pebbled, all variables are set to either true or false. Furthermore, as the claims show, at most k variables are set to true. □

4.2.1.3 3-or-None Gadget

The 3-or-None gadgets are used to ensure that every variable either has 3 pebbles on each x_i or x'_i or none on them. This is to ensure that the player cannot cheat by using less than 3 pebbles to set either x'_i or x_i true. A 3-or-None gadget is created for each variable. The 3-or-None gadget consists of sets of 2 vertices one picked from x_i and the other picked from x'_i. All such pairings are connected to a path with vertices of indegree 5 (the other vertices are roots) so that only one pebble is allowed to go through the path. See Fig. 4-4 for an example of a 3-or-None gadget. This gadget can be pebbled using 5 pebbles. However, more pebbles will not ensure that the gadget can be pebbled if it does not satisfy the invariant as stated in Lemma 4.5. In other words, more pebbles does not guarantee that the gadget can be pebbled without using any transitions. Attached to each node of the 3-or-None gadget are $3n - 4k - 3$ roots that have outgoing edges to the nodes in the path in the gadget. The All-False termination node is also connected to every node in the path of the 3-or-None gadget.

Lemma 4.3. *For every variable gadget that does not follow the condition specified in Lemma 4.5 and contains less than* 6 *red pebbles, at least two transitions are required to satisfy the gadget.*

Proof. Suppose that less than 6 red pebbles are on a variable gadget and no pair of components x_i and $\overline{x'_i}$ or x'_i and $\overline{x_i}$ are pebbled with red pebbles, then the distribution of red pebbles are ones that partially fill in some x_i or x'_i and does not pebble $\overline{x_i}$ or $\overline{x'_i}$. Therefore, in order to pebble the x_i or x'_i that does not contain a red pebble, we have to use two transitions to turn the blue pebble on $\overline{x_i}$ or $\overline{x'_i}$ to red. □

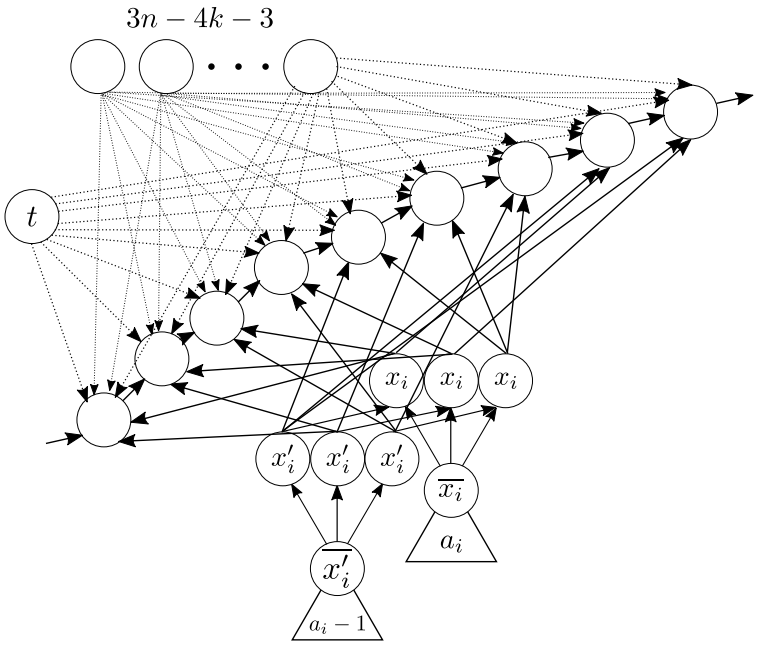

Figure 4-4: 3-or-None gadget. One is created for every variable.

Lemma 4.4. *Suppose some variable gadgets are set in the configuration with $\overline{x_i}$ and $\overline{x'_i}$ are pebbled, then the number of transitions needed to pebble both the All-False gadget and the 3-or-None gadget is greater than $2k$.*

Proof. Suppose that b variable gadgets have red pebbles on $\overline{x_i}$ and $\overline{x'_i}$. During Phase 2, $4k$ red pebbles must be removed and moved to the x_i nodes. In order for this to occur, some number of red pebbles are removed. For any pebble placed during Phase 1 removed, at least 2 transitions must be used to reset the value of the variable unless only the red pebble on $\overline{x_i}$ is removed and no other red pebbles are removed from the gadget. For each of the variable gadgets where the red pebble on $\overline{x_i}$ is removed, in order for the variable gadget to have red pebbles on $\overline{x_i}$ and $\overline{x'_i}$, two transitions must be spent on placing a red pebble on $\overline{x_i}$ during Phase 2. We know that having a variable gadget in the configuration $\overline{x_i}$ and $\overline{x'_i}$ saves 2 pebbles. However, in order to achieve this configuration, we must spend 4 transitions per gadget in Phase 2 which does not make up for the amount that is saved by this configuration. The removal of any other red pebble from a variable gadget requires at least two transitions; therefore, the optimal removal number is 4 (instead of 3 which would lead to a $\overline{x_i}$ and $\overline{x'_i}$ configuration). □

Lemma 4.5. *In order to satisfy all 3-or-None gadgets using at most $2k$ transitions, the only possible*

configurations for all pebbles must be placed in one of the two pairs of components: $\overline{x_i'}$ *and* $\overline{x_i}$ *or* x_i *or* $\overline{x_i'}$.

Proof. By Lemma 4.4, pebbles cannot be placed in the configuration $\overline{x_i}$ and $\overline{x_i'}$ using at most $2k$ transitions. Therefore, in order for a red pebble placement to satisfy the corresponding 3-or-None gadget without using 2 additional transitions, the must be placed in the pairs given in the lemma or red pebbles must be on all nodes x_i and x_i'. Suppose that a variable gadget has this configuration. Then, two pebbles will be removed from some other variable gadget. Since no variables can be in the configuration $\overline{x_i}$ and $\overline{x_i'}$ by Lemma 4.4, if a variable is not set in a configuration, then two transitions are used. By Lemma 4.3, at least k variables must be set to false at the end of Phase 2. Therefore, each variable must be in the configurations as stated in this lemma in order to satisfy all 3-or-None gadgets. □

The remaining gadgets may be pebbled with red pebbles without using any transitions.

4.2.2 Pebble Sink Path Gadget

The Pebble Sink Path Gadget is used to take up $3n - 4k - 6$ pebbles that were used in the k-True-Variables gadget (and were left over after passing through the gadget) leaving only 5 pebbles for the remaining parts of the winning path. The Pebble Sink Path occurs directly after the clause gadgets path and must be pebbled before the clause gadgets are pebbled. This sink path consists of $3n - 4k - 6$ pyramid gadgets of successively smaller value starting from $3n - 4k - 1$. See Fig. 4-5 for an example.

4.2.2.1 Clause Gadget

After the set of 3-or-None gadgets comes the Clause gadgets which are used to ensure that the 3SAT clauses are satisfied by the assignments. The clause gadget can only be pebbled with the 5 extra pebbles that remain after the pebble sink has been pebbled. Given that all $2k$ transitions are used in Phase 2 and/or the pebbling 3-or-None gadgets phase, no transitions can be spent in Phase 3 or pebbling the clause gadgets. The output of the clause path must be connected to each vertex of the Pebble Sink Path gadget. See Fig. 4-6.

Finally, the target vertex can be pebbled with a red pebble if and only if all previous gadgets

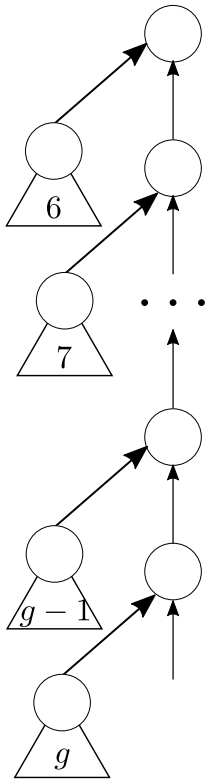

Figure 4-5: Pebble sink that captures $3n - 4k - 6$ pebbles leaving 5 pebbles to be used in the clauses. Here $g = 3n - 4k - 1$.

are pebbled according to the necessary rules and conditions. The $2k + 1$ nodes from the All-False gadget are also predecessors of this target vertex.

For an example reduction, see Fig. 4-7. The target vertex that must be pebbled is the one colored blue.

Lemma 4.6. *The clause gadget can be pebbled with* 5 *pebbles (not including the red pebble on* p_{i-1}*) if and only if at least one of the variable gadgets that connects to it is set in the true configuration.*

Proof. If at least one literal is true in the clause gadget, then we can pebble the gadget in the following way. First, pebble all the other literals that are not set to true. Pebbling the other literals requires at most 4 pebbles. Finally, pebble the bottom layer of the pyramid for the literal that is set to true. Then, the pyramid can be pebbled using the method described in Lemma 3.2 using five pebbles once the bottom layer of the pyramid has been pebbled.

Based on the proof provided in Lemma 3.2, the cost of pebbling the pyramid in each clause gadget is 5. In order for a red pebble to be placed on the pyramid in the clause gadget (aside

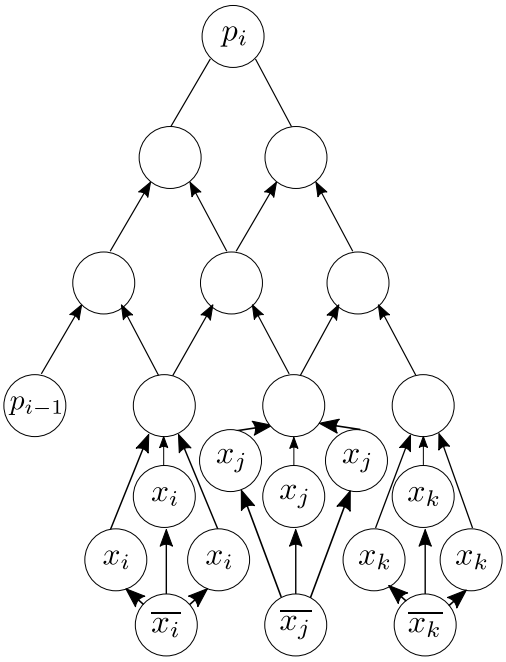

Figure 4-6: Clause gadget.

from the red pebble on p_{i-1}), 3 red pebbles must be used on all literals that are not set in the true configuration. Suppose that 2 pebbles are currently on the pyramid at some time t, then, to pebble the last literal (since it is not set to true) takes at least 4 red pebbles. At the time the last literal is pebbled, as least 2 red pebbles must already be on the pyramid (otherwise, the literal is not the last literal to be pebbled in the gadget). Therefore, if the clause gadget can be pebbled using 5 red pebbles (not including the red pebble on p_{i-1}), then the clause is satisfiable. □

4.3 Red-Blue Pebbling is W[1]-hard

In this section, we prove that red-blue pebbling parameterized by the number of transitions is W[1]-hard using the gadgets as specified in Section 4.2. An example construction is shown in Fig. 4-7. The order of the pebbling is given as the following. First, the variable gadgets are pebbled during Phase 1 of the pebbling which pebbles each pyramid gadget in each variable with a red pebble. Then, the All-False gadget is pebbled which results in all x_i' and $\overline{x_i}$ nodes being pebbled with red pebbles. During Phase 2 of pebbling the variable gadgets, k variables are switched from the false configuration to the true configuration. This in total uses the entirety of the allowed $2k$ transitions. To ensure the $2k$ transitions are used in this phase, the 3-or-None gadgets are pebbled using only

5 red pebbles and no transitions. After the 3-or-None gadgets are pebbled, we pebble the Pebble Sink Path gadget which consumes $3n - 4k - 6$ pebbles. The clause gadgets are pebbled with the remaining 5 pebbles not used in the pebble sink path. Finally, the variable gadgets are pebbled completely using all the pebbles during Phase 3 and the target node as indicated in Fig. 4-7 pebbled with a red pebble. The total number of red pebbles necessary is $r = 7n - 2k + 1$ and the total number of transitions is $t = 2k$.

Theorem 4.1. *Red-Blue Pebbling parameterized by the number of transitions k is W[1]-hard.*

Proof. We first show that our reduction is a valid FPT reduction. As defined above, our reduction is a polynomial time reduction in terms of n, m, and k. The number of nodes created is $O(n+m+k)$ and the number of edges is at most the square of this amount. The number of transitions is determined by the function $f(k) = 2k$ where k is the parameter in Weighted 3-CNF Satisfiability.

Now we will prove that a solution exists in our construction if and only if a solution exists for the expression ϕ. Suppose a solution exists for ϕ, then one can set the variables in the construction to have the truth value given by the solution to ϕ. We can set all the variables to their corresponding truth values using at most $2k$ transitions. Furthermore, we can pebble the remainder of the construction using the prescribed number of red pebbles.

As we proved in Lemma 4.5, the number of transitions that must be used after pebbling the 3-or-None gadgets is $2k$. Therefore, the remainder of the construction must be pebbled using red pebbles and no transitions. We proved in Lemma 4.6 that the clauses can only be pebbled using 5 red pebbles if they are satisfiable. Since the Pebble Sink Path gadget is pebbled after the 3-or-None gadgets are pebbled, they must be pebbled at the time when the clause gadgets are pebbled, leaving only 5 free red pebbles to pebble the clause gadgets. Finally, the path leading up to the blue pebble can be pebbled if all the clauses are successfully pebbled. Therefore, this is a valid reduction from Weighted 3-CNF SAT and the problem is W[1]-hard. □

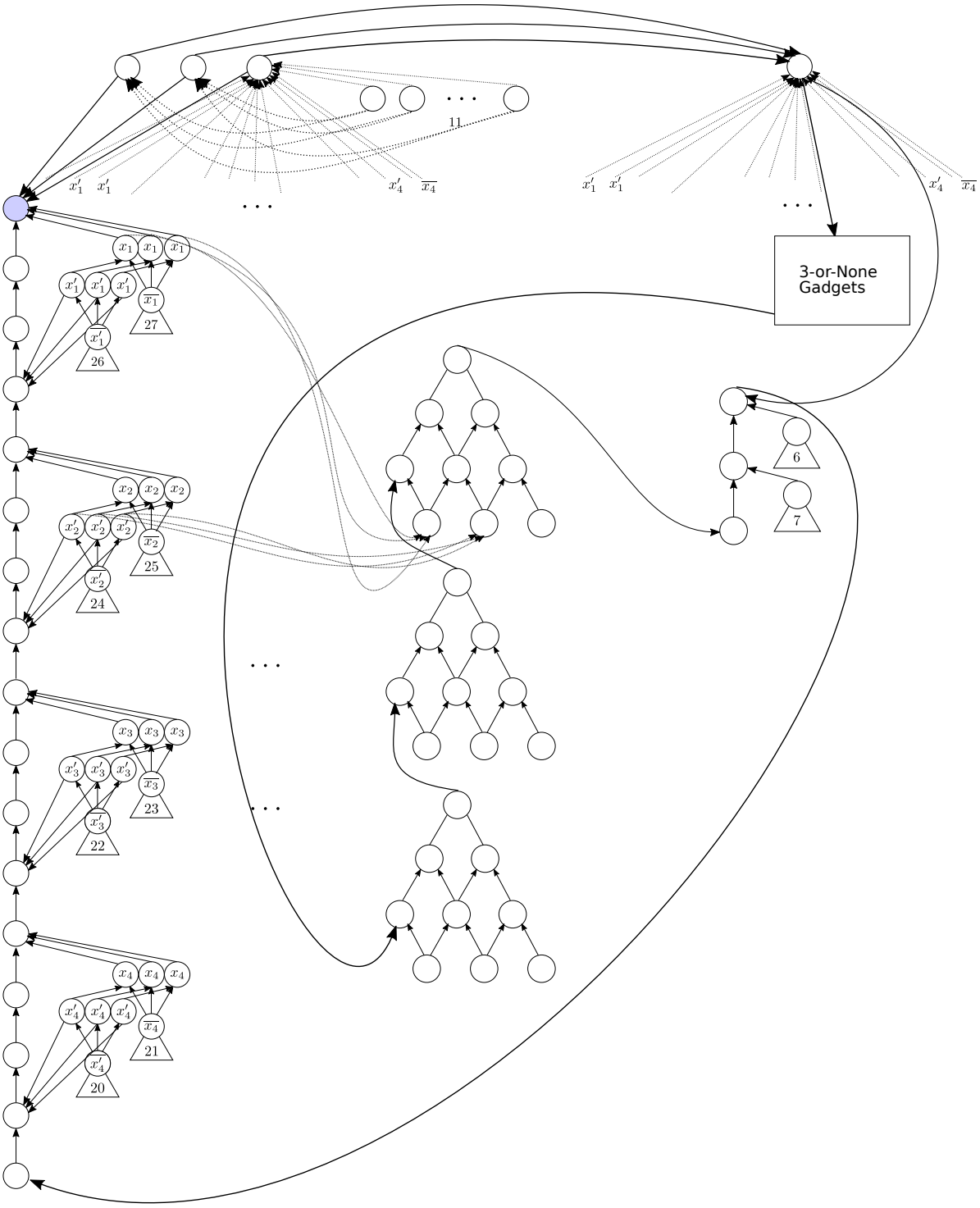

Figure 4-7: Example reduction. The vertex colored blue is the vertex that must be pebbled at the end and can only be pebbled if and only if the 3SAT instance has a solution that sets exactly k variables to True and uses at most $2k$ transitions.

Chapter 5

Inapproximability of Standard Pebbling Number and Moves

In this section, we provide an alternative and perhaps simpler proof of the result presented in [CLNV15] (without the use of the graph products used in [CLNV15]) that the standard pebbling game is inapproximable to any constant additive factor. Then, we show that our proof technique can be used to show that the minimum number of pebbles needed to pebble a DAG is inapproximable to any additive factor $n^{1-\varepsilon}$ for any ε.

5.1 Simpler Proof of the Additive Inapproximability of Standard Pebbling

We first note that the following graph (Fig. 5-1) (sometimes known in literature as the "road graph" [EBL79, Nor15]) requires a number of pebbles that is the width of the graph to pebble all the outputs.

Suppose we now replace all variable nodes in the proof provided in [GLT79] with road graphs of width $K + 1$. The modified quantifier gadgets are shown in Fig. 5-2.

To see details of the original proof of the PSPACE-completeness of standard pebbling, please refer to [GLT79]. In order to make the correct changes to the variable gadgets, we must make a few other changes to the quantifier and clause gadgets. Fig. 5-3 shows the changes we must make to the clause gadgets in order to prove the following lemmas about the PSPACE-hardness of

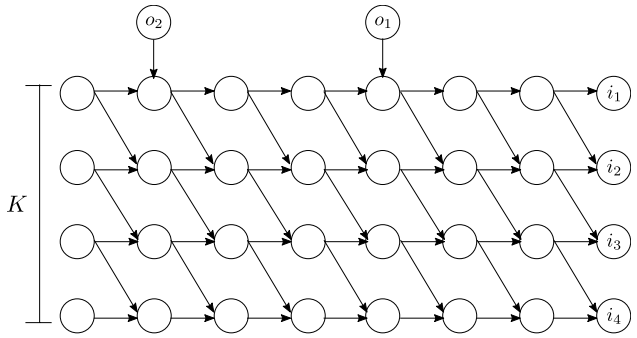

Figure 5-1: Road graph gadget. Here, in this example, a minimum of 5 pebbles are necessary to pebble o_1 and o_2. 4 pebbles must be used to pebble i_1, i_2, i_3, and i_4 and one more pebble is necessary to pebble o_1 since the four pebbles used to pebble i_1, i_2, i_3 and i_4 must remain on the road graph in order to pebble o_2. K is the width of this road graph gadget. In this example, $K = 4$.

approximating within an additive factor of $n^{1-\varepsilon}$.

In short, the proof relies on the fact that each quantifier gadget requires $2(K+1)$ pebbles to set the corresponding variable to true or false. Furthermore, a clause would consist of binary trees with the bottom layer containing $6(K+1)+K+1$ nodes. In this paradigm, we would need $(K+1)n + 2(K+1) + K$ pebbles to pebble the entire structure used in the proof if the given QBF expression ϕ is satisfiable. Furthermore, if ϕ is unsatisfiable, then there is no way to pebble the structure using less than $(K+1)n + 3(K+1) + K$ pebbles. Thus, if given an approximation algorithm that estimates the number of pebbles needed within an additive factor K, we can distinguish between the case when ϕ is satisfiable (at most $(K+1)n + 2(K+1) + K$ pebbles are needed) and the case when ϕ is unsatisfiable (when $(K+1)n + 3(K+1) + K$ pebbles are needed).

In this construction, K can be any polynomial function of v where v is the number of variables in ϕ and c is the number of clauses (in other words, $K = v^a c^b$ for any constants a and b). The total minimum number of pebbles necessary is $O(Kv)$ and the total number of nodes in the graph is $O(Kv + c)$. We prove the following:

Lemma 5.1. *The provided QBF instance, ϕ, is satisfiable if and only if the number of pebbles necessary to pebble the modified construction of the proof presented in [GLT79] is at most $(K+1)n + 2(K+1) + K$.*

Proof. By the same argument as provided in [GLT79], each quantifier gadget must be pebbled with $2(K+1)$ pebbles before the clause gadgets can be pebbled. In order to pebble the clause gadgets,

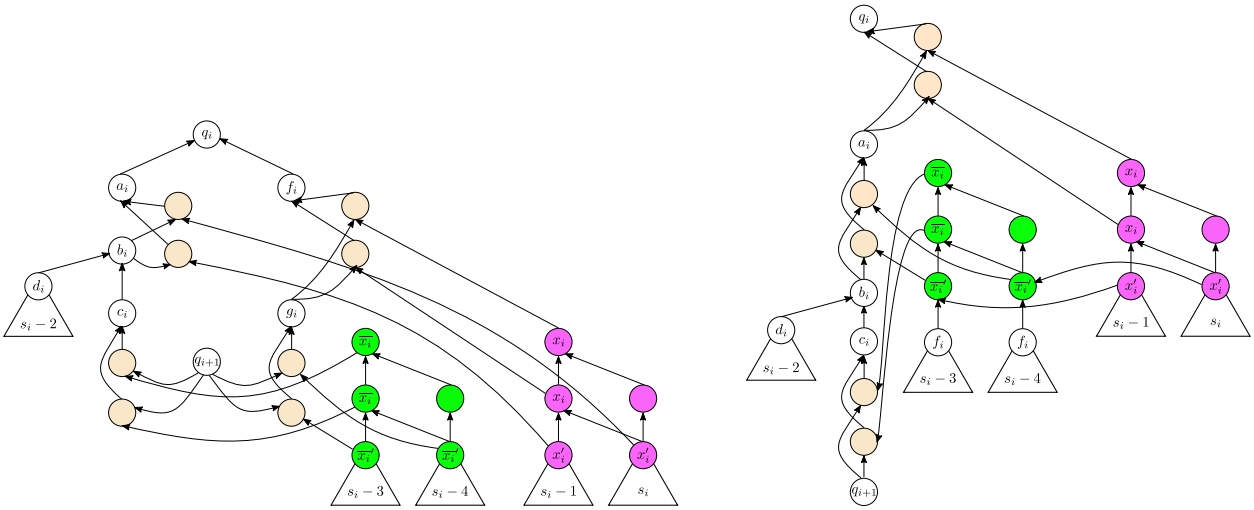

Figure 5-2: Modified quantifier gadgets from [GLT79] with road graphs replacing the variable nodes. The road width for each of the road graphs is $K + 1$ where $K = 1$ in this figure. The left figure is the modified quantifier gadget and the right figure is the modified existential gadget. The green nodes indicate one road graph gadget with width 2 and the purple nodes indicate another road graph gadget with width 2. The yellow nodes are nodes that are part of binary tree gadgets connecting to the road graph gadgets.

$2(K + 1) + K$ additional pebbles are needed provided that at most two literals are false in every clause. All parts of the proof follows as in [GLT79] if we prove that for every cost 1 operation in [GLT79] resulting from pebbling the quantifier gadgets, there is instead a $K + 1$ cost in pebbles, and that the increase in pebbles does not make pebbling any other parts of the gadget easier. First, we prove that the variables must take one of two possible configurations at the end of pebbling all the quantifier gadgets. As proven in [Nor15], a road graph gadget with w inputs and w outputs requires $2w - 1$ pebbles to pebble all the outputs of the gadget. Therefore, all road graphs in the literal gadgets require $2K + 1$ pebbles to be placed in the true configuration. Therefore, if any gadget is not in the true configuration, then $K + 1$ additional pebbles are necessary to pebble the gadget.

As proven in [GLT79], in order to pebble the remaining nodes after the clause gadgets have been pebbled, all variables must be set to either a true or false configuration. We show that with the road graph gadgets, this invariant still holds. Suppose for the sake of contradiction that there exists a variable gadget that is set in a configuration that is not true or false. Then, either, the corresponding pair that needs to be set false is not which is problematic for the same reason as the proof given in [GLT79] or the true component of the gadget is not set which results in $K + 1$

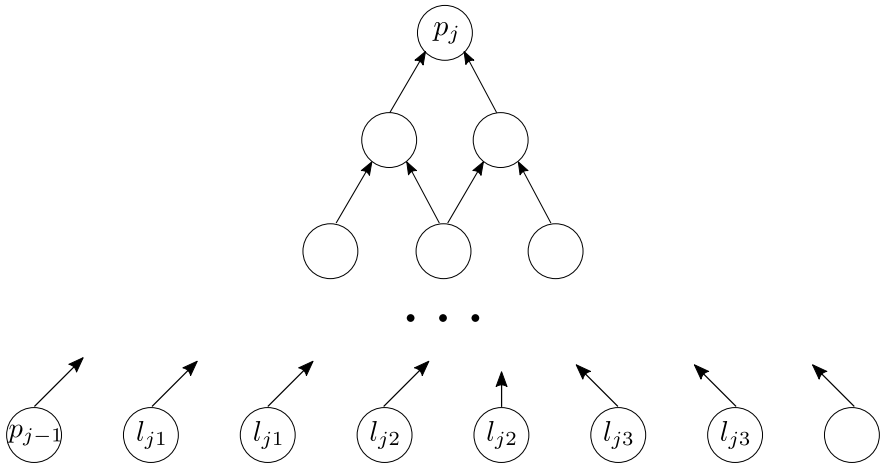

Figure 5-3: The clause gadgets are modified to account for the width $K+1$ variables. Each clause contains width $K+1$ literals and an added K nodes to take up the extra K pebbles that are necessary to pebble the false literals. In this example, $K = 1$ and $l_{j,i}$ for $i \in \{1, 2, 3\}$ are the literals.

pebbles being used to set the corresponding part to the true configuration.

Since all connections with quantifier gadgets are converted to pyramids that can only be pebbled when its predecessors (which includes the previous part of the gadget as shown in [GLT79], see Fig. 5-3 and Fig. 5-2) are pebbled, this is the same condition as that in [GLT79]. The extra pebbles in this construction can only be used to pebble the pyramids attached to each quantifier gadget and each of the pyramids is scaled accordingly. Therefore, the provided QBF instance, ϕ, is true if and only if the number of pebbles necessary to pebble the modified construction of the proof is at most $(K+1)n + 2(K+1) + K$. □

Lemma 5.2. *The provided QBF instance, ϕ, is unsatisfiable if and only if the number of pebbles necessary to pebble the modified construction of the proof presented in [GLT79] is at least* $(K+1)n + 3(K+1) + K$.

Proof. We proved in Lemma 5.1 that all variables must be set in either the true or false configuration. Therefore, the only way that more than $(K+1)n + 2(K+1) + K$ pebbles are used to pebble the construction is if ϕ is false by the proof given in [GLT79]. If ϕ is false (i.e. unsatisfiable), then, there exists a clause gadget which contains three false literals. In order to pebble the clause with three false literals, we must use an extra $K+1$ pebbles since we proved in Lemma 5.1 that all cost 1 operations need to be converted to cost $K+1$ operations. □

We can also use the above argument to show that it is PSPACE-hard to approximate the minimum number of black pebbles needed given v quantifier gadgets to an additive $n^{1-\varepsilon}$ factor for all ε.

Theorem 5.1. *It is PSPACE-complete to determine whether a DAG can be pebbled using the standard pebbling rules to within an additive $n^{1-\varepsilon}$ factor for any ε where n is the number of nodes in the graph.*

Proof. Using the method described above, suppose the cost of pebbling each variable is K. As a function of K, the cost of pebbling a satisfiable ϕ is $Kn + 2(K+1) + K$ whereas the cost of pebbling an unsatisfiable ϕ is $Kn+3(K+1)+K$. As we can see, the aforementioned reduction is a gap-producing reduction with a gap of K pebbles. Then, all that remains to be shown is a K exists that satisfies $K = ((c_1 + c_2K)v + c_3c)^{(1-\varepsilon)}$ for all $0 < \varepsilon < 1$. We can accomplish this by scaling up v and c. We can scale up v and c by duplicating the quantifier variables and clauses. We can further scale up c by adding a path of an arbitrary polynomial number of nodes. We set $K = v^a$ where we pick a such that $\frac{a}{1-\varepsilon} > 2$. Then, v and c are scaled up to v' and c' where $v' \approx \frac{v^{(\frac{1}{1-\varepsilon})}}{2}$ and $c' \approx \frac{v^{(\frac{a}{1-\varepsilon})}}{2}$ if $v > c$ and $K = c^a$ if $c \geq v$. Therefore, for every ε, we can find a K such that it is PSPACE-hard to find an approximation within an additive $n^{1-\varepsilon}$ factor where n is the number of nodes in the graph. □

5.2 Inapproximability of Number of Moves in Standard Pebbling

Now we will prove that approximating the number of moves to any factor 2^s given the number of pebbles s used to pebble the graph is PSPACE-hard. We note that the proof presented in [GLT79] is already a gap reduction, but here we show how to augment this reduction to achieve even greater asymptotic significance.

We use the same general construction as in [GLT79] except we make the following changes:

1. We replace each edge between a variable setting with a "super-hard" graph as seen in, for example, [EBL79].

2. We increase the pebbling prices of all the pyramids to account for the additions of the "super-hard" graphs.

The purpose of the family of graphs shown in [EBL79] is to show an exponential tradeoff between pebbling number and pebbling price. Following the notation in [EBL79], we define these graphs as H_n where n is the minimum pebbling number cost. These graphs exhibit the property that with n pebbles, the pebbling cost in the number of moves is $n!$. With $n+1$ pebbles, the pebbling cost in the number of moves decreases to n^4.

Now, we arrange the graph so that all universal quantifier gadgets are at the top of the reduction structure. Furthermore, we add the n existential quantifiers, y_i, as given in [GLT79] to ensure all variables are set to either True or False (i.e. no double False positions).

To the given ϕ we add the following n universal quantifier variables, $z_1, z_2, \ldots, z_n$ at the top of the construction. We also create a series of clauses of the form $(z_i, \overline{z_i}, z_{i+1})$, $(z_i, \overline{z_i}, \overline{z_{i+1}})$ for all $i \in [1, \ldots, n-1]$. This is to ensure that all boolean variables are set to some setting other than the double false setting. Suppose that all of these graphs H_K have cost K. Then, we need $6n+(K+5)$ pebbles to properly pebble this graph.

If ϕ is true, then the number of moves needed to pebble this graph is $\leq (\sum_{i=K+5}^{6n+(K+5)} i + 3nK^4 + 10(m+n) + c)2^{n_\forall + n}$ where c is some constant. However, if ϕ is false, then the number of moves needed to pebble the graph is $\geq (\sum_{i=K+5}^{6n+(K+5)} i + 10(m+n) + nK!)2^{n_\forall + n}$. Then, the ratio for large $K!$ is an approximation factor of this construction. Specifically, the ratio of approximation is $\frac{(\sum_{i=K+5}^{6n+(K+5)} i + 10(m+n) + 3nK^4 + nK!)}{(\sum_{i=K+5}^{6n+(K+5)} i + 3nK^4 + 10(m+n) + c)} \geq (K-5)!$ for sufficiently large $K \geq (m+n)^6$. Since, $K = (m+n)^{\Theta(1)}$, we can achieve an approximation factor of $(m+n)^{\Theta(1)}!$. In order for this to be a valid reduction K can be at most $(m+n)^{\Theta(1)}$. Therefore, we can show that it is hard to approximate the number of moves needed to pebble a DAG given s pebbles by a factor of $(m+n)^{\Theta(1)}!$ which is better than the factor of $2^{(m+n)^{\Theta(1)}}$ known previously (implicitly) by the construction in [GLT79].

Chapter 6

Parallel Pebbling Model

In this section, we prove some properties related to the parallel graph pebbling model as introduced in [AS15] and described Chapter 1.4, Definition 1.1. First, we give a brief overview of the known results and definitions related to the pROM model. We also define some graph families that are useful in this model of computation. The definition of cumulative complexity as provided by [AS15] is restated in Definition 1.2.

Although somewhat intuitively obvious, we provide an explicit proof for the equivalence between $P(G_{n,\delta})$ and $P^{\parallel}(G_{n,\delta})$.

Lemma 6.1. $Peb(G_{n,\delta}) = Peb^{\parallel}(G_{n,\delta})$.

Proof. Any sequential pebbling strategy, A_p can be simulated by a parallel pebbling strategy, $A_p^{\parallel}$ since $A_p^{\parallel}$ can choose to place one pebble at a time. Therefore, $Peb^{\parallel}(G_{n,\delta}) \leq Peb(G_{n,\delta})$. We now show that there exists a sequential pebbling strategy, A_p, that uses the same number of pebbles to pebble a graph as a parallel strategy $A_p^{\parallel}$. Suppose that at time t_i, a set of pebbles S_i are added to $G_{n,\delta}$ under algorithm $A_p^{\parallel}$. Then, $pred(S_i)$ must be pebbled at time t_{i-1}. A_p can thus spend $|S_i|$ pebbling steps to pebble the graph sequentially by adding pebbles on all vertices $v \in S_i$ sequentially until the state of the graph is the same as the state of the graph at time t_i under strategy $A_p^{\parallel}$. Similarly, if a set of pebbles D_i are deleted from the graph at time t_i, then A_p can spend $|D_i|$ sequential pebbling steps to delete $|D_i|$ pebbles. If both strategies start on identical graphs with the same starting configuration S, then we have shown that $Peb^{\parallel}(G_{n,\delta}) \geq Peb(G_{n,\delta})$. Thus, $Peb^{\parallel}(G_{n,\delta}) = Peb(G_{n,\delta})$. □

An immediate application of this result is that all hardness proofs given in Chapters 3-5 immediately transfer to this model since the proofs are hardness based on the minimum number of pebbles used in the pebbling of the construction.

Given the proof of Lemma 6.1, we only need to prove $\text{Peb}(G_{n,\delta})$ in order to show the minimum parallel pebbling space cost of any graph $G_{n,\delta}$.

6.1 Sequential and Parallel Time of Pebbling for Graphs

In this section, we discuss our results in determining the sequential/parallel gap of pebbling given S pebbles as defined in Definition 2.10. We first discuss common graph families used in the black pebbling literature. The goal of this analysis is not only to decrease $\mathsf{Dec}(G_{n,\delta}, S)$ but to also maximize S. Furthermore, we only consider $S = \mathsf{Peb}(G_{n,\delta}) = \mathsf{Peb}^{||}(G_{n,\delta})$ as proven by Lemma 6.1.

We immediately present graphs that establish an upper bound on $\mathsf{Dec}(G_{n,\delta}, S)$ as well as present a simple graph family that meet the lower bound for $\mathsf{Dec}(G_{n,\delta}, S)$ of 1.

First, we prove the simple fact about the parallel pebbling time cost of any graph, $G_{n,\delta}$.

Lemma 6.2. $\mathsf{Time}^{||}(G_{n,\delta}) \geq \Lambda(G_{n,\delta})$ *for any* S *where* $\Lambda(G_{n,\delta})$ *is the longest directed path in* $G_{n,\delta}$.

Proof. No successor may be pebbled in the same timestep as its predecessor as it would violate precedence constraints. Therefore, in order to pebble all nodes in a directed path of length n, we must use n time, proving the lemma. □

Lemma 6.3. $\mathsf{Dec}(G_{n,2}, 1) = 1$ *for lines,* $L_n \in \mathbb{L}_n$ *where* $L_n = \{v_0, \ldots, v_n\}$*, of length* n *where edges are directed from* v_i *to* v_{i+1} *for all* $i \in [n]$.

Proof. By simple observation, at most at most one additional pebble may be placed on L_n at each timestep by precedence constraints as stated in Lemma 6.2. Therefore, $\mathsf{Time}(L_n, 1) = n$ and $\mathsf{Time}^{||}(L_n, 1) = n$ and $\mathsf{Dec}(L_n, 1) = 1$. □

Note that this proof only holds for connected line graphs where all edges are directed in the same direction. For general directions of edges, the ratio is equal to the length of the largest connected component in the line whereby connected component, we mean all pairs of vertices that can be reached from one to another via a directed path.

We further prove an upper bound of $\sqrt{n}$ on $\mathsf{Dec}(G_{n,\delta}, \sqrt{n})$ when $S = \sqrt{n}$.

Lemma 6.4. $\mathsf{Time}(\Pi_n, h) = n$ *and* $\mathsf{Time}^{||}(\Pi_n, h) = h + 1$ *where* h *is the height of* π*.*

Proof. As proven in [Nor15], $\mathsf{Peb}(\Pi_n) = h - 1$ where h is the height of the pyramid. Then, by Lemma 6.1, $\mathsf{Peb}^{||}(\Pi_n) = h$. Because there are n nodes in the pyramid, the minimum time of pebbling is n. Therefore, $\mathsf{Time}(\Pi_n) = n$.

We prove that $\mathsf{Time}^{||}(\Pi_n) = h + 1$. First, we provide a parallel pebbling strategy that shows that $\mathsf{Time}^{||}(\Pi_n) \leq h + 1$. First, the bottom of the pyramid is pebbled using $h - 2$ pebbles. Then, a second turn is needed to pebble the two remaining vertices on the bottom layer that were not pebbled by the $h - 2$ pebbles. After these two steps, we can pebble the remaining portion of the pyramid in h steps. Therefore, $\mathsf{Time}^{||}(\Pi_n) \leq h + 1$.

By Lemma 6.2, we know that $\mathsf{Time}^{||}(\Pi_n) \geq h$. Using this fact, what remains to be shown is that $\mathsf{Time}^{||}(\Pi_n) \geq h + 1$. To do this, we need only to show that the number of step used is $> h$. In order to pebble Π_n using h time, one must have a pebble on all paths of length h in Π_n during all h steps of the pebbling. We show this is impossible.

There exists a total of h distinct paths from the sources to the one sink in Π_n since there exists h sources all of which there exists directed paths to the one sink. Therefore, at time 0, there needs to be h pebbles on all h sources of Π_n. However, at least 1 path cannot have a pebble at time 0. Therefore, $\mathsf{Time}^{||}(\Pi_n) > h$, proving our lemma. □

Lemma 6.5. $\mathsf{Dec}(G_{n,2}, \sqrt{n}) = \sqrt{n}$ *for pyramids,* Π_n*, with* n *total nodes.*

Proof. We proved in Lemma 6.4 that $\mathsf{Time}(\Pi_n) = n$ and $\mathsf{Time}^{||}(\Pi_n) = h+1$. Recall by the definition of the pyramid, $n = \frac{h(h+1)}{2}$. Then, $\mathsf{Dec}(G_{n,2}, \sqrt{n}) \leq \frac{\frac{h(h+1)}{2}}{h+1} = \frac{h}{2} \leq \sqrt{n}$. □

Another common graph used in traditional time/space tradeoff applications are bit-reversal graphs. As summarized in [Nor15], these graphs exhibit a tradeoff of $\mathsf{Time}(G_{n,\delta}, S) = \Theta(n^2/S)$.

While the pyramid graph is easy to analyze, another similar graph is slightly harder to analyze. For a binary search tree with n nodes, B_n, $\mathsf{Peb}(B_n) = \mathsf{Peb}^{||}(B_n) = h$ and $\mathsf{Time}(B_n) = n$ as proven in [adH81]. A trivial upper bound for $\mathsf{Dec}(B_n)$ is $\frac{n}{\log n}$. However, we show instead a tighter upper bound for $\mathsf{Dec}(B_n)$ to be $\log n$. To prove this bound, we show that $\mathsf{Time}^{||}(B_n) \geq n/\log n$.

Lemma 6.6. *For a balanced binary tree with* n *nodes,* B_n*,* $\mathsf{Time}^{||}(B_n) \geq n/\log n$*.*

Proof. There exists n different paths from the sources to the sink of the binary search tree each of length $\log n$. By the same logic as in the proof of Lemma 6.4, for each timestep t_i, we would like to

have a pebble on as many of these n different paths as possible. Suppose without loss of generality, $n = 2^h$ for some h. Then, given $h = \log n$ pebbles, we can pebble at most $\log n$ of the n paths at time t_0. Each additional timestep reduces the number of pebbles used by this first pebbling by half while maintaining the same previous number of paths that contain a pebble. Therefore, at time t_1, $\log n + \frac{\log n}{2}$ paths are pebbled. Therefore, at time t_i, at most $\log n + i\frac{\log n}{2}$ paths are pebbled. In order for all paths to be pebbled, we need at least $n/\log n$ timesteps. □

It follows immediately that

Lemma 6.7. $\mathsf{Dec}(B_n, \log n) \leq \log n$.

We see that thus far, $\mathsf{Dec}(G_{n,\delta}, S) = S$ where $S = \mathsf{Peb}(G_{n,\delta}) = \mathsf{Peb}^{||}(G_{n,\delta})$. However, it is not yet proven whether this is true for all graphs with n nodes and δ indegree. Intuitively, a lower bound of this form would be surprising since it essentially states that the more memory the sequential player and the parallel player have available, the bigger the gap in performance of the sequential and parallel player.

6.1.1 Pebbling Price of Composite Graphs

We define *composite graphs* to be graphs composed of parallel elements of the simple graphs defined earlier in this section.

Specifically, in this section, we look at $\mathsf{Dec}(G_{n,2}, \Theta(\log n))$ of graphs known as superconcentrators provided in [LT82]. However, rather than looking at the superconcentrator with $\Omega(n \log n)$ number of nodes where n is the width of the superconcentrator which has pebbling cost $\mathsf{Peb}(G_{n,\delta}) \leq c \log n$ for some constant c. We look at an equivalent graph composed of n binary search trees as defined below. We choose to use this graph instead of the original superconcentrator graphs since the minimum number of pebbles used in the superconcentrator graph $\leq \log n$.

Definition 6.1. *We define the* composite balanced binary tree *graph to be constructed by the following rules:*

1. *There exists* $\log n + 1$ *levels, where level* $l \in [\log n]$.
2. *Construct edges* $(v_i^l, v_{\lceil \frac{i}{2} \rceil + j}^{l+1})$ *for all* $j \in [0, 2^l, 2 \cdot 2^l, \ldots, k \cdot 2^l, \ldots, (\frac{n}{2} - 1) \cdot 2^l]$ *where* v_i^l *is the* i*-th vertex in level* l.

3. *Construct edges* $(v_i^{\log n}, v_i^{\log n+1})$.

4. *Construct edges* $(v_i^{\log n+1}, v_{i+1}^{\log n+1})$ *for all* $i \in [n]$.

Lemma 6.8. $\mathsf{Peb}(CB_n) = \log n + 1$ *and* $\mathsf{Time}(CB_n) \leq n(n+1)$ *where* CB_n *is a composite balanced binary tree as defined in Definition 6.1 with* n *nodes.*

Proof. From [adH81], $\mathsf{Peb}(B_n) = \log n$ and $\mathsf{Time}(CB_n) = n$. We first prove that $\mathsf{Peb}(CB_n) = \log n + 1$. $v_0^{\log n}$ is the apex of a balanced binary search tree. Therefore, to pebble this node alone needs $\log n$ pebbles. An additional pebble is needed to pebble the nodes in level $\log n + 1$. Since any two consecutive trees are disjoint at least down to level $\frac{\log n}{2}$, the pebbles must be removed from the first tree and used to pebble the next tree. Therefore, all $\log n$ pebbles must be used to pebble the next tree and another pebble must remain in the nodes on level $\log n + 1$ resulting in $\log n + 1$ pebbles.

Given $\log n$ pebbles, we can pebble the binary trees one by one while maintaining a pebble on the $\log n + 1$-th level. This results in a sequential pebbling time of $n(n + 1)$. □

Lemma 6.9. $\mathsf{Peb}^{\|}(CB_n) = \log n + 1$ *and* $\mathsf{Time}(CB_n) \geq n(n/\log n + 1)$.

Proof. The proof of $\mathsf{Peb}^{\|}(CB_n)$ follows from the Lemmas 6.6 and 6.8. To prove that $\mathsf{Time}(CB_n) \geq n(n/\log n + 1)$, we make the observation that all pebbles from the balanced binary search tree with target node v_i must be removed in order to pebble the tree with target node v_{i+1} since consecutive trees have disjoint nodes from levels $\log n$ down to level $\frac{\log n}{2}$. A simple analysis from this observation leads to $\mathsf{Time}(CB_n) \geq n(n/\log n + 1)$. □

Lemma 6.10. $\mathsf{Dec}(CB_n) \leq \log n + 1$.

Proof. From Lemmas 6.8 and 6.9, we obtain that $\mathsf{Dec}(CB_n) \leq \frac{n(n+1)}{n(n/\log n+1)} \leq \log n + 1$. □

We see that with the composite balanced binary tree, the ratio is approximately the same to a negligible additive factor. Therefore, in this respect, it seems that both constructions are equally "good" in terms of the decremental complexity of the graph.

However, as is often the case in cryptography, attackers often have less space than the honest party with which to conduct their attack. Suppose instead that an honest part has space n to perform their computation, which the adversary only has space $\log n$ to perform their computation.

Then, in this model, the decremental complexity is $\frac{n(\log n+1)}{n(\log n+1)} = 1$. This is the same decremental complexity as a line but requires the adversary to have more than $\Theta(1)$ space available to conduct the attack. Therefore, the concept of composite graphs is useful in certain attack models.

Although it is interesting to know the time tradeoffs of switching from the sequential pebbling model to the parallel pebbling model for common DAG families and simple constructions of graphs, it is even more interesting to consider the parallel pebbling time of graphs where $\mathsf{Peb}(G_{n,\delta}) = \mathsf{Peb}^{||}(G_{n,\delta}) = \Omega(n/\log n)$ since the maximum number of pebbles needed to pebble any graph is $O(n/\log n)$ [HPV77b]. A natural candidate for this analysis is then the stacked superconcentrator graphs provided by [LT82]. Although these graphs are not analyzed in this thesis, we are interested in analyzing these graphs in future work. It would also be interesting to determine lower bounds for $\mathsf{Dec}(G_{n,\delta}, S)$ for various values of S. Furthermore, it would be interesting to determine whether the ratio is tight specifically for $\mathsf{Peb}(G_{n,2}) = \mathsf{Peb}^{||}(G_{n,2}) = \Omega(n/\log n)$ is tight or whether we could improve the argument by using a more intricate analysis than the one provided above.

Chapter 7

Other Pebbling Results

There are a number of hardness results that directly follow (or with some very minor adjustments) from the results provided in the literature. In this chapter, we summarize these results and provide more details in the cases where additional details and proofs are necessary.

7.1 One-Shot Standard Pebbling is Fixed-Parameter Tractable

Austrin, Pitassi, and Wu [APW12] previously showed the equivalence between the one-shot standard (or black) pebbling problem and a problem known as *register sufficiency*. The formal definition of register sufficiency is given in Definition 7.1.

Definition 7.1 (Register Sufficiency)**.** *Let $G = (V, E)$ be a directed acyclic graph, find a topological ordering of the vertices of the graph, π, to minimize the following quantity:*

$$\min_{\pi} \max_{i \in [n]} |V_i(\pi)|.$$

Here, $V_i(\pi)$ is the number of predecessors (i.e. nodes that edges point from) of edges that cross the location i in the topological ordering π.

Given the equivalence between these two problems as provided by [APW12], if there exists a fixed-parameter tractable algorithm for register sufficiency, then there also exists a FPT algorithm for the one-shot pebbling problem. Register sufficiency has been shown to be fixed-parameter tractable [BFH94]; therefore, one-shot standard pebbling is also fixed-parameter tractable.

Theorem 7.1. *One-shot pebbling problem is fixed parameter tractable when parameterized by the number of pebbles, k.*

7.2 A Graph Family that Requires $\Omega((\frac{n-k+1}{2k})^k)$ time to pebble for $k < \sqrt{n}$

It is common folklore in the pebbling community that there exists families of graphs that take $\Theta(n^k)$ time to pebble sequentially given k pebbles (see Section 7.3 for proof). However, to the best of the author's knowledge, examples of such families of graphs do not exist in the literature. In this section, we present a simple to construct family of graphs that require $\Theta(n^k)$ time for constant k number of pebbles. We further reduce the indegree of nodes in this family of graphs to 2 and show that our results still hold. Such families of graphs could potentially have useful applications in cryptography in the domain of proofs of space and memory-hard functions.

We construct the following family of graphs, $\mathbb{G}_{k,n,k}$, below with indegree k and show that for constant k, the number of steps it takes to pebble the graph $G_{k,n} \in \mathbb{G}_{k,n}$ with k pebbles and n nodes is $O(n^k)$ for constant k. We also show a family of graphs, $\mathbb{G}_{k,n,2}$ with indegree 2 that shows the same asymptotic tradeoff.

We construct the family of graphs in the following way.

Definition 7.2. *Given a set of n nodes and maximum number of pebbles k where $k < \sqrt{n}$, we lexicographically order the nodes (from 0 to $n-1$) and create the following set of edges between the nodes where directed edges are directed from v_i to v_j where $i < j$:*

1. *v_i and v_{i+1} for all $i \in [k-1, n-1]$*
2. *v_i and v_j for all $i \in [0, l-1]$ for all $1 \leq l \leq k-1$ and $j \in \{f(q) + 2i - 1\}$ for all $i \in [\frac{n-k+1}{2k}]$ where $f(q) = k - 1 + (q-1)(\frac{n-k+1}{k})$ where $q \in [2, k]$*
3. *v_i and v_j for all $i = f(q)$ and $j \in \{f(q) + 2i\}$ for all $i \in [\frac{n-k+1}{2k}]$ where $q \in [2, k]$.*

The target node (the only sink) is v_{n-1}. Note that the sources in our construction are v_0 and v_j for all $j \in [0, p(k)]$.

We now prove the time bound for this family of graphs $\mathbb{G}_{k,n,k}$ for all $n \in \mathbb{R}$ and $k < \sqrt{n}$.

To prove the minimum number of pebbles necessary to pebble the graph, it is sufficient to study the number of blocked paths in any graph $G_{k,n,k} \in \mathbb{G}_{k,n,k}$[Nor15]. We define *blocking* as in [Nor15].

Definition 7.3 (Blocking [Nor15])**.** *A set of vertices, U, blocks a path, P, if $U \cap P \neq \emptyset$. U blocks a set of paths $\mathbb{P}$ if U blocks P for all $P \in \mathbb{P}$.*

Lemma 7.1. *The minimum number of pebbles necessary to pebble $G_{k,n,k} \in \mathbb{G}_{k,n,k}$ is k.*

Proof. The degree of the graph $G_{k,n,k}$ is k, therefore, at least k pebbles are necessary to pebble $G_{k,n,k}$. □

Theorem 7.2 ($G_{k,n,k}$ moves bound)**.** *The number of moves necessary to pebble $G_{k,n,k}$ is $\Theta((\frac{n-k+1}{2k})^k)$ for $k < \sqrt{n}$.*

Proof. Suppose that at time t, there exists k paths which are not blocked by any of the pebbles placed on the graph. By Lemma 7.1 and the proof of blocking paths in [Nor15], we know t must exist at some point in the pebbling of $G_{k,n,k}$. Let the first degree k node to be pebbled be v_i. Then, $|p(v_i)| = k$ and there exists k blocking paths from each $u \in p(v_i)$ as its source. Placing a pebble on each of its predecessors blocks all k paths. After v_i has been pebbled, by Definition 7.2, there are two different paths going from the sources through v_{i+1}. However, none of the pebbles placed at time $t \leq t' \leq t_{now}$ blocks the path from v_0 to v_{n-1}. Therefore, a pebble must be placed on this path to block it. The resulting number of moves is $T(k) = \frac{n-k+1}{2k}T(k-1) + k(\frac{n-k+1}{2k})$ by induction on our argument above with base case $T(1) = \frac{n-k+1}{k}$. Therefore,

$$T(k) = (\frac{n-k+1}{2k})^k + \sum_{i=0}^{k-1}(k-i)(\frac{n-k+1}{2k})^{i+1} \geq 2(\frac{n-k+1}{2k})^k = \Theta((\frac{n-k+1}{2k})^k). \tag{7.1}$$

□

This result partially answers an open question posed in [Nor15] whether a family of graphs can have a number of moves, $\Omega(n^k)$, that meets the upper bound for constant k number of pebbles. We believe that our graph family also meets the upper bound for the black-white pebbling case. We are interested in resolving this question in future work as that would completely resolve the open question.

7.3 Standard, Red-Blue, and Reversible Pebbling are in XP

We provide a short proof that standard, red-blue, and reversible pebbling are in APX. We showed in Chapter 4 that the red-blue game when parameterized by the number of transitions is W[1]-hard. Here we show that none of the pebbling games mentioned can be APX-hard.

Theorem 7.3. *Standard, red-blue, and reversible pebbling are in the complexity class, XP.*

Proof. We provide a simple $O(n^k)$ algorithm for solving any of the pebbling problems mentioned in this lemma. Given a graph $G = (V, E)$ with n nodes and m edges, there are $\binom{n}{k}$ different configurations of pebble placements on nodes of the graph where k is the number of black or red pebbles depending on the problem. We now create a digraph containing nodes each of which is a different configuration of the pebbles on the graph.

Then, we create edges between nodes if there is a legal transition from one node to the next. Then, we find all nodes that are legal terminal nodes (i.e. the end nodes are ones that exhibit legal termination configurations). Then, we run a polynomial time single-source shortest path algorithm from the starting node which consists of the blank graph. Therefore, the running time is polynomial in terms of n^k. □

Chapter 8

Open Questions

There are a number of open questions that remain as a result of this thesis. Here we summarize this open questions by the section of the thesis they would appear in:

8.1 Standard and Red-Blue Pebble Games Hardness

- Is the standard pebble game is fixed-parameter tractable or hard even in the fixed-parameter sense when parameterized by the number of pebbles and the case when the number of moves is restricted?
- Can the standard pebble game or the red-blue pebble game be approximated to any factor smaller than $n/\log n$.
- Furthermore, are the games PSPACE-hard to approximate to any constant multiplicative factor?

8.2 Parallel Pebbling Model and Decremental Complexity

- What is the lower bound on the decremental complexity for any family of graphs using $S = \Theta(n/\log n)$ pebbles?
- What is the decremental complexity of stacked superconcentrators?

8.3 Space-Time Tradeoffs

- Do the bounds stated in Chapter 7 also hold in the black-white pebbling model?

Bibliography

[AB16a] Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 241–271, 2016.

[AB16b] Joël Alwen and Jeremiah Blocki. Towards practical attacks on argon2i and balloon hashing. *IACR Cryptology ePrint Archive*, 2016:759, 2016.

[ABP16] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. *IACR Cryptology ePrint Archive*, 2016:875, 2016.

[ACK+16] Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. On the complexity of scrypt and proofs of space in the parallel random oracle model. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 358–387, 2016.

[adH81] Friedhelm Meyer auf der Heide. A comparison of two variations of a pebble game on graphs. *Theoretical Computer Science*, 13(3):315 – 322, 1981.

[AGK+16] Joël Alwen, Peter Gazi, Chethan Kamath, Karen Klein, Georg Osang, Krzysztof Pietrzak, Leonid Reyzin, Michal Rolínek, and Michal Rybár. On the memory-hardness of data-independent password-hashing functions. *IACR Cryptology ePrint Archive*, 2016:783, 2016.

[APW12] Per Austrin, Toniann Pitassi, and Yu Wu. Inapproximability of treewidth, one-shot pebbling, and related layout problems. In *Approximation, Randomization, and Combinato-*

rial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings, pages 13–24, 2012.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 595–603, 2015.

[AV88] Alok Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, September 1988.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, August 1989.

[BFH94] Hans L. Bodlaender, Michael R. Fellows, and Michael T. Hallett. Beyond np-completeness for problems of bounded width (extended abstract): Hardness for the w hierarchy. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 449–458, New York, NY, USA, 1994. ACM.

[BZ16] Jeremiah Blocki and Samson Zhou. On the computational complexity of minimal cumulative cost graph pebbling. *CoRR*, abs/1609.04449, 2016.

[Cha13] Siu Man Chan. Just a pebble game. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:42, 2013.

[CLNV15] Siu Man Chan, Massimo Lauria, Jakob Nordström, and Marc Vinyals. Hardness of approximation in PSPACE and separation results for pebble games. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 466–485, 2015.

[Coo73] Stephen A. Cook. An observation on time-storage trade off. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 29–33, New York, NY, USA, 1973. ACM.

[CRSS16] Timothy Carpenter, Fabrice Rastello, P. Sadayappan, and Anastasios Sidiropoulos. Brief announcement: Approximating the I/O complexity of one-shot red-blue pebbling. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 161–163, 2016.

[CS74] Stephen Cook and Ravi Sethi. Storage requirements for deterministic / polynomial time recognizable languages. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 33–39, New York, NY, USA, 1974. ACM.

[DF95] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theor. Comput. Sci.*, 141(1-2):109–131, April 1995.

[DT85] Patrick W. Dymond and Martin Tompa. Speedups of deterministic machines by synchronous parallel machines. *Journal of Computer and System Sciences*, 30(2):149 – 161, 1985.

[EBL79] Peter Emde Boas and Jan Leeuwen. *Theoretical Computer Science 4th GI Conference: Aachen, March 26–28, 1979*, chapter Move rules and trade-offs in the pebble game, pages 101–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 1979.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[GLT79] John R. Gilbert, Thomas Lengauer, and Robert Endre Tarjan. The pebbling problem is complete in polynomial space. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 237–248, New York, NY, USA, 1979. ACM.

[GT78] John R. Gilbert and Robert E Tarjan. Variations of a pebble game on graphs. Technical report, Stanford, CA, USA, 1978.

[HP10] Philipp Hertel and Toniann Pitassi. The PSPACE-completeness of black-white pebbling. *SIAM J. Comput.*, 39(6):2622–2682, April 2010.

[HPV77a] John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *J. ACM*, 24(2):332–337, April 1977.

[HPV77b] John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *J. ACM*, 24(2):332–337, April 1977.

[JWK81] Hong Jia-Wei and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 326–333, New York, NY, USA, 1981. ACM.

[LT82] Thomas Lengauer and Robert E. Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *J. ACM*, 29(4):1087–1130, October 1982.

[Nor15] Jakob Nordstrom. New wine into old wineskins: A survey of some pebbling classics with supplemental results. 2015.

[PH70] Michael S. Paterson and Carl E. Hewitt. Record of the project mac conference on concurrent systems and parallel computation. chapter Comparative Schematology, pages 119–127. ACM, New York, NY, USA, 1970.

[PT78] W. J. Paul and R. E. Tarjan. Time-space trade-offs in a pebble game. *Acta Inf.*, 10(2):111–115, June 1978.

[PTC76] Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, pages 149–160, New York, NY, USA, 1976. ACM.

[Set75] Ravi Sethi. Complete register allocation problems. *SIAM J. Comput.*, 4(3):226–248, 1975.