Red-Blue Pebble Game: Complexity of Computing the Trade-Off between Cache Size and Memory Transfers

Erik D. Demaine
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
edemaine@mit.edu

Quanquan C. Liu
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
quanquan@mit.edu

ABSTRACT

The red-blue pebble game was formulated in the 1980s [14] to model the I/O complexity of algorithms on a two-level memory hierarchy. Given a directed acyclic graph representing computations (vertices) and their dependencies (edges), the red-blue pebble game allows sequentially adding, removing, and recoloring red or blue pebbles according to a few rules, where red pebbles represent data in cache (fast memory) and blue pebbles represent data on disk (slow, external memory). Specifically, a vertex can be newly pebbled red if and only if all of its predecessors currently have a red pebble; pebbles can always be removed; and pebbles can be recolored between red and blue (corresponding to reading or writing data between disk and cache, also called I/Os or memory transfers). Given an upper bound on the number of red pebbles at any time (the cache size), the goal is to compute a game execution with the fewest pebble recolorings (memory transfers) that finish with pebbles on a specified subset of nodes (outputs get computed).

In this paper, we investigate the complexity of computing this trade-off between red-pebble limit (cache size) and number of recolorings (memory transfers) in general DAGs. First we prove this problem PSPACE-complete through an extension of the proof PSPACE-hardness of black pebbling complexity [13]. Second, we consider a natural restriction on the red-blue pebble game to forbid pebble deletions, or equivalently, forbid discarding data from cache without first writing it to disk. This assumption both simplifies the model and immediately places the trade-off computation problem within NP. Unfortunately, we show that even this restricted version is NP-complete. Finally, we show that the trade-off problem parameterized by the number of transitions is W[1]-hard, meaning that there is likely no algorithm running in a fixed polynomial for constant number of transitions.

KEYWORDS

red-blue pebble game; external memory model; computational complexity

SPAA '18, July 16–18, 2018, Vienna, Austria

ACM ISBN 978-1-4503-5799-9/18/07…$15.00
https://doi.org/10.1145/3210377.3210387

ACM Reference Format:
Erik D. Demaine and Quanquan C. Liu. 2018. Red-Blue Pebble Game: Complexity of Computing the Trade-Off between Cache Size and Memory Transfers. In *SPAA '18: 30th ACM Symposium on Parallelism in Algorithms and Architectures, July 16–18, 2018, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3210377.3210387

1 INTRODUCTION

Pebble games were originally introduced to study compiler operations and programming languages. One example of such an application is when a directed acyclic graph (DAG) represents the computational dependencies between operations and the pebbles represent register or memory allocation. Minimizing the resources allocated to perform a computation is accomplished by minimizing the number of pebbles placed on the graph [17], and the time of computation is modeled by the number of pebbles moves the player makes in a strategy that ultimately pebbles the desired output vertices. In addition to the standard pebble game (also known as the black pebble game in the literature) that models register or memory allocation, there are several other pebble games that are useful for studying computation. The *red-blue pebble game* is used to study I/O complexity [14], the *reversible pebble game* is used to model reversible computation [2], and the *black-white pebble game* is used to model non-deterministic straight-line programs [5].

In this paper, we study the *red-blue pebble game* used to model the cost of programs in a two-level memory hierarchy. The two-level memory hierarchy model of [14] and the blocked version, the I/O model or external memory model [1], were introduced in the 1980s [1, 14] to capture the computational bottleneck of transferring data between a large but slow disk and a small but fast cache. (See [7] for more about the history.) The *red-blue pebble game* models the number of such data transitions that are necessary between cache and disk, as well as the limit of the cache size. The rules of the game are as follows:

Red-Blue Pebble Game [14]. Given a DAG $G = (V, E)$, the game works as follows:

(1) At the start, every source node has a blue pebble, and no other nodes have pebbles.
(2) The player can place a red pebble on a node if and only if all of its predecessors are currently pebbled with red pebbles.
(3) The player can recolor a blue pebble to red, or a red pebble to blue.
(4) The player can delete a pebble from a node at any time.

Goal: Pebble all sink nodes with blue pebbles.

Red pebbles represent data in cache and blue pebbles represent data in disk. We suppose (as in real systems) that there is a limited amount of cache and an unlimited amount of disk. In terms of the red-blue pebble game, this means that the number of red pebbles is limited by some upper bound r. Subject to this constraint, the goal is to pebble all target nodes while minimizing the number k of pebble recolorings between red and blue, i.e., minimizing the number k of memory transfers between cache and disk. Recoloring a pebble from red to blue corresponds to writing data out from cache to disk, while recoloring from blue to red corresponds to reading data in from disk to cache.

Much previous research has focused on proving lower and upper bounds on the pebbling cost (i.e., the number of pebbles and/or transitions used) of pebbling a given family of DAGs under the rules of the red-blue pebble game. Such upper and lower bounds are computed with respect to the number of transitions given a number of red pebbles, r, that are provided to pebble the graph. Such previous results include upper and lower bounds of $O(n \log n / \log r)$ and $\Omega(n \log n / \log r)$, respectively, on the minimum number of transitions needed to pebble an FFT graph. Upper and lower bounds on number of transitions for other graph classes such as r-pyramids, diamond graphs, butterfly graphs, and matrix multiplication graphs can be found in [11, 14, 16]. More recently, the model of one-shot red-blue pebble games was introduced in [3]. 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.

Despite somewhat extensive research on the upper and lower bounds of optimally pebbling a DAG in pebble 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 a constant or logarithmic multiplicative approximation factor [4, 8]. 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 to within an additive $n^{1/3-\varepsilon}$ term is PSPACE-complete [8] and finding the minimum number of black pebbles needed in the one-shot case is NP-complete [17]. 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 using the red-blue pebble game, 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.

The purpose of this paper is twofold. We seek to answer the question of computational complexity of finding the optimal number of red pebbles and minimum number of transitions used. Secondly, we seek to motivate the study of finding lower bounds and optimal pebbling strategies for certain graph classes by showing that such hardness results even hold for the very restricted class of layered graphs. In this paper, we discuss the following new results:

(1) In Section 2, we show that the red-blue pebble game is PSPACE-complete, via a simple extension of the result of [13].
(2) In Section 3, we introduce a new red-blue pebble game and prove it NP-complete. Specifically, we consider the natural restriction to when all data must be kept somewhere, either in cache or on disk, or equivalently, the player cannot delete pebbles, only recolor them.
(3) In Section 4, we analyze the complexity of the red-blue pebble game when parameterized by the allowed number t of transitions. We prove that this problem is W[1]-hard, so it does not have a fixed-parameter algorithm (with running time $f(t)n^{O(1)}$) unless FPT = $W[1]$. (Note that some PSPACE-complete problems are fixed-parameter tractable, so this result is not implied by our other results.)

Due to space constraints, all proofs which are not given in the main body of the paper can be found in the full version of our paper.

2 RED-BLUE PEBBLE GAME

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 [13] (and is almost identical to the proof provided there). Therefore we include this result first before our main result on red-blue pebbling *without deletions* whose proof we expand upon in more detail in the next section.

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 (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 obtained from disk and all outputs are written back to 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). Furthermore, for the *red-blue start-in-cache* game, we assume that all targets must be computed at least once in cache without the need to write the results back into disk (i.e. this is also known as a visiting pebbling in cache [15] in that the red pebbles do not need to be turned into blue pebbles even if they are deleted 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 red pebble space usage r and minimum number of transitions k can be converted to a DAG G' with minimum pebbling space usage $r+1$ and number of transitions $k+1+|T|$ (where T is the target set or output nodes) using the rules of the red-blue start-in-disk game.

Theorem 1 (Red-Blue Disk to Cache). *Given a DAG, $G = (V, E)$ with target set T and bounded in-degree 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 $k + 1 + |T|$ 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 in-degree up to 3. In the final step of creating G', we replace all vertices with in-degree 3 with pyramids of height 3. Note that a pyramid of height 3 functions in the same manner as a node with in-degree 3 by normality of pebbling strategies proven in [13, 15].

Let $\mathcal{P}$ 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 $k + 1 + |T|$ 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 $\mathcal{P}$ to pebble the remaining nodes of G'.

We now consider the minimum number of transitions that can be used with at most $r + 1$ red pebbles and before all outputs are written back to disk. We 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 $\mathcal{P}$ uses a minimum of k transitions and turning the pebble on u to red requires 1 transition). Thus, given that the red pebble remains on the source during the entire pebbling of G' (i.e. the red pebble on the source u is present at time $t = \text{argmax}_{P_t \in \mathcal{P}} \{|P_t|\}$) where the maximum number of red pebbles are on the graph using strategy $\mathcal{P}$) 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 before all outputs are written back into disk.

Finally, given the set T of targets, one transition must be spent to turn a red pebble into a blue pebble on each $v_T \in T$. Thus, at least $|T|$ transitions must be spent in this case, resulting in a total of at least $k + 1 + |T|$ transitions. □

In the remaining sections of the paper, we prove all results with respect to the rules of the red-blue start-in-cache game, even if we do not explicitly state that we do so. Note that using Theorem 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.

The red-blue start-in-cache pebble game as defined above is PSPACE-hard as a simple extension of the proof given in [13]. The formal definition of the problem is given below.

Definition 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 start-in-cache pebbling rules as defined 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.*

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 [13]. 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. [13] 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 [13]. We show that 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 [13] can be pebbled using the rules of the standard pebble game; hence the red-blue pebble game is PSPACE-complete.

Lemma 1. *The proof construction provided in [13] 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 start-in-cache pebble game.*

Theorem 2. *Determining the minimum pebbling cost and number of transitions is PSPACE-complete (even given constant number of transitions) to compute in the red-blue pebble game.*

By noticing that we can transform the above hardness construction to a layered graph by topologically sorting the vertices used in the construction and replacing each edge that go between non-consecutive layers by a path of length equal to the number of layers that the edge go between, we result in a multiplicative factor increase of at most $O(n^2)$ in the number of nodes in the graph since the number of layers is at most $O(n)$. It is trivial to see that all pebbling constraints are still preserved by replacing edges with paths.

Corollary 1. *Determining the minimum pebbling cost and number of transitions is PSPACE-complete to compute in the red-blue pebble game even for layered graphs.*

3 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 without using any additional transitions, i.e. the vertex already contains a blue pebble.)
(4) No pebbles can be deleted from a vertex.

As usual, red pebbles represent fast memory and blue pebbles represent slow memory; we assume that we have infinitely large slow memory, but only a bounded fast memory. The goal of this

game is to pebble all vertices in G while minimizing the number of *transition* moves. 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 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 at most k red-blue transitions are used.*

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

3.1 Proof Overview

We provide a reduction broadly similar in concept to [13] 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 (Positive 1-in-3 SAT [12]). *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 literal truth 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.

The pebbling of the gadgets occur in two phases: Phase 1 and Phase 2. During Phase 1 of the pebbling, all variables gadgets are set to a truth value. During Phase 2 of the pebbling, the portions of the variable gadgets that were not pebbled in Phase 1, leading to the pebbling of the target node. The gadgets are connected in the following way: all variable gadgets are connected to a pebble sink path (see g_i, g_i', and g_i'' in Fig. 2) where the end of the path is connected to the first clause gadget; all clause gadgets are connected via a path and the last clause is connected to another pebble sink path (see s_i, s_i', and s_i'' in Fig. 2) which is connected to the target node. Each variable gadget is also connected to the clause it appears in as well as the corresponding anti-clause. We define the pebbling of the clauses and anti-clauses to be the *clause verification phase.*

3.2 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 [13] that is used to "trap" a certain minimum number of pebbles that must be used to pebble 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) [6]. Let Π_h be a pyramid graph with height h. It was proven in [15] 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 3.3.

Lemma 2. *Given a pyramid graph of height h, the $PebRBD(\Pi_h)$ is $r_{\Pi_h} = h + 1$ and $t_{\Pi_h} = \frac{h(h+1)}{2}$ such that no red pebbles remain on the pyramid at the end of the pebbling.*

Fig. 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 3.3. 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.

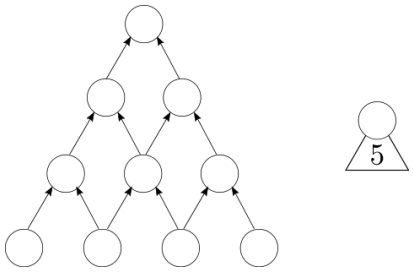

Figure 1: Example of a pyramid gadget with $r_{\Pi_4} = 5$ and $t_{\Pi_4} = 10$.

Using the pyramid gadget we can construct the variable gadget as in Fig. 2.

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. Since each q_i is part of two variable gadgets, we define q_{i-1} to be

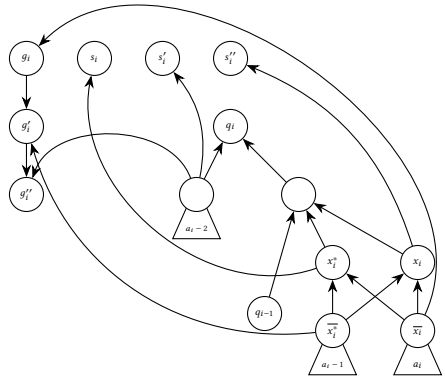

Figure 2: Example of a variable gadget, x_i, with pyramid costs a_i, pebble sink path connections g_i, g_i', and g_i''. The corresponding pebble sinks that correspond with this gadget are s_i, s_i', and s_i''.

part of variable gadget x_i and q_i to be part of the next variable gadget. We show that 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. The apex of these pyramids are connected to *pyramid sink paths* which are paths of length n occuring after each set of clause and two anti-clauses triples. Then, either the pair of nodes x_i and x_i^* is 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. We show that 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 s_i, s_i', and s_i'' since no transitions are allowed to be spent on these nodes.

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 in Fig. 3. The order of the clauses is determined arbitrarily and all clauses are duplicated and occur in the same topological order in the two sets (the original clauses and the duplicate clauses).

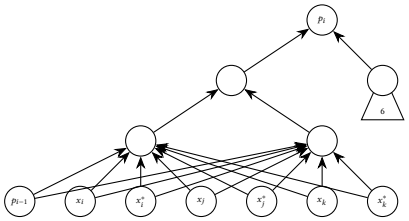

Figure 3: Example of a clause gadget with $r_{c_i} = 2$ and $t_{c_i} = 29$ for clause $c_i = (x_i \vee x_j \vee x_k)$. The number of red pebbles that is needed to fill this gadget is 6 (excluding the two red pebbles that are present on the true literal and the red pebble on p_{i-1}).

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. 5.

Each variable gadget is connected to a *pebble sink path* as shown in Fig. 4. 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.*

We now prove our claims above more formally. We begin by proving that the end of Phase 1, at most 3 red pebbles can remain on each of the variable gadgets.

Lemma 3. *At most* $3n$ *red pebbles can remain on the variable gadgets at the end of Phase* 1 *and the beginning of the clause verification phase where* n *is the number of variable gadgets.*

We prove a matching lower bound for the number of red pebbles that should remain on the variable gadgets after Phase 1.

Lemma 4. *At least* 3 *red pebbles must remain on each variable gadget at the end of Phase* 1 *in order to be able to pebble the constructed graph using a minimum number of red pebbles and transitions.*

Using Lemma 3, we can now prove the number of red pebbles and transitions necessary to pebble the variable gadgets.

Lemma 5. *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* $t_{x_i} =$

$\sum_{j=1}^{3} \frac{(a_i-j+1)(a_i-j)}{2} + 4$ *assuming all red pebbles are removed from the gadget at the end of the pebbling. The transitions cost includes the cost of pebbling and removing red pebbles from all nodes as shown in Fig. 2 except* $s_i, s_i', s_i'', g_i, g_i', g_i''$ *and* q_i*. Assuming all variable gadgets are set to a truth value during Phase* 1 *and Phase* 2 *begins with the pebbling of* q_0 *and ends with red pebbles on* s_i, s_i', s_i'', 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* $5 \leq t_{x_i} \leq 7$ *and red pebbles remain only on* s_i, s_i', s_i''*, and* q_i*.*

We prove the following lemma to help us prove that all variable gadgets must be set in Phase 1.

Lemma 6. *If* n' *variables do 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.*

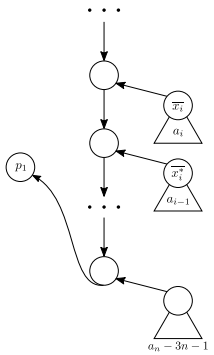

Figure 4: Example pebble sink path. Each node is connected to the root of each pyramid in each variable gadget.

Lemma 7. *Given* $r_{c_i} = 6$ *and* $t_{c_i} = 29$*, the clause gadget* c_i *(shown in Fig. 3) cannot be pebbled if all three variable gadgets incident on* c_i *are in the false configuration and no red pebbles remain on* c_i *after it is pebbled (i.e. when* p_i *is pebbled).*

Lemma 8. *Given* $r_{\overline{c_i}} = 6$ *and* $t_{\overline{c_i}} = 61$*, the anti-clause gadget cannot be pebbled if less than* 2 *negative literals are true.*

Lemma 9. *Each clause gadget must contain exactly one true literal pebbled with red pebbles and each anti-clause gadget must contain exactly two true literals pebbled with red pebbles at the end of Phase 1 before the clause verification phase.*

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

3.3 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. To see

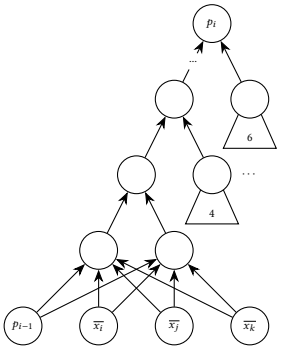

Figure 5: Example of an anti-clause gadget with $r = 6$ **and** $t = 64$**. The number of red pebbles that is needed to fill this gadget is** 6 **(excluding the two red pebbles that are present on the true negative literals).**

an example full construction, please refer to the full version of our paper. .

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 10. *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* $O(n^2)$*.*

Theorem 3. *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.*

Corollary 2. *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.*

One can produce a reduction with indegree 2 graphs by replacing all nodes with indegree greater than 2 with pyramids with height equal to the indegree minus 1. Similar proofs can prove hardness in this case.

4 RED-BLUE PEBBLE GAME PARAMETERIZED BY TRANSITIONS

In this section, we prove that the red-blue pebble game with deletion where the number of red-to-blue or blue-to-red transitions

is parameterized by k is W[1]-hard by reduction from the W[1]-complete problem, Weighted q-CNF Satisfiability. It was previously shown by [9] 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 Section 2. However, we note that an NP-hardness result does not necessarily supersede a parameterized complexity result since they are different complexity domains. In fact, there exist NP-complete problems with natural parameterizations that have fixed-parameter tractable algorithms. Furthermore, the techniques presented in this section are techniques that could be important for future proofs of hardness or hard-to-pebble graph family constructions.

Definition 4 (Weighted q-CNF Satisfiability [9, 10]). *Given a CNF formula, ϕ, a set U of variables ($n = |U|$), and a set C of clauses ($m = |C|$) 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 both clauses must be true. The presence of these clauses is to ensure that each of the variable gadgets in the reduction are assigned a valid 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 Section 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 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 gadgets to be set to false, then picks exactly k variables to set to true and uses exactly $2k$ transitions. The problem is parameterized by the number of transitions, $2k$, 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. 6.

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.

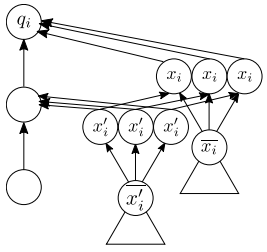

Figure 6: Variable gadget.

Lemma 11. *All variable gadgets must be in the false configuration after Phase 1.*

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.

4.2.2 All False Gadget. The All False gadget is used to check that all variables are initially set to false. See Fig. 7. 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.

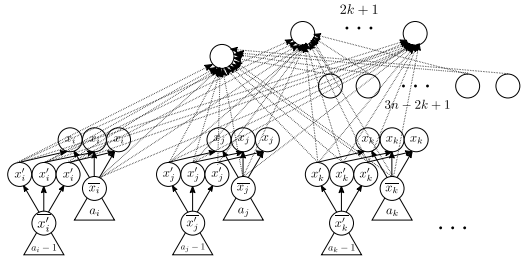

Figure 7: The All False gadget consists of $2k+1$ nodes that all have x'_i and $\overline{x_i}$ as predecessors. Each of these $2k+1$ nodes are connected to the k-True-Variables gadget and the clause gadgets.

4.2.3 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. 8 for an example.

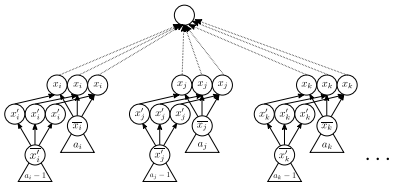

Figure 8: k-True gadget connects to all x_i for all i.

Lemma 12. *At the end of Phase 2, exactly k variable gadgets are set in the true configuration.*

4.2.4 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. 9 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 15. 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.

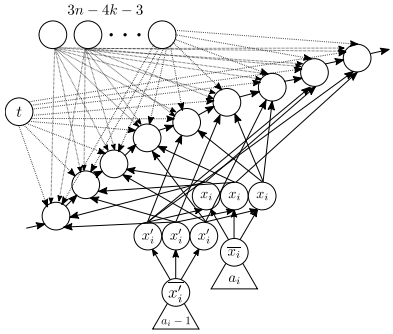

Figure 9: 3-or-None gadget. One is created for every variable.

Lemma 13. *For every variable gadget that does not follow the condition specified in Lemma 15 and contains less than 6 red pebbles, at least two transitions are required to satisfy the gadget.*

Lemma 14. *Suppose some variable gadgets are set in the configuration where $\overline{x_i}$ and $\overline{x'_i}$ are pebbled with red pebbles, then the number of transitions needed to pebble both the All-False gadget and the 3-or-None gadget is greater than $2k$.*

Lemma 15. *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 x'_i.*

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

4.2.5 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. 10 for an example.

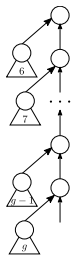

Figure 10: Pebble sink that captures $3n-4k-6$ pebbles leaving 5 pebbles to be used in the clauses. Here $g = 3n-4k-1$.

4.2.6 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. 11.

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

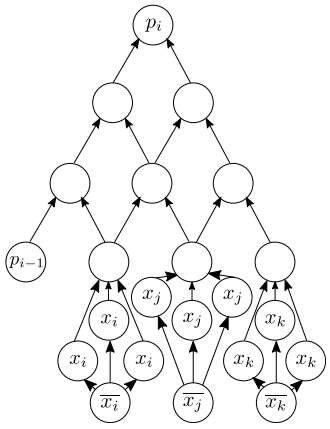

Figure 11: Clause gadget.

For an example reduction, see Fig. 12. The target vertex that must be pebbled is the one colored blue.

Lemma 16. *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.*

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. 12. 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. 12 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. *The red-blue pebble game parameterized by the number of transitions k is W[1]-hard.*

5 OPEN PROBLEMS

We conclude this paper with several open questions:

(1) Are red pebbling number and minimum number of transitions hard to approximate?
(2) Does there exist FPT algorithms for restricted class of graphs (such as bounded width graphs)?
(3) Is finding the red pebbling number W[1]-hard? Recall in this paper that we proved W[1]-hardness only for transitions, not for number of red pebbles.

Acknowledgements. This research was supported in part by NSF Graduate Research Fellowship grant 1122374.

REFERENCES

[1] Alok Aggarwal and S. Vitter, Jeffrey. 1988. The Input/Output Complexity of Sorting and Related Problems. *Commun. ACM* 31, 9 (Sept. 1988), 1116–1127. https://doi.org/10.1145/48529.48535
[2] Charles H. Bennett. 1989. Time/Space Trade-offs for Reversible Computation. *SIAM J. Comput.* 18, 4 (Aug. 1989), 766–776. https://doi.org/10.1137/0218053
[3] Timothy Carpenter, Fabrice Rastello, P. Sadayappan, and Anastasios Sidiropoulos. 2016. 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 '16)*. ACM, New York, NY, USA, 161–163. https://doi.org/10.1145/2935764.2935807
[4] Siu Man Chan, Massimo Lauria, Jakob Nordström, and Marc Vinyals. 2015. Hardness of Approximation in PSPACE and Separation Results for Pebble Games. In *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS '15)*. 466–485. https://doi.org/10.1109/FOCS.2015.36
[5] Stephen Cook and Ravi Sethi. 1974. Storage Requirements for Deterministic / Polynomial Time Recognizable Languages. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing (STOC '74)*. 33–39. https://doi.org/10.1145/800119.803882
[6] Stephen A. Cook. 1973. An Observation on Time-storage Trade off. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC '73)*. 29–33. https://doi.org/10.1145/800125.804032
[7] Erik D. Demaine. 2017. Lecture 22: History of memory models. In *6.851: Advanced Data Structures*. Massachusetts Institute of Technology. http://courses.csail.mit.edu/6.851/fall17/lectures/L22.html
[8] Erik D. Demaine and Quanquan C. Liu. 2017. Inapproximability of the Standard Pebble Game and Hard to Pebble Graphs. In *Proceedings of the 16th International Symposium on Algorithms and Data Structures, WADS '17 (Lecture Notes in Computer Science)*, Vol. 10389. Springer, 313–324.
[9] Rod G. Downey and Michael R. Fellows. 1995. Fixed-Parameter Tractability and Completeness I: Basic Results. *SIAM J. Comput.* 24, 4 (1995), 873–921. https://doi.org/10.1137/S0097539792228228 arXiv:https://doi.org/10.1137/S0097539792228228
[10] Rod G. Downey and Michael R. Fellows. 1995. Fixed-parameter Tractability and Completeness II: On Completeness for $W[1]$. *Theor. Comput. Sci.* 141, 1-2 (April 1995), 109–131. https://doi.org/10.1016/0304-3975(94)00097-3
[11] Venmugil Elango, Fabrice Rastello, Louis-Noël Pouchet, J. Ramanujam, and P. Sadayappan. 2015. On Characterizing the Data Access Complexity of Programs. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15)*. 567–580. https://dl.acm.org/citation.cfm?id=2677010
[12] Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.
[13] John R. Gilbert, Thomas Lengauer, and Robert Endre Tarjan. 1980. The Pebbling Problem is Complete in Polynomial Space. *SIAM J. Comput.* 9, 3 (1980), 513–524. https://doi.org/10.1137/0209038 arXiv:https://doi.org/10.1137/0209038
[14] Hong Jia-Wei and H. T. Kung. 1981. I/O Complexity: The Red-blue Pebble Game. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC '81)*. 326–333. https://doi.org/10.1145/800076.802486
[15] Jakob Nordstrom. 2015. New Wine into Old Wineskins: A Survey of Some Pebbling Classics with Supplemental Results. https://www.csc.kth.se/~jakobn/research/PebblingSurveyTMP.pdf. (2015).
[16] Desh Ranjan, John E. Savage, and Mohammad Zubair. 2012. Upper and lower I/O bounds for pebbling r-pyramids. *J. Discrete Algorithms* 14 (2012), 2–12.
[17] Ravi Sethi. 1975. Complete Register Allocation Problems. *SIAM J. Comput.* 4, 3 (1975), 226–248. https://doi.org/10.1137/0204020

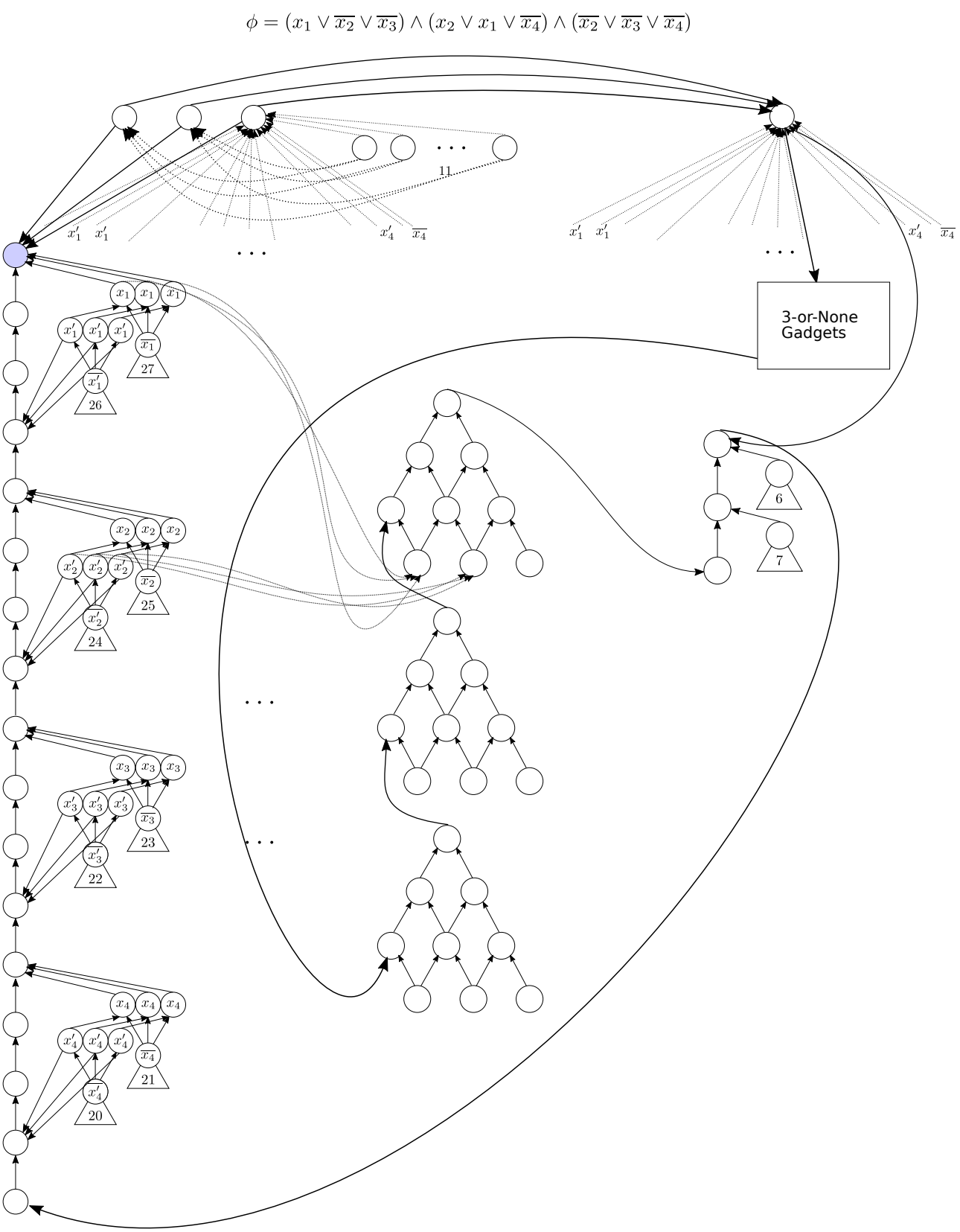

Figure 12: Example W[1]-hardness reduction for the red-blue pebble game. The vertex colored blue is the vertex that must be pebbled at the end and can only be pebbled if and only if the 3SAT instance has a solution that sets exactly k variables to True and uses at most $2k$ transitions.