Electronic Colloquium on Computational Complexity, Report No. 116 (2014)

2048 IS (PSPACE) HARD, BUT SOMETIMES EASY

Rahul Mehta*
Princeton University
rahulmehta@princeton.edu

August 26, 2014

Abstract

We prove that a variant of 2048, a popular online puzzle game, is PSPACE-Complete. Our hardness result holds for a version of the problem where the player has oracle access to the computer player's moves. Specifically, we show that for an $n \times n$ game board $\mathcal{G}$, computing a sequence of moves to reach a particular configuration $\mathbb{C}$ from an initial configuration $\mathbb{C}_0$ is PSPACE-Complete. Our reduction is from Nondeterministic Constraint Logic (NCL).

We also show that determining whether or not there exists a fixed sequence of moves $\mathcal{S} \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}^k$ of length k that results in a winning configuration for an $n \times n$ game board is fixed-parameter tractable (FPT). We describe an algorithm to solve this problem in $O(4^k n^2)$ time.

1. Introduction

The online video game 2048 [1] (read as "twenty-forty-eight") has recently gained a great deal of popularity. Played on a 4×4 game board with tiles containing positive powers of 2, the goal of the game is to combine equally-valued tiles to reach the 2048 tile.

More specifically, the game board begins with an initial configuration of two tiles, of value 2 or 4, placed at arbitrary locations on the grid. The player then selects to move UP, DOWN, LEFT, or RIGHT (denoted as $\{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}$). Each move shifts *all* tiles on the grid in the direction chosen. If two adjacent tiles have the same value (i.e. 2^i for some $i > 0$), they combine, and the single resulting tile after the combination will have value 2^{i+1}. Following the player's move, the computer places a tile of value 2 or 4 at a random free location on the board. Figure 1 outlines the first six moves of a game of 2048. Observe that if a tile's row or column in the direction of the move $x \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}$ is unobstructed, the tile will slide as far as possible on the grid.

Sliding tile games of a similar variety have been well-studied, both in the realm of puzzles as well as in algorithmic motion planning. Most recently, Demaine and Hearn ([4, 2]) proved that several problems including solving sliding-block puzzles, Rush Hour, Sokoban, and Push-2-F are all PSPACE-Complete. To accomplish this, they developed the Nondeterministic Constraint Logic (NCL) model of computation as a generic framework for PSPACE-Hardness results (see [4]).

1.1. Definitions

We first define some basic concepts and terms that we will use throughout the paper, and then proceed to outline the two problems that will be studied in subsequent sections. When we refer to a *game board* $\mathcal{G}$, we are referring to an $n \times n$ grid that can take on particular configurations, as described below;

*Department of Computer Science, 35 Olden St., Princeton University, Princeton, NJ, 08540.

ISSN 1433-8092

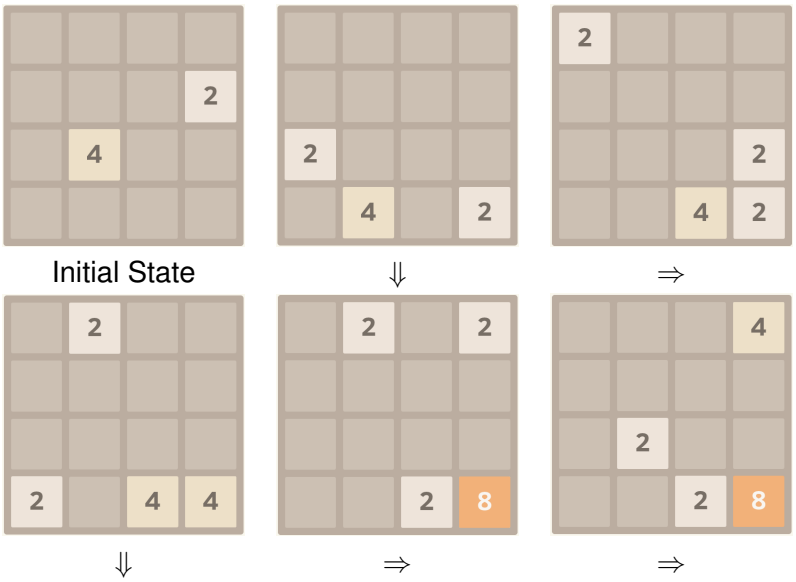


Figure 1: The first six moves of a game of 2048.

Definition 1. Given a game board $\mathcal{G}$, a *configuration* of the board $\mathbb{C} : [n] \times [n] \to \{2^\ell \mid \ell \in \mathbb{Z}_{>0}\} \cup \{0\}$ is a function that maps each of the n^2 locations on the game board to a positive power of 2, or 0. For a particular configuration $\mathbb{C}$, a location (i, j) is *empty* if $\mathbb{C}(i, j) = 0$.

We describe locations in row-major order; for example, in the final (lower-right) configuration in Figure 1, the tile of value 4 is located at $(1, 4)$, the two tiles of value 2 at $(3, 2)$ and $(4, 3)$, and the tile of value 8 at $(4, 4)$.

We study two variants of 2048; first, we examine the complexity of computing a sequence of moves to reach a particular configuration $\mathbb{C}$ from an initial configuration $\mathbb{C}_0$, and second, we exhibit an efficient algorithm for determining whether a winning sequence of moves of length k exists, starting from a particular initial configuration $\mathbb{C}_0$. In order to formalize these two variants of the problem, we first recast 2048 as a two-player game between the human player $\mathcal{A}$ and the computer adversary $\mathcal{B}$;

Definition 2 (The Game)**.** Given an initial configuration $\mathbb{C}_0$ of an $n \times n$ game board $\mathcal{G}$, consider the following game between adversaries $\mathcal{A}$ and $\mathcal{B}$, with a *goal configuration* $\mathbb{C}_f$. The game is played as follows;

(1) $\mathcal{A}$ makes a move $x \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}$.

(2) $\mathcal{B}$ places one or more tiles of value $2^{\ell_1}, \ldots, 2^{\ell_k}$ $(\ell_1, \ldots, \ell_k \in \mathbb{Z}_{>0})$ at locations $(i_1, j_1), \ldots, (i_k, j_k)$.

This game is played until one of two conditions occur; (1) the goal configuration $\mathbb{C}_f$ is reached, and $\mathcal{A}$ wins, or (2) there are no more moves that $\mathcal{A}$ can make (i.e. for any $x \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}$, applying x will not change the configuration of the game board). In this case, $\mathcal{B}$ wins. Such a configuration is shown in Figure 2

1.2. Problems

We now introduce the two problems that will be studied in this paper. Both variants that we study give player $\mathcal{A}$ *perfect knowledge*; that is, given a configuration $\mathbb{C}$ and a move $x \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}$, $\mathcal{A}$ has oracle access to $\mathcal{B}$'s move if x is applied to a board $\mathcal{G}$ with configuration $\mathbb{C}$. This is denoted as $\mathcal{B}(\mathbb{C}, x)$. The response is a set of 3-tuples of the form $\{(i_1, j_1, \ell_1), \ldots, (i_k, j_k, \ell_k)\}$, where for index t, (i_t, j_t) is the location of the added tile of value 2^{ℓ_t}. Accordingly, an instance of 2048-GAME is uniquely determined

Figure 2: An example of a configuration satisfying (2) (game over).

by (1) the initial configuration $\mathbb{C}_0$, and (2) the responses of the computer $\mathcal{B}$. In a sense, this is an easier version of the more general problem – it is interesting to note that our hardness result still holds despite the additional information available to $\mathcal{A}$.

Problem 1 (2048-GAME)**.** Given an $n \times n$ game board $\mathcal{G}$ and an oracle $\mathcal{B}$ to the computer player, does there exist a sequence of moves $\mathcal{S} \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}^*$ such that $\mathbb{C}$ is reached from $\mathbb{C}_0$?

In Section 3, we prove the following theorem:

Claim 1 (Theorem 1)**.** 2048-GAME *is PSPACE-Complete.*

We then examine the related problem of determining whether or not there exists a fixed-length sequence of moves that results in a winning configuration.

Problem 2 (2048-k-MOVES)**.** Given an $n \times n$ game board $\mathcal{G}$, an oracle $\mathcal{B}$ to the computer player, and a goal tile 2^m (for some $m > 0$), does there exist a sequence of moves $\mathcal{S} \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}^k$ of length k such that after k moves, the board is in a winning configuration $\mathbb{C}_f$ (that is, $\mathbb{C}_f(i, j) = 2^m$, for some i and j)?

When the problem is paramaterized in this manner (limiting the number of moves to some constant $k > 0$), we show that in fact 2048-k-MOVES is fixed-parameter tractable (FPT). In Section 4, we prove the following theorem:

Claim 2 (Theorem 2)**.** *Given an $n \times n$ game board $\mathcal{G}$,* 2048-k-MOVES *is solvable in $O(4^k n^2)$ time.*

Before concluding the section, we show that the value of the maximum tile is monotonically nondecreasing. For a configuration $\mathbb{C}$, we denote the tile of maximum value as $\max(\mathbb{C})$.

Lemma 1. *For a game board $\mathcal{G}$, a configuration $\mathbb{C}_i$, and the subsequent configuration $\mathbb{C}_{i+1}$, $\max(\mathbb{C}_i) \leq \max(\mathbb{C}_{i+1})$.*

Proof. Immediate from the combination rule of the game. When adjacent tiles of value 2^ℓ combine, they form a single tile of value $2^{\ell+1}$. Consider two cases; (1) the maximum tile does not change value, or (2) the maximum valued tile does change value. If (1) occurs, then we have equality, since $\max(\mathbb{C}_i)$ is left unchanged. For (2), let $\max(\mathbb{C}_i) = 2^\ell$, for some $\ell > 0$. The only case in which the tile in question changes is if it combines with a tile of the same value. By our previous observation, two adjacent tiles of value 2^ℓ will combine to form a single tile of value $2^{\ell+1}$. Thus, we have $\max(\mathbb{C}_{i+1}) = 2^{\ell+1} > 2^\ell = \max(\mathbb{C}_i)$, which completes the proof. □

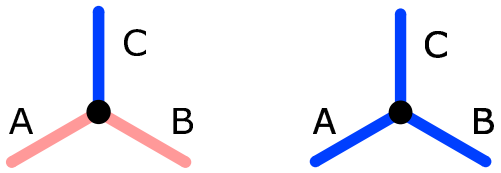
The remainder of the paper is organized as follows; Section 2 describes the Nondeterministic Constraint Logic (NCL) model of computation due to Demaine and Hearn [4], and specifically, the framework it provides for PSPACE-Hardness reductions. Section 3 describes in detail our reduction from NCL to 2048-GAME. Finally, Section 4 describes an efficient algorithm to solve 2048-k-MOVES, which establishes that it is fixed-parameter tractable (FPT).

2. Nondeterministic Constraint Logic (NCL)

We will now outline the Nondeterministic Constraint Logic (NCL) model of computation, which was developed by Demaine and Hearn [4] as a general framework for proving PSPACE-Completeness results. All of their PSPACE-Completeness results are reductions from Quantified Boolean Formula (QBF), a well-known PSPACE-Complete problem [3].

An NCL *machine* consists of a *constraint graph*, which is an arbitrary undirected graph $G = (V, E)$ with edge weights $w : E \to \{1, 2\}$ (all edges have weight 1 or 2). A *configuration* of the machine is an orientation of the edges of G. A configuration is *valid* if for each vertex $v \in V$, the sum of incoming edge weights is at least 2. A move is made by reversing a single edge in the network such that the configuration remains valid.

A natural question to ask is whether or not a particular edge can be reversed after a sequence of moves (CONFIG-TO-EDGE), or alternately, whether there exists a sequence of moves to reach a particular configuration from some initial configuration (CONFIG-TO-CONFIG). Both of these problems are PSPACE-Complete (see [4]). This hardness result still holds when we restrict the vertices to those with incident edge weights 1,1, and 2, and those with 1, 1, and 1. These vertices are illustrated in Figure 3 (taken from [4]):



(a) AND (b) OR

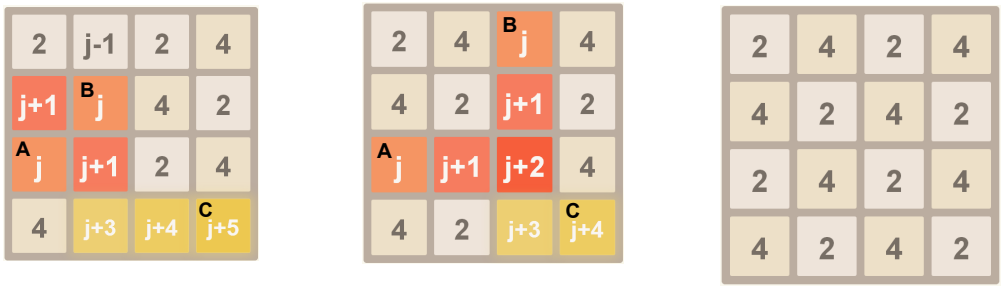
Figure 3: NCL AND and OR Vertices

Red edges have weight 1, and blue edges have weight 2. For an orientation of the edges, observe that for vertex (a), the blue edge can point out if and only if the two red edges are pointing in (to satisfy the previously-mentioned constraint that the in-flow on all vertices must be at least 2). Similarly, for (b), the top blue edge can point outward if and only if one of the bottom edges is pointing in. Thus, we can clearly see that (a) functions as an AND gate of sorts and (b) as an OR gate.

Demaine and Hearn in fact strengthen this result even further, and show that CONFIG-TO-EDGE and CONFIG-TO-CONFIG both remain PSPACE-Complete when the constraint graph G is planar. Thus, proving a game to be PSPACE-Hard reduces to simply constructing NCL AND and OR gadgets from the problem in question, and then demonstrating how to use them to construct arbitrary planar constraint graphs.

3. The Reduction

In this section, we outline the reduction from planar NCL to 2048-GAME. We begin by constructing NCL AND and OR vertex gadgets with 4×4 subinstances of 2048-GAME, and then demonstrate how to



(a) j-AND Gadget (b) j-OR Gadget (c) $(2, 4)$ Lattice

Figure 4: NCL AND and OR Vertices. Tiles with labels j, $j+1$, etc. correspond to values 2^j, 2^{j+1}, etc. In both gadgets, connections A and B are facing *out*, while connection C is facing *in*.

connect them together to make arbitrary planar constraint graphs. This section will prove the following theorem;

Theorem 1. 2048-GAME *is PSPACE-Complete.*

Before outlining the reduction, however, we must show that 2048-GAME is in fact contained in PSPACE. A rigorous proof of this fact is given below;

Lemma 2. 2048-GAME $\in$ *PSPACE.*

Proof. We begin by giving an NPSPACE algorithm to decide 2048-GAME. For an $n \times n$ game board $\mathcal{G}$, we clearly can represent the current configuration of the game board in polynomial space. Starting with an initial configuration $\mathbb{C}_0$ and a goal configuration $\mathbb{C}_f$, we can nondeterministically select a move $x \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}$ at each step (without consulting the oracle to $\mathcal{B}$), and update the previous configuration with the current one. We repeat the above procedure until one of the following conditions occurs;

(1) The goal configuration $\mathbb{C}_f$ is reached, in which case output YES.

(2) There are no possible moves from the current configuration $\mathbb{C}$, and it is not the goal configuration (game over); output NO.

(3) For the current configuration $\mathbb{C}$, $\max(\mathbb{C}) > \max(\mathbb{C}_f)$, in which case output NO.

Checking for these three conditions guarantees that the algorithm will terminate. Specifically, for condition (3), if there is a tile of value greater than the maximum-valued tile in the goal configuration $\mathbb{C}_f$, then $\mathbb{C}_f$ is unreachable from the current configuration $\mathbb{C}$. This is immediate from Lemma 1. Thus, we have an NPSPACE algorithm for deciding 2048-GAME. This is easily converted to a PSPACE algorithm due to Savitch's Theorem [7], completing our proof. □

3.1. NCL AND and OR Gadgets

In order to prove Theorem 1, we first construct NCL AND and OR gadgets from small instances of 2048-GAME, and then show how to connect them into arbitrary planar graphs.

Figure 4 shows two examples of AND and OR vertex gadgets. Since the tiles in 2048 are assigned a numerical value, an added difficulty presents itself when connecting vertex gadgets together (since the gadgets rely on adjacent tiles containing specific powers of 2, including the connectors to "activate" each vertex gadget). Thus, we introduce the notion of a j-OR and j-AND gadget, as is shown in Figure

4. The labels A, B, and C refer to the connection points for each gadget; these labels correspond to the edges for the NCL vertices in Figure 3.

A connection A or B is considered to be *activated* for a j-AND or j-OR gadget if the tile has value not equal to 2^j – this directly corresponds to reversing an edge in an NCL constraint graph. For both AND and OR gadgets, A and B are facing *in* if the corresponding tile is activated, and *out* otherwise.

The tile labeled C is also considered to be activated if its value has increased; thus, for a j-AND gadget, C is activated if the tile a value not equal to 2^{j+5} or greater, and for a j-OR gadget, if the tile has not equal to 2^{j+4}. Accordingly, C is facing *out* if it is activated, and *in* otherwise.

Before proving the correctness of the two gadgets, we outline the use for the gadget in Figure 4(c); the $(2, 4)$ Lattice. This structure has an important property; for any move $x \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}$, the configuration remains unchanged. Thus, it is perfectly rigid. We will embed all of the other gadgets and connection pieces into the $(2, 4)$ lattice so as to prevent large portions of the game board shifting when a move is made by $\mathcal{A}$.

Additionally, whenever a square of the game board is uncovered by a move, the oracle $\mathcal{B}$, in this construction, will respond by placing a tile $t \in \{2, 4\}$ in the vacant location, depending on which tile correctly continues the $(2, 4)$ lattice pattern. This is where the *perfect knowledge* comes into play; we are allowed to "program" player $\mathcal{B}$'s responses to various moves, which greatly simplifies parts of the reduction.

Lemma 3. *For a j-AND vertex gadget,* C *can face out if and only if* A *and* B *are facing in.*

Proof. First, we observe that the tiles in the j-AND gadget will not shift at all unless either A or B is activated, for any move $x \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}$, since there are no adjacent tiles of the same value.

Next, we demonstrate that for any sequence of moves which results in C activating, A and B must both be activated. We claim that for C to be activated in configuration $\mathbb{C}_i$, $\mathbb{C}_{i-3}(3, 2) = 2^{j+3}$. That is, location $(3, 2)$ must contain 2^{j+3} before C can be activated. This is summarized in the diagram below;

$$\mathsf{C} = \mathbb{C}_i(4, 4) = 2^{j+6} \;\rightarrow\; \mathbb{C}_{i-1}(4, 3) = 2^{j+5} \rightarrow \mathbb{C}_{i-2}(4, 2) = 2^{j+4} \rightarrow \mathbb{C}_{i-3}(3, 2) = 2^{j+3}$$

Correctness of the diagram can be shown inductively, by assuming that the value in $\mathbb{C}_{k-1}$ is necessary for the specified value in $\mathbb{C}_k$ to appear. Thus, proving the lemma reduces to determining the conditions under which there is a $\mathbb{C}$ such that $\mathbb{C}(3, 2) = 2^{j+3}$.

Since we have already shown that A or B must be activated for any movement within the gadget to occur, it suffices to prove that only activating A or B does not suffice. Without loss of generality, assume that A is activated but not B (the proof of the opposite assumption follows the exact same reasoning). If only A is activated, then for some $\mathbb{C}$, $\mathbb{C}(3, 1) = 2^{j+1}$, and there exists a sequence of moves such that for some subsequent $\mathbb{C}'$, $\mathbb{C}'(3, 1) = 2^{j+2}$. However, $\mathbb{C}'(3, 2) = 2^{j+1}$, so no combination is possible to achieve $\mathbb{C}''(3, 2) = 2^{j+3}$, as is desired. Thus, A and B must be activated for C to be activated, which concludes the proof.

□

An example of a sequence of moves to activate C for a 4-AND vertex gadget is in Appendix A (Figure 12). Now, we turn our attention to the j-OR vertex. The proof closely follows that of Lemma 3.

Lemma 4. *For a j-OR vertex gadget,* C *can face out if and only if* A *or* B *are facing in.*

Proof. Again, we see that tiles in the j-OR gadget will not shift unless A or B are activated, for $x \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}$, since no adjacent tiles in the gadget's initial configuration are of the same value.

Next, we show that for any sequence of moves resulting in the activation of C, either A or B must be activated. Suppose C is activated in configuration $\mathbb{C}_i$. Backtracking along the tiles in the gadget, we show that for C to be activated in $\mathbb{C}_i$, then in $\mathbb{C}_{i-4}$, A or B must be activated. The diagram below

describes the required configurations. The correctness of the diagram is proven inductively by assuming that the value(s) described in $\mathbb{C}_{k-1}$ is a necessary condition for the value(s) in $\mathbb{C}_k$ to be reached.

$$\mathsf{C} = \mathbb{C}_i(4,4) = 2^{j+5} \rightarrow \mathbb{C}_{i-1}(4,3) = 2^{j+4} \rightarrow \mathbb{C}_{i-2}(3,3) = 2^{j+3} \begin{cases} \nearrow \mathbb{C}_{i-3}(3,2) = 2^{j+2} \rightarrow \mathbb{C}_{i-4}(3,1) = 2^{j+1} = \mathsf{A} \\ \searrow \mathbb{C}_{i-3}(2,3) = 2^{j+2} \rightarrow \mathbb{C}_{i-4}(1,3) = 2^{j+1} = \mathsf{B} \end{cases}$$

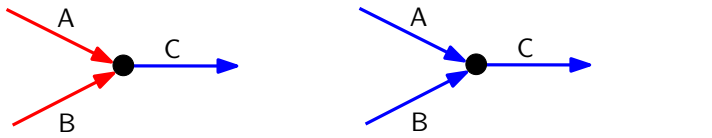
The validity of the moves in the diagram can be verified by inspection of the j-OR gadget in Figure 4(b). The fact that $\mathbb{C}_{i-4}(\mathsf{A}) = 2^{j+1}$ or $\mathbb{C}_{i-4}(\mathsf{B}) = 2^{j+1}$ must hold for $\mathbb{C}_i(\mathsf{C}) = 2^{j+5}$ is immediate from the diagram; we conclude that C can be activated, and thus face out, if and only if A or B is activated, and therefore facing in. □

An example of a sequence of moves to activate C for a 4-OR vertex gadget is in Appendix A (Figure 13).

3.2. The Reversible NCL AND/OR Gadget

Now that we have described the NCL AND and OR gadgets, we have one final vertex gadget left to define. Observe that for either the NCL j-AND or j-OR gadget, once it is activated, the configuration cannot be reversed; that is, once a gadget is set in a particular configuration, it cannot be altered. Thus, if the initial configuration of an NCL machine $\mathbb{C}_0$ contains an activated AND vertex, and the goal is to reverse one of the incoming edges to the vertex, then this is impossible in our vertex gadget construction but possible in NCL.

To alleviate this, we construct a final gadget, namely the Reversible j-AND/OR gadget. This gadget is placed in an instance of 2048-GAME when an AND vertex in the original NCL constraint graph is activated, or when both edges of weight 1 are entering an OR vertex. The conditions under which a Reversible j-AND/OR gadget is used are outlined in Figure 5.



Activated AND Activated OR (2 incoming edges)

Figure 5: The conditions under which a Reversible j-AND/OR is used in place of a normal vertex gadget.

We now introduce the gadget. A and B are considered to be activated if their value is not equal to 2^{j+4}. Thus, in the initial configuration they are facing *in*, and when activated, are facing *out*. C is activated when its value is not equal to 2^j; when it is activated, it is facing *in*, and when it is not, it is facing *out*. The gadget is defined in Figure 6.

One specific that we note about the gadget is it's behavior after the tile at location $(2,3)$ (in the initial configuration, 2^{j+2}) is shifted from that location, the computer player $\mathcal{B}$ places a new tile of value 2^{j+3} at $(2,3)$. That way, both A and B can be activated, and therefore face out. However, it is also possible for either A or B to face out, but not the other. In fact, we prove that all possible configurations of the vertex gadgets in Figure 5 are reachable with the Reversible j-AND/OR gadget. These reachable configurations are listed in Figure 7 below;

We now prove that these configurations are reachable by exhibiting sequences of moves;

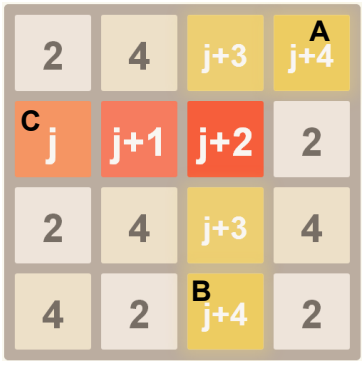
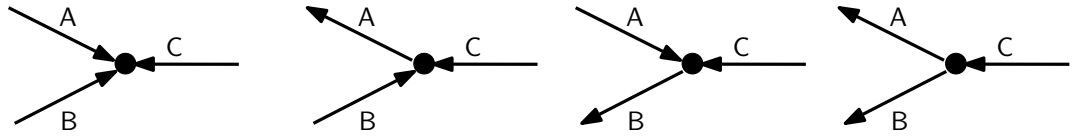


Figure 6: The Reversible j-AND/OR vertex gadget.

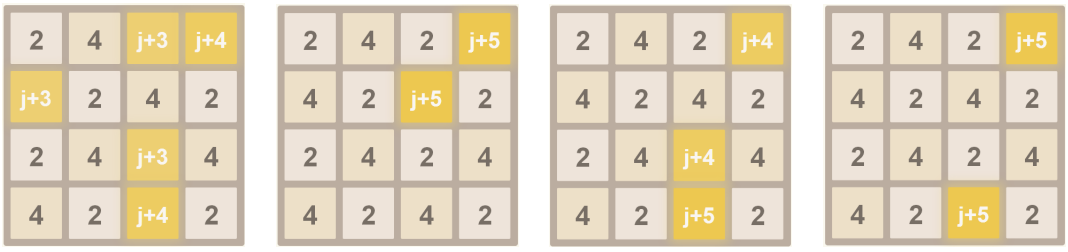


(a) (b) (c) (d)

Figure 7: Possible configurations reachable by Activated AND and OR vertices with two incoming edges.

Lemma 5. *The configurations listed in Figure 7 are reachable for any given Reversible j-AND/OR gadget.*

Proof. The following diagram depicts the Reversible j-AND/OR gadget in the four configurations listed in Figure 7. The proof of the lemma below contains the sequences of moves necessary to reach each configuration; this can be verified by inspection. The diagram shows the configurations for a Reversible j-AND/OR gadget.



(a) (b) (c) (d)

We now give sequences of moves to achieve the given configurations. The sequences can easily be verified by inspection of the gadget; (a) $\Rightarrow, \Leftarrow, \Leftarrow$ (b) $\Rightarrow, \Rightarrow, \Rightarrow, \Uparrow, \Rightarrow, \Uparrow, \Uparrow$ (c) $\Rightarrow, \Rightarrow, \Rightarrow, \Downarrow, \Downarrow$ (d) $\Rightarrow, \Rightarrow, \Rightarrow, \Uparrow, \Rightarrow, \Downarrow, \Downarrow$. Thus, the configurations in Figure 7 are all reachable by the Reversible j-AND/OR vertex, completing the proof. □

3.3. Constructing Arbitrary Planar Constraint Graphs

We now describe how to construct arbitrary planar graphs from the NCL vertex gadgets. The vertex gadgets, as well as the connection pieces described in this section, are embedded in an arbitrarily large game board $\mathcal{G}$ comprised of 4×4 sub-instances. Such a layout is shown in Figure 8

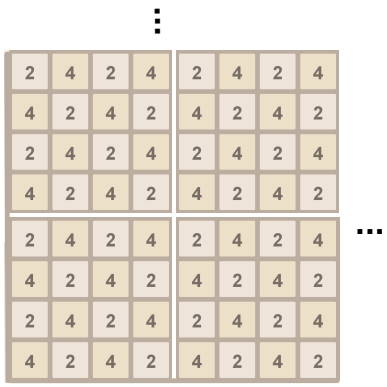


Figure 8: An arbitrarily large game board $\mathcal{G}$ comprised of 4×4 sub-instances of 2048-GAME.

There is also a second issue that we must navigate. Due to the vertex gadgets' connection tiles (A, B, and C) containing numerical values of 2, connecting arbitrary vertices is not as simple as in other reductions from NCL.

We address this by turning to the famed Four Color Theorem (see [5] for more detail regarding several proofs). Since we are constructing *planar* NCL constraint graphs, we can turn to the Four Color Theorem, and in particular, an $O(n^2)$ algorithm by Robertson, et. al. [6] to find a four coloring. By assigning colors to the vertices in the constraint graph, we are able to specify the value of j for the j-AND, j-OR, and Reversible j-AND/OR gadgets. For each of the four colors, let $j = 4, 5, 6, 7$ respectively. Note that the diagrams in Appendix A are for NCL gadgets with $j = 4$.

The k-k'-CONNECTION Gadget We now describe a connection piece to connect gadgets with differing values of j. We call this a k-k'-CONNECTION piece, and its layout in our context of the 4×4 sub-instance of 2048-GAME. The diagram is contained in Figure 9

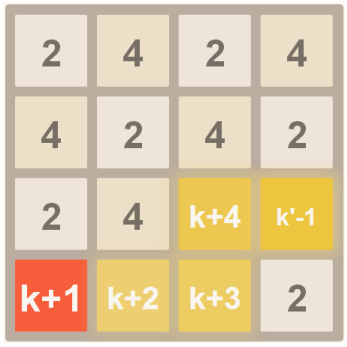


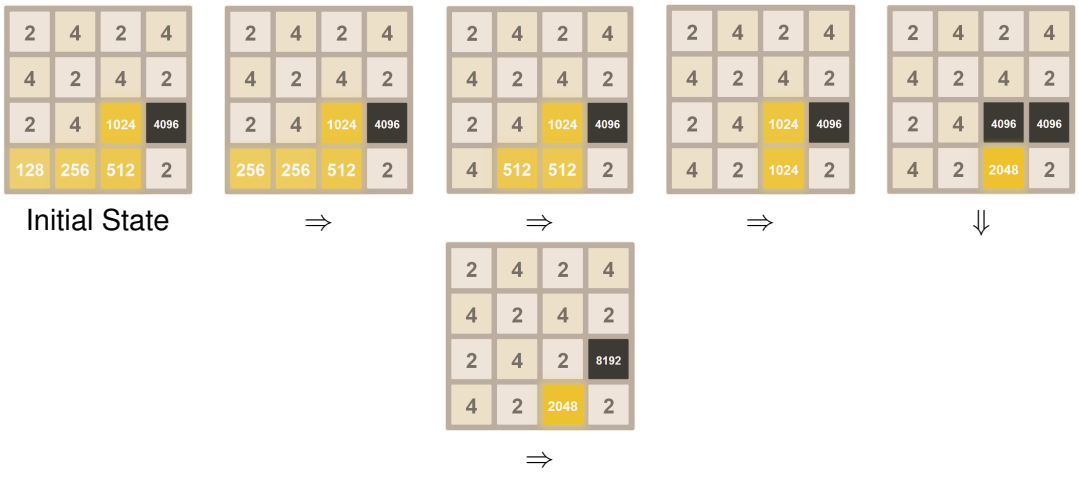
Figure 9: The k-k'-CONNECTION gadget.

The gadget is attachable to a gadget that will terminate with value 2^k on any one of its output edges (C for normal AND/OR gadgets, and A and B for the Reversible AND/OR gadget). We note that this gadget can be shifted vertically within the 4×4 sub-instance to yield connection pieces for output tiles at varying locations.

The major "trick," so to speak, lies in the gadget's activation sequence. When the tile with initial value of $k + 4$ at $(3, 3)$ vacates that location, the computer player $\mathcal{B}$ will place a tile of value $k' - 1$ at $(3, 3)$. Then, the tile is free to combine with the second tile of value $k' - 1$ at $(3, 4)$, which will result in $(3, 4)$ having the value k', which in turn implies that it will activate a gadget with $j = k'$.

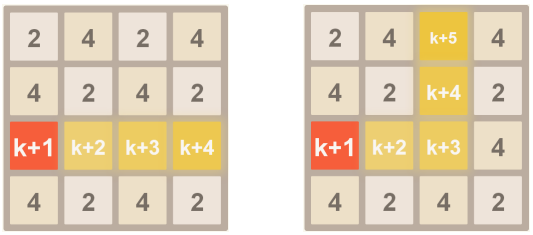
Lemma 6. *The k-k'-*CONNECTION *gadget contains an activation sequence such that an outgoing tile of value k will activate a gadget with $j = k'$.*

Proof. We will exhibit an activation sequence of the gadget to demonstrate that it indeed fulfills its promise, namely that it will activate an NCL vertex gadget with $j = k'$ if the outgoing tile of the previous gadget has value k. This is exhibited with values $k = 7$ and $k' = 14$.



□

The k-LINE and k-CORNER Pieces We finally describe two additional connection pieces which will finish our construction of arbitrary planar graphs. These are the k-LINE and k-CORNER gadgets, and are outlined below in Figure 10;



(a) k-LINE (b) k-CORNER

Figure 10: k-LINE and k-CORNER connection pieces.

3.4. Proof of Theorem 1

Now that we have finished describing the components of the reduction, we are ready to prove the main theorem (that 2048-GAME is PSPACE-Complete);

Proof of Theorem 1. Immediate from the construction described in Section 3. Specifically, we first find a valid four-coloring of the NCL constraint graph G using the algorithm due to Robertson, et. al. [6], and then follow the procedure outlined in Section 3.3 to connect the NCL vertex gadgets together using a combination of k-k'-CONNECTION, k-LINE, and k-CORNER pieces.

Next, we must orient the edes of the constraint graph G according to its current configuration. We set the tiles in the vertex gadgets accordingly; if an edge corresponds to an *activated* configuration, the appropriate tile value is set on the vertex gadget (either at A, B, or C).

We have now shown how to convert an orientation of an arbitrary planar constraint graph into an instance of 2048-GAME; if we can decide 2048-GAME, then we can decide CONFIG-TO-CONFIG. Thus, 2048-GAME is PSPACE-Hard, and by Lemma 2, 2048-GAME $\in$ PSPACE. Therefore, we conclude that 2048-GAME is PSPACE-Complete.

□

4. An FPT Algorithm for 2048-k-MOVES

On one hand, our paper shows that the most natural problem that emerges from 2048 is intractable. However, we show in this section that another variant of the game is fixed-parameter tractable (FPT). Recall that 2048-k-MOVES is the problem of deciding whether or not there exists a sequence of moves of length k $\mathcal{S} \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}^k$ such that after each of the moves is executed, the board will be in a winning configuration.

This problem is decidable in polynomial time due to the fact that the number of moves is constant. While the constant may be quite large, it does not change the asymptotic running time of our algorithm. First, we introduce the notion of a *game tree*;

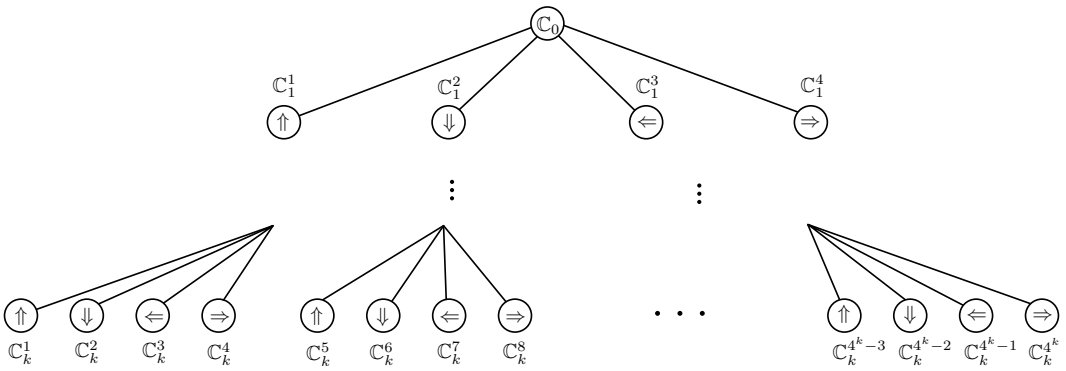


Figure 11: A game tree for an instance of 2048-k-MOVES of depth k.

Each vertex of the tree that can be reached by a path of length ℓ represents a possible configuration of the game after ℓ moves. The tree can be constructed in $O(4^k)$ time by querying the oracle for the 4^k possible sequences of moves $\mathcal{S} \in \{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}^k$. Our algorithm, in essence, traverses the vertices of the game tree and halts when a winning configuration is found, or if there are none. We are now ready to prove the second result of our paper;

Theorem 2. *Given an $n \times n$ game board $\mathcal{G}$,* 2048-k-MOVES *is solvable in $O(4^k n^2)$ time.*

Proof. The algorithm can perform any recursive traversal of the game tree to explore the leaf nodes. We first observe that the game tree is in fact a 4-ary tree. It is well-known that for a d-ary tree of depth ℓ, the number of leaf nodes is d^ℓ. Thus, for our 4-ary tree of depth k, there are 4^k leaf nodes.

When each leaf is visited, the value of each of the n^2 tiles on the game board is compared to 2^m, the goal tile. If they are equal, halt and output YES and the configuration $\mathbb{C}_f$. If a game board is in a game over configuration (i.e. the subsequent configurations of the four children nodes are precisely equal to the current configuration), terminate the recursion for that branch of the tree.

Since we check each of the n^2 tiles of $\mathcal{G}$ at each of the 4^k leaf nodes, the worst-case running time of the procedure is $O(4^k n^2)$, showing that 2048-k-MOVES $\in$ FPT. □

References

[1] Gabriele Cirulli, *2048*, http://gabrielecirulli.github.io/2048/, Accessed: 2014-07-02.

[2] Erik D. Demaine, Robert A. Hearn, and Michael Hoffmann, *Push-2-f is pspace-complete*, CCCG, 2002, pp. 31–35.

[3] 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, 1979.

[4] Robert A. Hearn and Erik D. Demaine, *Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation*, Theor. Comput. Sci. **343** (2005), no. 1-2, 72–96.

[5] Jirí Matousek and Jaroslav Nesetril, *Invitation to discrete mathematics (2. ed.)*, Oxford University Press, 2009.

[6] Neil Robertson, Daniel P. Sanders, Paul D. Seymour, and Robin Thomas, *The four-colour theorem*, J. Comb. Theory, Ser. B **70** (1997), no. 1, 2–44.

[7] Walter J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. Syst. Sci. **4** (1970), no. 2, 177–192.

A. Activation Sequences for AND and OR Gadgets

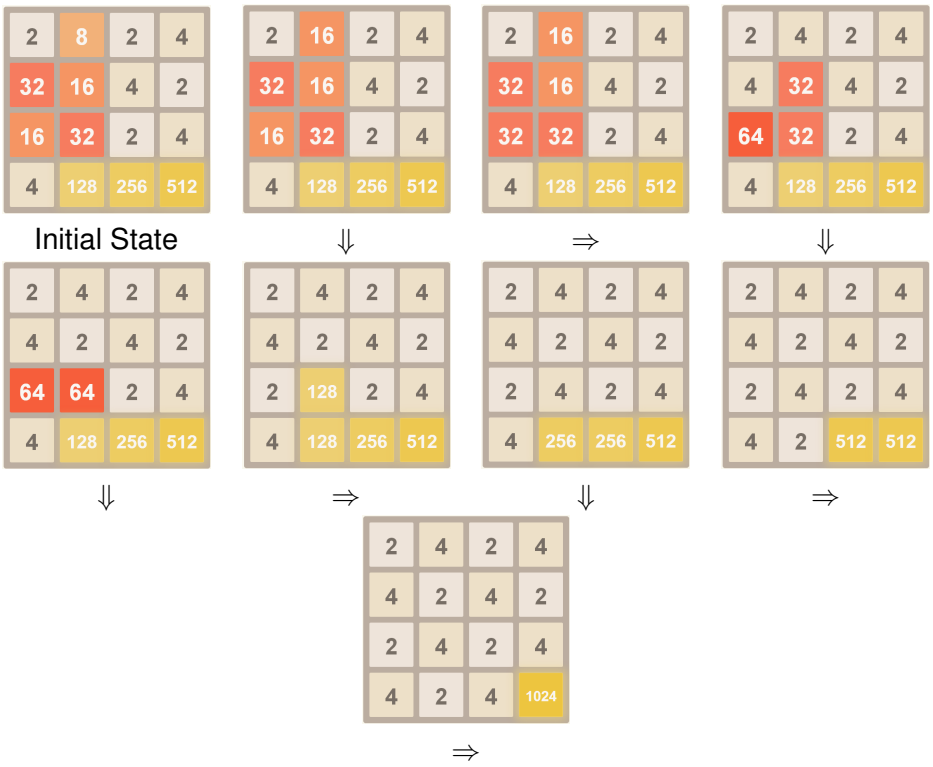


Figure 12: A possible sequence for activating C in a 4-AND gadget (⇓, ⇒, ⇓, ⇓, ⇒, ⇓, ⇒, ⇒).

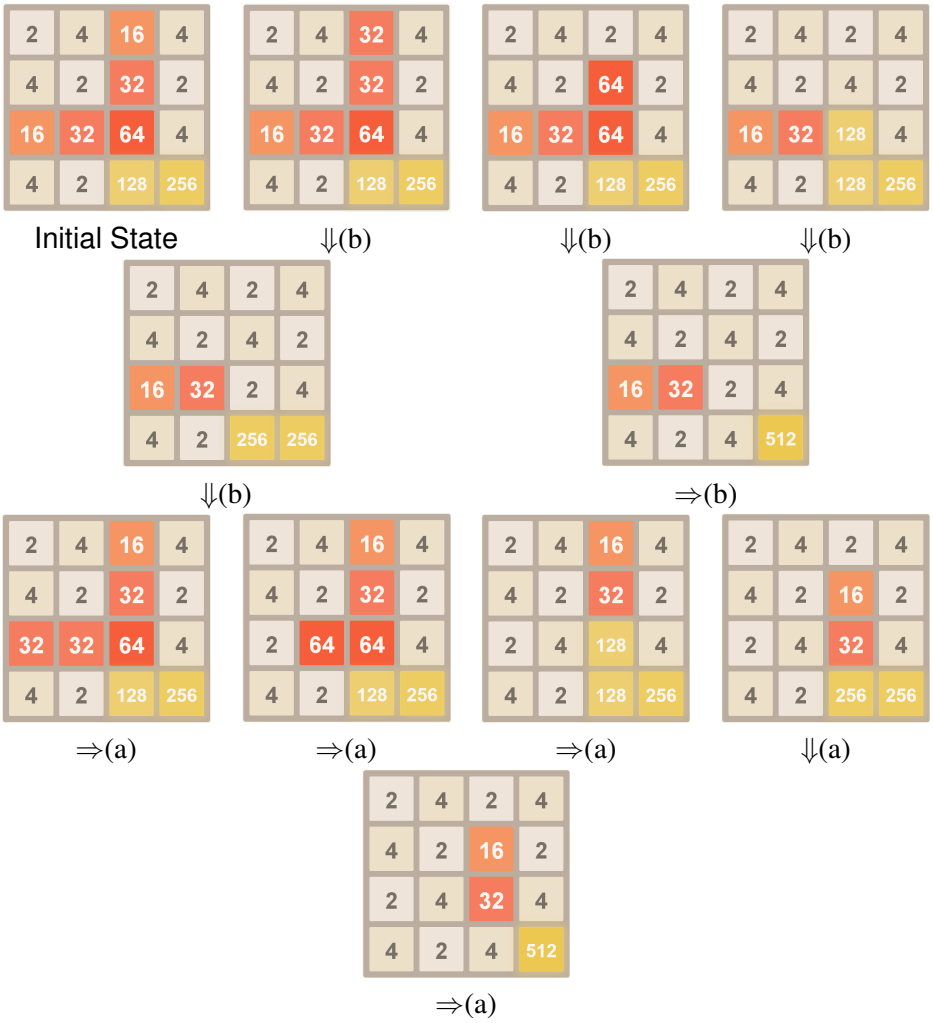


Figure 13: Two possible sequences of moves to activate C; (b) $\Downarrow, \Downarrow, \Downarrow, \Downarrow, \Rightarrow$, and (a) $\Rightarrow, \Rightarrow, \Rightarrow, \Downarrow, \Rightarrow$ (corresponding to A and B as the activating tile, respectively).

ECCC ISSN 1433-8092

http://eccc.hpi-web.de