

Threes!, Fives, 1024!, and 2048 are Hard

Stefan Langerman^{*1} and Yushi Uno²

1 Département d'informatique, Université Libre de Bruxelles, ULB CP 212,
avenue F.D. Roosevelt 50, 1050 Bruxelles, Belgium
stefan.langerman@ulb.ac.be

2 Department of Mathematics and Information Sciences, Graduate School of
Science, Osaka Prefecture University, 1-1 Gakuen-cho, Naka-ku, Sakai
599-8531, Japan
uno@mi.s.osakafu-u.ac.jp

Abstract

We analyze the computational complexity of the popular computer games Threes!, 1024!, 2048 and many of their variants. For most known versions expanded to an $m \times n$ board, we show that it is NP-hard to decide whether a given starting position can be played to reach a specific (constant) tile value.



1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2 Discrete Mathematics, F.1.2 Modes of Computation

Keywords and phrases algorithmic combinatorial game theory

Digital Object Identifier 10.4230/LIPIcs.FUN.2016.22

1 Introduction

Threes! [13] is a popular puzzle game created by Asher Vollmer, Greg Wohlwend, and Jimmy Hinson (music), and released by Sirvo for iOS on January 23, 2014. The game received considerable attention from players, game critics and game designers. Only a few weeks after its release, an Android clone *Fives* appeared, and then an iOS clone *1024!* with slightly modified rules. Shortly after, two open source web game versions, both called *2048* were released on github on the same day, one by Saming [12], the other by Gabriele Cirulli [5]. Since then over a hundred new variant have been catalogued [9]. In December 2014, Threes! received the Apple Game of the Year and the Apple Design award.

2048 We first describe Cirulli's 2048 [5] (or just 2048 for short) which has a slightly simpler set of rules (see Fig. 2). The game is played on a 4×4 square grid *board*, consisting of 16 *cells*. During the game, each cell is either empty or contains a *tile* bearing a *value* which is a power of two. When the game begins, a (random) starting *configuration* of tiles is placed on the board. Then, in every turn, one plays a move by indicating one of four directions, up, down, left or right, and then each numbered tile moves in that direction, either to the boundary of the board (a *wall*) or until it hits another tile. When two tiles of value K hit, they merge to become a single tile of value $2K$. If three or more tiles with the same value hit, they merge two by two, starting with the two closest to the wall in the direction of the move. If no tile can move in some direction (e.g., all tiles touch the wall), then that move is *invalid*. After each turn, a new tile of value  or  appears in

* Directeur de Recherches du F.R.S.-FNRS.



© Stefan Langerman and Yushi Uno;
licensed under Creative Commons License CC-BY

8th International Conference on Fun with Algorithms (FUN 2016).

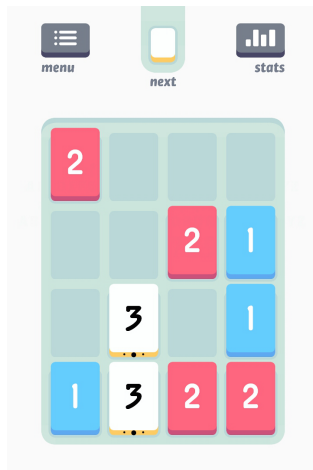
Editors: Erik D. Demaine and Fabrizio Grandoni; Article No. 22; pp. 22:1–22:14

Leibniz International Proceedings in Informatics

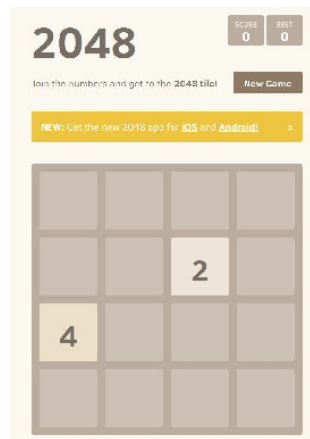


LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

22:2 Threes!, Fives, 1024!, and 2048 are Hard



■ **Figure 1** Threes!



■ **Figure 2** The game 2048: a board and one of its initial configurations.



■ **Figure 3** A forbidden (game over) configuration.

a random empty cell. The objective of the game is to make a tile of value 2048, and/or to maximize the *score* defined as the sum of all new tiles created by merges during the game. If during the game, the board is completely filled and no move is valid, then the game is over and the player loses. We call such a configuration *forbidden* (Fig. 3).

Threes! The tiles in Threes! have values 1, 2, and $3 \cdot 2^i, i \geq 0$. The tiles 1 and 2 combine to form tile 3, and tiles of value $K \geq 3$ combine to form tile $2K$. Another important difference is that when performing a move, all tiles move in the corresponding direction by at most one cell instead of moving as far as possible. For example during a left move, a tile next to the left wall (if any) doesn't move (we say it is *blocked*). Then looking at the tiles from left to right, tiles immediately to the right of one that is blocked will either be blocked as well or will move left to merge with the blocked one if they can combine. A tile next to one that moves or next to an empty space will move one cell to the left. New tiles appear according to an unknown rule, which seems to change depending on the version of the game. It seems to be always of low value and on a cell next to a wall. At the end of the game, the score is computed by totalling 3^{i+1} points for each tile of value $3 \cdot 2^i$ on the board.

Fives was the first clone of Threes! for Android devices. Its rules are nearly identical to Threes!, except that the base tiles have values 2 and 3 which combine to form tile 5, all other tiles have values $5 \cdot 2^i, i \geq 0$.

1024! The main difference with 2048 is that there is a fixed block in the middle of the board that doesn't move during the game. A new tile appears after each turn at a random location on the board (not necessarily next to a wall). The goal is to reach 1024.

Saming's 2048 Just as in Cirulli's 2048, tiles are powers of two, however the tiles move according to slightly different rules. During a left move for example, tiles are considered in each row from right to left. A tile t will only move if its left neighboring cell is empty or of identical value. If there is no tile left of t , then t is moved to a cell adjacent to the left wall. Otherwise let s be the next tile left of t . If t and s are of identical values, the two cells are merged (and the value doubled), and the merged tile does not move this turn. Otherwise, the tile t stops just to the right of s . A new tile of value 2 or 4 is inserted in a random empty cell at the end of the move.



■ **Figure 4** Deterministic 2048.

Det2048 This version is identical to 2048 except that a new tile always appears in the first empty cell (leftmost, then topmost) and its value is always 2. In its initial configuration, only a single tile 2 is placed in the upper left cell (Fig. 4).

Fibonacci In this popular version, the tiles have values from the Fibonacci sequence, and only tiles of successive values in the sequence are combined.

Other than those, the most natural variants use larger boards, and set higher goal tile values.

The goal of this paper is to determine the computational complexity of Threes!, 2048 and many of their variants. We follow the usual offline deterministic model introduced by Demaine et al. for the videogame Tetris [3]. Given an initial configuration of tiles in an $m \times n$ board, we assume the player has full knowledge of the new pieces that will be added to the board after each move¹. We prove that even with offline deterministic knowledge (and in all variants listed above), it is NP-hard to optimize several natural objectives of the game:

- maximizing the largest tile created (MAX-TILE),
- maximizing the total score (MAX-SCORE), and
- maximizing the number of moves before losing the game (MAX-MOVE).

We show in fact that all three problems are inapproximable. The decision problem for MAX-TILE is already NP-hard for a constant tile value, where the constant depends on the variant of the game considered. On the other hand, MAX-MOVE is clearly fixed-parameter tractable (FPT) [7], that is, determining if k moves can be performed without losing takes only $O(4^k mn)$ time (since there are only 4 moves possible at every step). Likewise, every merge increases the score by at least 4, and so the number of moves to achieve score x is at most $x/4$, therefore determining if score x can be achieved takes only $O(4^{x/4} mn)$ and MAX-SCORE is also FPT.

Related works. The tractability of computer games falls under the larger field of *Algorithmic Combinatorial Game Theory* which has received considerable interest over the past decade. See Demaine and Hearn [6] for a recent survey. Block pushing and sliding puzzles are probably the most similar to the games studied here, a notable difference being that here (i) new tiles are (randomly) inserted after each move and (ii) a merging mechanism reduces the number of tiles at each step. Without these differences, the game would be nearly identical to the *Fifteen puzzle* and its generalizations, which interestingly can be solved very efficiently.

¹ We ignore for now the issue of representing the position of the new tiles, which might depend on which cells of the board are empty, and thus on previous moves. As we will see, this will have little influence on the main results.

22:4 Threes!, Fives, 1024!, and 2048 are Hard

In a blog post [4], Christopher Chen proved that 2048 is in NP, but for a variant where no new tile is inserted after each move. More recently, two articles have appeared on arXiv with the aim of analyzing the complexity of Threes! and 2048. The first one [8] notes 2048 is FPT (as discussed above) and claims PSPACE-hardness of 2048 by reduction from Nondeterministic Constraint Logic. However Abdelkader et al. [1] noted several issues with that reduction. For instance, in order for their gadgets to function properly, they need to modify the way tiles are moved and inserted during the game. In particular, they allow tiles to be inserted by the game at specific places in the middle of rows and columns in order to maintain an invariant base pattern. Furthermore, the goal tile value in the reduction is a (rather large) function of the input size. In an attempt to resolve these issues, Abdelkader et al. [1, 2] analyze the variant studied by Chen, in which no new tiles are generated during the game. For that case, they show that for 2048, it is NP-complete to decide if a specific (constant) tile value can be reached.

In the present paper, we analyze Threes! and 2048, where new tiles appear and tiles move and merge exactly as they do in the original games. Our proofs are easily extended to most existing variants of the game. We also prove inapproximability results for all these games, and show most of them are in NP.

2 Definitions

The simplest version of these games to describe is probably Det2048, where after each move a tile $\boxed{2}$ appears in the first empty cell of the board (in lexicographic order, leftmost, then topmost).

MAKE- \boxed{T} -DET2048

Instance: An $m \times n$ board with an initial configuration of tiles, each of which has for value a power of 2, and a number T , where $T = 2^c$ with some constant c . At the end of every turn, a new tile $\boxed{2}$ appears in the first empty cell in lexicographic order.

Question: Can one make \boxed{T} from the given configuration by a sequence moves (up, down, left and right)?

However in the original game, both tiles $\boxed{2}$ and $\boxed{4}$ can appear, at a location determined by the game. Since we consider an offline model, the value and location of the new tiles should be provided in the input. But while encoding the value of the tile is easy, its location does not have such a natural representation, because the new tile can only be inserted in an empty cell of the board, and the set of empty cells depends on the previous moves in the game. One could conceive several reasonable encodings (e.g., for each new tile, coordinates (x, y) such that the new tile should be placed in the closest/lexicographically first empty cell from cell (x, y)). To make our results as general as possible, we just assume the *location* information is encoded in constant space, and the game uses that information to place the new tile.

MAKE- \boxed{T}

Instance: An $m \times n$ board with an initial configuration of tiles, each of which has for value a power of 2, a number T , where $T = 2^c$ with some constant c , and the sequence of values ($\boxed{2}$ or $\boxed{4}$) and location of the new tiles to be placed by the game at the end of every turn.

Question: Can one make \boxed{T} from the given configuration by a sequence moves (up, down, left and right)?

In this setting, the original game 2048 is as MAKE-TILE with $m = n = 4$ and $T = 2048 = 2^{11}$ ($c = 11$).

It will be useful to denote some of the variants of MAKE-TILE by appending qualifiers to their name. For example, in the variant MAKE-TILE ONLY-2, only tiles 2 appear after each move. In the DETERMINISTIC variant, new tiles always appear in the lexicographically first empty cell, and so MAKE-TILE ONLY-2 DETERMINISTIC is exactly MAKE-TILE-DET2048.

For the purpose of analyzing the complexity of the game, one might argue that the random nature of the original game might make the game more (or less) tractable. To the ante, some variants of the game, such as *Evil2048* [11], use a heuristic to guess the worst possible location and value for the new tile at the end of every move. On the other hand, for our hardness proofs it might make sense to define a MAKE-TILE-ANGEL version, where the player can decide the value and location of the new tile after every move. However, as we will see, none of this makes the game significantly easier, as our NP-hardness proofs and inapproximability results hold for all variants mentioned, including ANGEL version.

We will also define optimization problems MAX-TILE, MAX-SCORE, and MAX-MOVES, whose objective is to maximize the value of the maximum tile created in the game, the total score of the game, defined as the sum of the values of all tiles created by merges, and the number of moves played before losing the game (reaching a forbidden configuration). These will be discussed in more detail in the section on inapproximability.

Finally, variants using different merging and movement rules, such as Threes! or Fibonacci will be defined and discussed after the main NP-hardness proof.

3 NP

In a blog post [4] Christopher Chen showed that 2048 is in NP. However to simplify the proof, they assume no new piece gets added to the board after a move. It turns out the proof for the regular game (ONLY-2 version) is not much harder.

The complexity analysis of every problem depends on a reasonable representation of the input. We here assume the input is provided in the form of b , the size of the board, and a list L of tiles present on the board at the beginning of the game. Thus the input size is $\log b + |L|$.

► **Lemma 1.** *For any constant value T , MAKE-TILE ONLY-2 is in NP.*

Proof. If the board is of size $b \times b$, then the maximum total value of all the tiles on the board without ever reaching T is $\leq Tb^2/2$. Since every move adds a tile with value 2, the total number of moves without reaching T is $\leq Tb^2/4$. If the number of tiles in the starting configuration is $\geq b$, then so is the input size, and the maximum number of moves reaching T , that is, the size of any yes certificate, is polynomial in the input size. If the number of tiles in the starting configuration is $< b$, then by the pigeonhole principle there is an empty row. Therefore playing down repeatedly will eventually, and repeatedly add new 2 tiles in that row which will accumulate to a single tile of value $2^b \geq T$ for b large enough. ◀

For the more general version without the ONLY-2 restriction, it is plausible that a similar strategy would work. An interesting question is whether the problem is still in NP if T is not a constant. The difficulty here is that the size of the input could be as small as $\log T$, and so number of moves would be exponential in that. A good first step would be to settle the question of what is the maximum tile value achievable on a $b \times b$ board in the game Det2048. For now, the highest value, even on a 4×4 board is unknown, the highest value was found using a heuristic algorithm [10].

4 NP-hardness

We will show NP-hardness of MAKE- T by reduction from 3SAT.

3SAT

Instance: Set U of variables, collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$.

Question: Is there a satisfying truth assignment for C ?

► **Theorem 2.** MAKE- T is NP-hard for any fixed T greater than 2048.

Proof. Reduction from 3SAT. From an arbitrary instance of 3SAT with n variables and m clauses, we construct a MAKE- T instance. The construction will ensure that only tiles of value 2 may be merged (into 4 tiles) except for two tiles of value $T/2$ that can be combined to obtain the target value T at the end of the game if and only if the 3SAT instance is satisfiable.

In order to facilitate the analysis of the game, the instances produced by the reduction will start with all cells of the board filled with tiles, and maintain this invariant after each move (hereafter named *fullness* invariant). In order to achieve this, we ensure that at most one pair of 2 tiles is adjacent on the board at all times (*one-move* invariant) and no other pair of identical tiles ever become adjacent during the game until the very last move. This forces the player to make a binary choice: left/right or up/down. In this manner, at the end of each move, only one cell next to a wall is freed, and so the game (in every known variant) will have to place a new tile in that exact cell.

We explain our construction by specifying the locations where tiles 2 are placed. Since we always construct gadgets by putting tiles 2 in pairs, we denote this by a pair of two 2D points (\cdot, \cdot) . In the subsequent figures, they will be represented by black dots on a 2D lattice plane. The rest of the board will be filled with a pattern of tiles that will prevent any accidental merges.

Sketch. The construction has three parts: the *variable* gadgets, the *literal* gadgets, and the *clause checking* gadgets. We place each variable gadget in a rectangle below the x axis in distinct rows and columns (we use negative y coordinates for ease of notation). The clauses will each take up 12 rows above the x axis, 4 for each literal. Each literal will lie in 4 of the rows of its clause and 3 of the columns of its variable. The variable gadgets will cause vertical (down) shifts in their columns, which will be transformed into horizontal (left) shifts in the rows of corresponding clauses by the literal gadgets. Finally, the clause checking gadgets to the right of the board will check, for each clause, that at least one of its rows has been shifted.

Base Pattern. We start the construction by filling the board with the repeating *base pattern* shown in Fig. 5. Assuming the bottom left cell is numbered $(0, 0)$, cell (i, j) of the base pattern contains the tile $2^{3(i \bmod 3) + (j \bmod 3) + 3}$. See Fig. 5.

The gadgets replace some of the cells by the tile 2. This will cause some shifts in rows and columns during the game as those tiles become adjacent. But because of the one-move invariant, at most one row or column shifts in each move. In order to avoid accidental merges between tiles of the base pattern, gadgets will be constructed in such a way that no row or column will ever be shifted more than once (avoiding unwanted merges between new tiles appearing in the same cell). Second, we will adjust the size of the board so that gadgets are

64	128	256	64	128	256	64	128
8	16	32	8	16	32	8	16
512	1024	2048	512	1024	2048	512	1024
64	128	256	64	128	256	64	128
8	16	32	8	16	32	8	16
512	1024	2048	512	1024	2048	512	1024
64	128	256	64	128	256	64	128
8	16	32	8	16	32	8	16

■ **Figure 5** Base pattern.

at a distance of at least 3 from the walls, to avoid new tiles to interfere with the 2 tiles of the gadgets. Furthermore, we will place gadgets in such a way that no two adjacent columns will ever be shifted. Likewise, we will ensure that no two adjacent rows will ever be shifted, except in one place in the clause checking gadget where the one-move invariant will have to be argued more carefully.

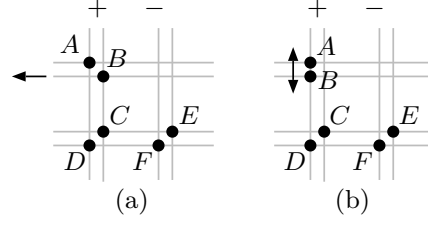
Ignoring this last case for now, notice that a tile can only be offset by one position horizontally and one position vertically during the entire game. Two tiles of same value are adjacent if the difference between their coordinates are $(0, 1)$ or $(1, 0)$, and any identical pair of tiles that are not on the same row or the same column start with coordinate difference at least $(3, 3)$, and thus can never become adjacent. If two identical tiles are on the same column, their vertical distance, starting at 3, would have to be reduced twice (using two opposite vertical shifts) in order for them to be at distance 1. However, since adjacent columns cannot be shifted in one game, this can only happen if they are at horizontal distance 2 at some point during the game, but since they started at horizontal distance 0, making that happen would already spend the 2 horizontal moves for those two tiles, making it impossible to bring them back close enough to become adjacent. The same argument applies to two identical tiles on the same row at distance 3.

Finally, consider the special case occurring in the clause checking gadget, where two adjacent rows are shifted. A similar case analysis will show that identical tiles on the same row could become adjacent, but only if the last move is vertical. A careful inspection of the gadget will reveal that no vertical move will occur within the shifted portion of those rows after the adjacent horizontal shift occurs.

Variables. For each variable gadget, we reserve 6 rows of the board, and 3 columns for each clause in which that variable appears. Assume, without loss of generality, that every variable appears at least once in the positive form (otherwise negate it).

Let k_i^+ and k_i^- be the numbers of clauses containing x_i and \bar{x}_i , respectively. We define the offset coordinates for each variable by

$$\begin{cases} X_0^V = 0, \\ X_i^V = X_{i-1}^V + 3(k_i^+ + k_i^-) + 7 \quad (1 \leq i \leq n) \end{cases} \quad \text{and} \quad Y_i^V = -6i \quad (0 \leq i \leq n-1).$$



■ **Figure 6** Variable gadget.

Now for each variable x_i ($i = 1, \dots, n$) we construct a gadget as follows. For a choice of true or false, we put a pair of tiles $\boxed{2}$ at

$$(A_i^V, B_i^V) = ((X_{i-1}^V + 1, Y_{i-1}^V + 1), (X_{i-1}^V + 2, Y_{i-1}^V)).$$

For the false part we place two pairs of tiles $\boxed{2}$ at coordinates

$$\begin{aligned} (C_i^V, D_i^V) &= ((X_{i-1}^V + 2, Y_{i-1}^V - 2), (X_{i-1}^V + 1, Y_{i-1}^V - 3)), \\ (E_i^V, F_i^V) &= ((X_{i-1}^V + 3k_i^+ + 5, Y_{i-1}^V - 2), (X_{i-1}^V + 3k_i^+ + 4, Y_{i-1}^V - 3)). \end{aligned}$$

So in the variable gadget for x_i , six tiles of value $\boxed{2}$ are placed as shown in Fig. 6(a) in general. The gadget is *activated* by pulling the row of B_i left (i.e., by merging a pair of adjacent $\boxed{2}$ on the row of B_i to the left). Now (see Fig. 6(b)), the tiles A_i and B_i become adjacent and on the same column. At this point the player has the choice to move this column containing A_i down (positive assignment to x_i and to make each clause containing literal x_i true), or up (negative assignment to make each clause containing literal \bar{x}_i true). If the player chooses to move up, then D_i moves up one cell and becomes adjacent to C_i , allowing the player to move left, causing E_i to move one cell left. The tile E_i is now adjacent and on the same column as F_i and the column of F_i can be moved down. Because there are no $\boxed{2}$ tiles below the variable gadget or to its left, any sequence of moves other than this one will cause the game to end.

Thus the x_i variable gadget has for effect to move down either the column of A_i (true), or that of F_i (false). This choice will be propagated through each corresponding literal gadgets.

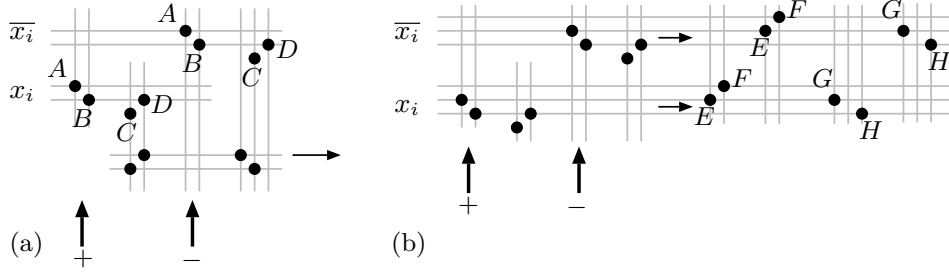
Literals. For literals in clauses we introduce the following coordinates:

$$\begin{cases} X_i^L = X_i^V & (0 \leq i \leq n), \\ X_{n+j}^L = X_{n+j-1}^L + 25 & (1 \leq j \leq m) \end{cases} \quad \text{and} \quad Y_j^L = 12(j-1) + 4 \quad (1 \leq j \leq m).$$

For variable x_i , suppose its positive literals x_i appear in the p_k -th position of the j_k -th clause ($p_k \in \{0, 1, 2\}$; $k = 1, \dots, k_i^+$; $1 \leq j_1 < \dots < j_{k_i^+} \leq m$). Remember that setting x_i to true will shift column $X_{i-1}^V + 1$ down. The gadget for the first positive literal x_i will receiving this vertical activation, shift one of its rows left and propagate the down move to activate the next literal. For this, we put two pairs of tiles $\boxed{2}$ at

$$\begin{aligned} (A_{j_k, p_k}^L, B_{j_k, p_k}^L) &= ((X_{i-1}^L + 3(k-1) + 1, Y_{j_k}^L + 4p_k + 1), (X_{i-1}^L + 3(k-1) + 2, Y_{j_k}^L + 4p_k)), \\ (C_{j_k, p_k}^L, D_{j_k, p_k}^L) &= ((X_{i-1}^L + 3(k-1) + 4, Y_{j_k}^L + 4p_k - 1), (X_{i-1}^L + 3(k-1) + 5, Y_{j_k}^L + 4p_k)). \end{aligned}$$

Likewise, when $k_i^- > 0$, negative literals \bar{x}_i appear in the p_k -th position of the j_k -th clause ($p_k \in \{0, 1, 2\}$; $k = 1, \dots, k_i^-$; $1 \leq j_1 < \dots < j_{k_i^-} \leq m$). For receiving and propagating



■ **Figure 7** Literal gadget.

vertical activations we put two pairs of tiles $\boxed{2}$ at

$$\begin{aligned} (A_{j_k, p_k}^L, B_{j_k, p_k}^L) &= ((X_{i-1}^L + 3(k_i^+ + k) + 1, Y_{j_k}^L + 4p_k + 1), (X_{i-1}^L + 3(k_i^+ + k) + 2, Y_{j_k}^L + 4p_k)), \\ (C_{j_k, p_k}^L, D_{j_k, p_k}^L) &= ((X_{i-1}^L + 3(k_i^+ + k) + 4, Y_{j_k}^L + 4p_k - 1), (X_{i-1}^L + 3(k_i^+ + k) + 5, Y_{j_k}^L + 4p_k)). \end{aligned}$$

See Fig. 7(a).

The horizontal (right) shifts for both positive and negative literals has for effect to move $\boxed{2}$ tiles placed to the right of the board, for use in the clause checking gadgets. We add those pairs of tiles $\boxed{2}$ at

$$\begin{aligned} (E_{j_k, p_k}^L, F_{j_k, p_k}^L) &= ((X_{n+j_k-1}^L + 6p_k + 1, Y_{j_k}^L + 4p_k + 1), (X_{n+j_k-1}^L + 6p_k + 2, Y_{j_k}^L + 4p_k + 2)), \\ (G_{j_k, p_k}^L, H_{j_k, p_k}^L) &= ((X_{n+j_k-1}^L + 3p_k + 16, Y_{j_k}^L + 4p_k + 1), (X_{n+j_k-1}^L + 3p_k + 18, Y_{j_k}^L + 4p_k)). \end{aligned}$$

See Fig. 7(b).

The final appearance of literals x_i and \bar{x}_i will also be represented by two pairs of tiles like above, but the second pair will cause a vertical shift up which will be propagated to activate the next variable gadget or to activate the clause checking gadgets as shown in Fig. 7(b). This process is described next.

Activate. The first variable gadget is activated by a pair of $\boxed{2}$ placed at

$$((-3, 0), (-2, 0))$$

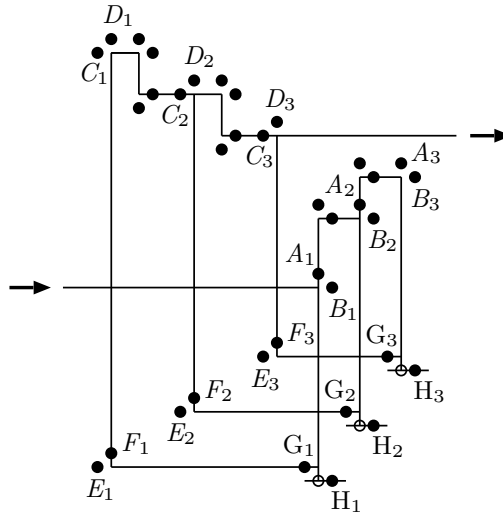
causing a horizontal shift left for B_1^V . For subsequent variables, assigning the truth value to the final literal x_i or \bar{x}_i ($1 \leq i \leq n-1$) will cause a vertical shift up at column $X_{i-1}^L + 3(k_i^+ - 1) + 4$ or $X_{i-1}^L + 3(k_i^+ + k_i^-) + 4$. Note that if $k_i^- = 0$, then it is E_i^V and F_i^V which will cause the vertical shift at that position. We propagate that shift into a horizontal left shift activating variable x_{i+1} using two pairs of tiles $\boxed{2}$ at

$$\begin{aligned} ((X_{i-1}^L + 3(k_i^+ - 1) + 4, Y_{i-1}^V - 7), ((X_{i-1}^L + 3(k_i^+ - 1) + 5, Y_{i-1}^V - 6)), \\ ((X_{i-1}^L + 3(k_i^+ + k_i^-) + 3, Y_{i-1}^V - 6), (X_{i-1}^L + 3(k_i^+ + k_i^-) + 4, Y_{i-1}^V - 7)). \end{aligned}$$

See the bottom four tiles of Fig. 7(a).

After the truth assignments of the literals of the last variable x_n or \bar{x}_n , one of the same columns is shifted, but this time down and that shift is propagated to activate the clause checking gadgets using two pairs of tiles $\boxed{2}$ at

$$\begin{aligned} ((X_{n-1}^L + 3(k_n^+ - 1) + 4, 12m + 5), (X_{n-1}^L + 3(k_n^+ - 1) + 5, 12m + 4)), \\ ((X_{n-1}^L + 3(k_n^+ + k_n^-) + 3, 12m + 4), (X_{n-1}^L + 3(k_n^+ + k_n^-) + 4, 12m + 5)). \end{aligned}$$



■ **Figure 8** Clause checking gadget.

Checking Clauses. For clause checking gadgets we take coordinates as follows:

$$X_j^T = X_{n+j}^L \quad (0 \leq j \leq m) \quad \text{and} \quad \begin{cases} Y_1^T = 12m + 12, \\ Y_j^T = Y_{j-1}^T + 15 \quad (1 < j \leq m). \end{cases}$$

For each clause C_j ($j = 1, \dots, m$) the corresponding gadget has for purpose to check that at least one literal of that clause has been set to true. To choose which of the three literals will be checked, we we place the following five pairs of tiles [2](#) at

$$\begin{aligned} (A_{j,1}^T, B_{j,1}^T) &= ((X_{j-1}^T + 17, Y_j^T - 7), (X_{j-1}^T + 18, Y_j^T - 8)), \\ ((X_{j-1}^T + 17, Y_j^T + 2), (X_{j-1}^T + 18, Y_j^T + 1)), \\ (A_{j,2}^T, B_{j,2}^T) &= ((X_{j-1}^T + 20, Y_j^T + 2), (X_{j-1}^T + 21, Y_j^T + 1)), \\ ((X_{j-1}^T + 20, Y_j^T + 5), (X_{j-1}^T + 21, Y_j^T + 4)), \\ (A_{j,3}^T, B_{j,3}^T) &= ((X_{j-1}^T + 23, Y_j^T + 5), (X_{j-1}^T + 24, Y_j^T + 4)). \end{aligned}$$

The pair $(A_{j,1}^T, B_{j,1}^T)$ is activated by shifting the row $Y_j^T - 8$ left. Then either the move up shifts the column $X_{j-1}^T + 17$ up, or the sequence down, left, up shifts the column $X_{j-1}^T + 20$ up, or the sequence down, left, down, left, up shifts column $X_{j-1}^T + 23$ up. Any other sequence of moves ends the game. Note that the column of $A_{j,p}^T$, $p = 1, 2, \text{ or } 3$ being shifted up is exactly one column left of the one containing the [2](#) at $H_{j,p}^L$ at the beginning of the game. If the corresponding literal was set to true in the literal gadget, then the [2](#) had been shifted left and is now shifted up, bringing $G_{j,p}^L$ and $H_{j,p}^L$ next to each other. The row of $G_{j,p}^L$ can now be shifted left, activating the pair $(E_{j,p}^L, F_{j,p}^L)$, and the column of $F_{j,p}^L$ can now be shifted down.

To collect the down shift in the column of $F_{j,p}^L$ for the chosen $p = 1, 2, \text{ or } 3$, we place seven pairs of tiles [2](#) at coordinates

$$\begin{aligned} (C_{j,1}^T, D_{j,1}^T) &= ((X_{j-1}^T + 1, Y_j^T + 13), (X_{j-1}^T + 2, Y_j^T + 14)), \\ ((X_{j-1}^T + 4, Y_j^T + 14), (X_{j-1}^T + 5, Y_j^T + 13)), & ((X_{j-1}^T + 4, Y_j^T + 9), (X_{j-1}^T + 5, Y_j^T + 10)), \end{aligned}$$

$$\begin{aligned}
(C_{j,2}^T, D_{j,2}^T) &= ((X_{j-1}^T + 7, Y_j^T + 10), (X_{j-1}^T + 8, Y_j^T + 11)), \\
((X_{j-1}^T + 10, Y_j^T + 11), (X_{j-1}^T + 11, Y_j^T + 10)), &((X_{j-1}^T + 10, Y_j^T + 6), (X_{j-1}^T + 11, Y_j^T + 7)), \\
(C_{j,3}^T, D_{j,3}^T) &= ((X_{j-1}^T + 13, Y_j^T + 7), (X_{j-1}^T + 14, Y_j^T + 8)).
\end{aligned}$$

Now the vertical shift of column $F_{j,p}^L$ aligns the $\boxed{2}$ tiles of $C_{j,p}^T$ and $D_{j,p}^T$, and the rest of the $\boxed{2}$ can be used to propagate the horizontal shift until the row of $C_{j,3}^T$ is shifted left. This is the same row as $B_{j+1,1}^T$ and so activates the next clause checking gadget for $j < m$.

Goal. To make a target number X (> 2048), we place a pair of tiles of value $X/2$ at $((X_{n+m}^T + 1, Y_m^T + 8), (X_{n+m}^T + 2, Y_m^T + 7))$. This pair will become adjacent when the last clause checking gadget is successfully played and shifts the row of $C_{m,3}^T$ left.

From the construction, it is clear that the tile \boxed{T} can be created if and only if the given 3SAT formula is satisfiable. The size of the board is $\Theta((n+m)^2)$, and the size of the sequence of new tiles is $\Theta(m+n)$. So this reduction takes polynomial space and polynomial time with respect to the input size $n+m$ of the 3SAT instance. \blacktriangleleft

We illustrate a complete example of our reduction in Fig. 9, where the formula for 3SAT is $f = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$. In this figure, two goal tiles $T/2$ are represented by squares.

5 Inapproximability

It is fairly easy to extend the construction from the previous section to show it is NP-hard to approximate MAX-TILE, MAX-SCORE or MAX-MOVES. For MAX-TILE and MAX-SCORE, it would be enough to change the value of the goal pair of tiles to an arbitrarily high number. However one might want to impose that the tiles of the input configuration be all of small value. In that case, we can still show inapproximability by using the *pot of gold* technique.

Note that in the previous construction, if the formula is satisfiable, then the goal tiles will be adjacent in column $X_{n+m}^T + 1$. We add tiles:

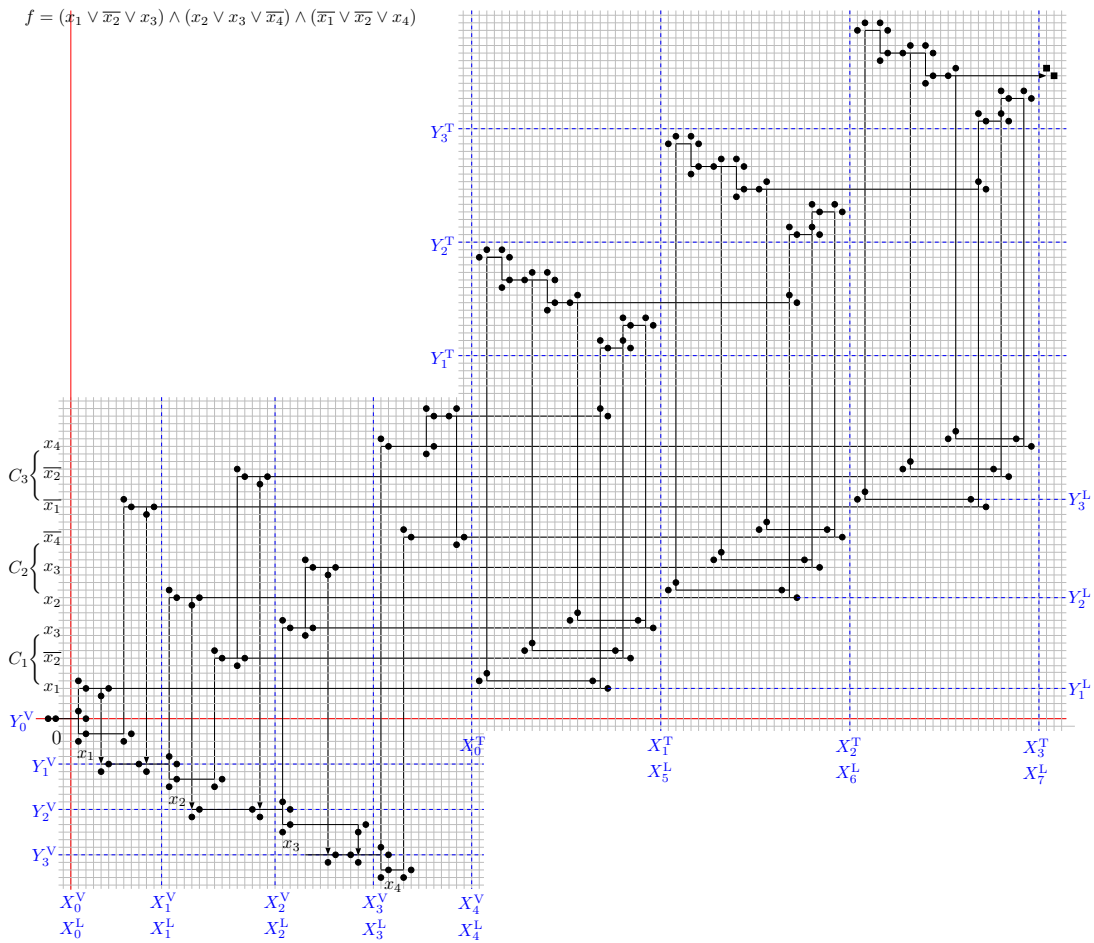
$$(A^A, B^A) = ((X_{n+m}^T + 1, Y_m^T + 21), (X_{n+m}^T + 2, Y_m^T + 20)).$$

We then extend the board to the left of the first variable gadget by $K = 2^p$ columns, and place tiles of alternating values $\boxed{8}$ and $\boxed{16}$ on row $Y_m^T + 20$ at negative x coordinates. On the row $Y_m^T + 19$, we place tiles of alternating values $\boxed{32}$ and $\boxed{8}$ (so the tiles $\boxed{8}$ are just below the tiles $\boxed{16}$ and can't merge).

If the formula is satisfiable (and only then), the player can solve the game as before until she shifts column $X_{n+m}^T + 1$ down. One can then shift the row of B^A left which aligns all the $\boxed{8}$ of that row with the $\boxed{8}$ of the row below. A move up now merges all those $\boxed{8}$ s into $\boxed{16}$ s, and repeatedly shifting right $p = \log K$ times will merge all those tiles into one tile of value $16K$. We can then continue the sequence with $S = 2^q \boxed{2}$ appearing in the leftmost cell of row $Y_m^T + 20$, with $q < K$. The total score is then $\Theta(m+n+K+S)$ and the maximum tile is $\Theta(\max(K, S))$.

The input size in this game is the entire size of the board plus the length of the tile sequence and all tiles are of constant value. The original board size is $\Theta((n+m)^2)$ so the augmented board is of size $\Theta((n+m)(n+m+K))$. The number of moves is $\Theta(n+m+\log K+S)$.

So in the standard game:



■ **Figure 9** An example of NP-hardness reduction from 3SAT to MAKE-T .

- Taking $K = n + m$, $S = K^3$, the input size is $N = K^3 + \Theta(K^2)$. The maximum tile value is $S = N - O(N^{2/3})$ if the formula is satisfiable, 2048 otherwise. So, it is NP-hard to approximate MAX-TILE within a factor N/c for some constant c .
- Taking $K = n + m$, $S = 2^K$, the input size is $N = 2^K + \Theta(K^2)$. The maximum score is $S = N - O(\log^2 N)$ if the formula is satisfiable, $O(K) = O(\log N)$ otherwise. So it is NP-hard to approximate MAX-SCORE within a factor $o(N/\log N)$.
- Using the same parameters as above, the maximum number of moves is at least $S = N - O(\log^2 N)$ if the formula is satisfiable, $O(K) = O(\log N)$ otherwise. So it is NP-hard to approximate MAX-MOVES within a factor $o(N/\log N)$.

Note the importance of S in the input size N for the standard version of the game. In the game DET2048, however, the sequence of new tiles is implicit and not part of the input. The inapproximability results are then strengthened: all three problems are inapproximable within a factor $o(2^N)$ or $o(2^N/N)$.

6 Variants

Only minor modifications are required to make the NP-hardness reduction work for most known variants of Threes! and 2048. We just describe them for the original game Threes!,

and for the Fibonacci version mentioned in the introduction. The extension of these results to other variants such as Fives, 1024! and Saming's 2048 are immediate and are left as an exercise to the reader.

6.1 Threes!

The reduction for Threes! is nearly identical as for 2048. The easiest way to repeat the proof would be to replace every tile of value 2^a by a tile of value $3 \cdot 2^{a-1}$. A slightly better bound can be obtained in the following manner. Every occurrence of tile **2** is replaced by a **3**. The base pattern uses only tiles **1**, and the new tiles added after each move are **1**. Since a **1** can only merge with a **2**, this will ensure the base pattern never causes an unwanted merge. The goal tiles are replaced by two **6**.

Recall that tiles in Threes! move according to slightly different rules (most importantly, every tile stays in place, shifts to an adjacent cell or merges with an adjacent tile in every move). However, because of the fullness and one-move invariants, the result of a move will be the same in Threes! as it was for 2048.

Therefore, an identical proof shows that it is NP-hard to decide if it is possible to achieve tile **12** in Threes!. The inapproximability results for MAX-TILE and MAX-MOVES extend as well. For MAX-SCORE, the situation is even worse, as following the same reduction ending it with a sequence of $S = 2^q$ tiles **3**, we would produce a tile of value $3 \cdot 2^q$ which will produce a score of $3^{q+1} = 3S^{\log_2 3} = \Omega(N^{\log_2 3})$ if the formula is satisfiable, and $O(K) = O(\log N)$ otherwise. Therefore, it is NP-hard to approximate MAX-SCORE in Threes! within a factor $o(N^{\log_2 3} / \log N)$.

6.2 Fibonacci

Denote the i -th Fibonacci number by F_i , that is, $F_1 = F_2 = 1$, and $F_{i+2} = F_{i+1} + F_i$. In the Fibonacci version, tiles merge only if they are adjacent in the Fibonacci sequence.

We modify the reduction so that every occurrence of tile **2** is replaced by a 1 (since $F_1 = F_2 = 1$, they can merge into a 2 when adjacent). The base pattern uses only tiles of value 5, and the new tiles added after each move are 1. Since a 5 can only merge with a 3 or 8, this will ensure the base pattern never causes an unwanted merge. The goal tiles are replaced by 13 and 21. Therefore is NP-hard to decide if it is possible to achieve tile 34. Inapproximability results extend as well.

References

- 1 Ahmed Abdelkader, Aditya Acharya, and Philip Dasler. On the complexity of slide-and-merge games. *CoRR*, abs/1501.03837, 2015. URL: <http://arxiv.org/abs/1501.03837>, arXiv:1501.03837.
- 2 Ahmed Abdelkader, Aditya Acharya, and Philip Dasler. 2048 without new tiles is still hard. *Proceedings of the 8th International Conference on Fun with Algorithms*, LIPICS volume 49, 2016.
- 3 Ron Breukelaar, Erik D. Demaine, Susan Hohenberger, Hendrik Jan Hoogeboom, Walter A. Kosters, and David Liben-Nowell. Tetris is hard, even to approximate. *International Journal of Computational Geometry and Applications*, 14(1-2):41-68, 2004.
- 4 Christopher Chen. 2048 is in NP. <http://blog.openendings.net/2014/03/2048-is-in-np.html>, March 2014.
- 5 Gabriele Cirulli. 2048. <http://gabrielecirulli.github.io/2048/>, March 2014.

22:14 Threes!, Fives, 1024!, and 2048 are Hard

- 6 Erik D. Demaine and Robert A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In Michael H. Albert and Richard J. Nowakowski, editors, *Games of No Chance 3*, volume 56 of *Mathematical Sciences Research Institute Publications*, pages 3–56. Cambridge University Press, 2009. [arXiv:cs.CC/0106019](#).
- 7 Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer Heidelberg, 1999.
- 8 Rahul Mehta. 2048 is (PSPACE) hard, but sometimes easy. *CoRR*, abs/1408.6315, 2014. URL: <http://arxiv.org/abs/1408.6315>, [arXiv:1408.6315](#).
- 9 Phenomist. 2048 variants. <http://phenomist.wordpress.com/2048-variants/>, 2014.
- 10 QuadmasterXLII. Solve a deterministic version of 2048 using the fewest bytes. <http://codegolf.stackexchange.com/questions/24885/solve-a-deterministic-version-of-2048-using-the-fewest-bytes>, 2014.
- 11 A.J. Richardson. Evil 2048. <http://aj-r.github.io/Evil-2048/>, March 2014.
- 12 Saming. 2048. <http://saming.fr/p/2048/>, March 2014.
- 13 Asher Vollmer, Greg Wohlwend, and Jimmy Hinson. Threes! <http://asherv.com/threes/>, January 2014.