TwoDots is NP-Complete

Neeldhara Misra[1]

1 **Indian Institute of Technology, Gandhinagar**
mail@neeldhara.com

Abstract

TwoDots is a popular single-player puzzle video game for iOS and Android. In its simplest form, game consists of a board with dots of different colors, and a valid move consists of connecting a sequence of adjacent dots of the same color. We say that dots engaged in a move are "hit" by the player. After every move, the connected dots disappear, and the void is filled by new dots (the entire board shifts downwards and new dots appear on top). Typically the game provides a limited number of moves and varying goals (such as hitting a required number of dots of a particular color). We show that the perfect information version of the game (where the sequence of incoming dots is known) is NP-complete, even for fairly restricted goal types.

1998 ACM Subject Classification F.2 Analysis Of Algorithms And Problem Complexity

Keywords and phrases puzzle, NP-complete, perfect information, combinatorial game theory

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

TwoDots® (http://weplaydots.com/twodots.html) is a puzzle game that came out in May 2014 on the iOS and Android platforms. Having surpassed 30 million downloads after just about a year of launch, it is a popular game that is widely accepted as addictive and frustrating in roughly equal measure. In this work, we set out to investigate the computational complexity of playing this puzzle. In its present form, it is a single-player game, and we will not encounter any strategic questions.

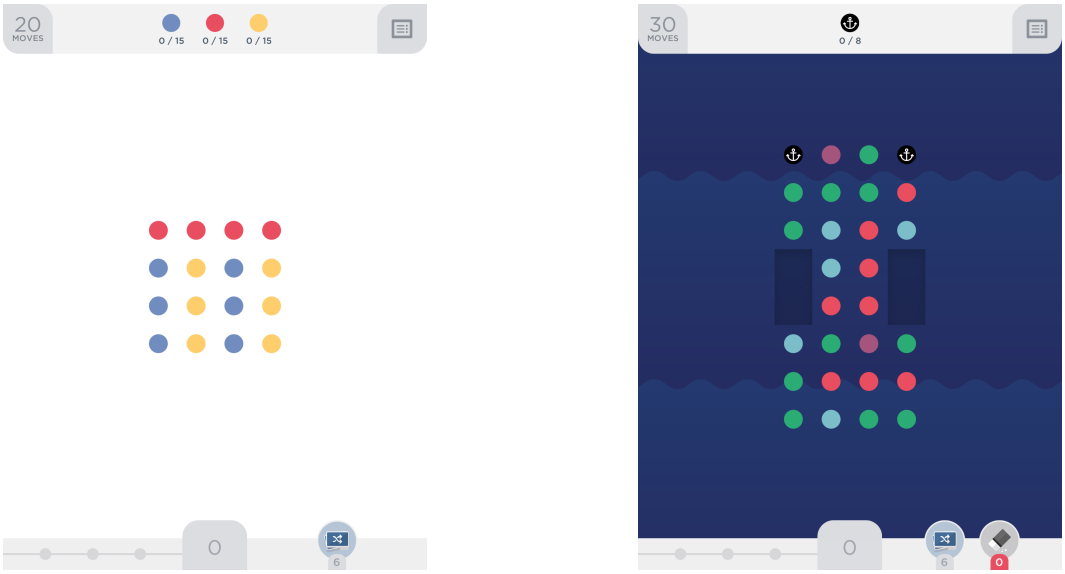
The game arena consists of a grid, and at each location there is a colored dot. Dots of the same color can be "connected" by the player, as long as they are adjacent horizontally or vertically (but never diagonally). When dots are connected, they disappear, and the remaining dots fall down as if influenced by gravity. The voids on top are filled with fresh dots. The game provides a certain number of moves, and demands that certain goals should be met (which are typically of the form of collecting at least so many dots of such and such a color, where a dot of a particular color is collected whenever the player connects dots of that color).

An interesting move is the square, wherein, if there are four dots of the same color arranged in a square like configuration (with no gaps, see Figure 3), then swiping across the square causes all dots of said color to disappear. It turns out that this move is frequently a game-changer, and plays an important role in our results too. It is clearly a popular heuristic, and the official TwoDots tutorial even offers the helpful quip: *"When in doubt, make squares"*.

Having spent several frustrating hours with the game of TwoDots, and making limited progress through the increasingly challenging levels, the author was compelled to ask if

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) Level One of Two Dots. **(b)** A more advanced level of Two Dots.

Figure 1 Screenshots from the Two Dots game. The top-left corner shows the total number of moves allowed at the start of the game, and the top-middle shows the target goals. The bottom-middle tracks the score after every move, an aspect that will not be relevant to our present study.

finding a winning sequence of moves that meets all the goals is an easy problem, even in the unrealistic but optimistic perfect-information setting, where we know exactly what dots are coming up. We discover that the dots from the future are quite irrelevant, and the gameplay even on just the starting board can get quite intricate, especially since the newly arriving dots can be designed to be of no help to the stated goals. After setting up the formal decision version of the TwoDots game, we establish the following:

- The game is NP-complete, even when the board has only four rows. (Theorem 3.1.)
- The game is W[1]-hard when parameterized by the number of moves. (Corollary 3.3.)
- The game is NP-complete even when played without the square move. (Theorem 4.1.)
- The game remains NP-complete when there is only one goal of collecting two dots of a particular color, even when there is no restriction on the number of moves. (Theorem 4.2.)

The main objective of this work is to provide reassurance to those TwoDots addicts who might occasionally wonder if the game is truly hard. Of course, we note that the literature is rich in the analysis of the algorithmic aspects of combinatorial games and puzzles. More specifically, games that are popularized by mobile platforms have drawn a lot of attention in the recent past. This work is inspired by, and is in the spirit of some of these developments, which include showing the hardness of popular games like Flow [1], 1024 [6] and CandyCrush [7].

Organization of this Paper. After setting up the formulation of TwoDots as a decision problem in the next section, we dedicate Section 3 to a proof of Theorem 3.1, and Section 4 to showing hardness in the more restricted scenarios.

2 Preliminaries

The game of TwoDots gets more challenging as the levels progress, largely due to the addition of new goal types and harder obstructions. Players of the game will recognize issues like ice-breaking, containing fire, dealing with monsters, anchors, slime, bombs and so forth. However, in the interest of keeping the exposition straightforward, we will only deal with the minimalistic version of the game, which we will formulate in this section. This approach also leads us to the conclusion that TwoDots is hard even in its more rudimentary versions. We will also ignore the scoring systems and pose the game as a decision problem.

2.1 A Game of TwoDots

An instance of TwoDots consists of the following[1]:

1. A $(m \times n)$ grid $\mathcal{B}$, a set of colors $\mathcal{C}$, and a mapping f from $\mathcal{B}$ to $\mathcal{C}$.
2. A natural number k, specifying the number of moves allowed in the game.
3. A set of goals $\mathcal{G}$. Every element of $\mathcal{G}$ is a pair (c, ℓ), where $c \in \mathcal{C}$ and $\ell \in \mathbb{N}$.
4. A sequence of colors σ, representing the dots that will fill the voids created by the moves.

We note that the map f in (1) above need not be injective, in other words, multiple locations on the board may be mapped to the same color. The semantics of the pairs given by (3) is that ℓ dots of color c need to be "hit", a notion that we will define in a moment. Summarizing the above, an instance of TwoDots is fully specified by the following tuple:

$$\mathfrak{D} := \langle f : [n] \times [m] \longrightarrow \mathcal{C}, k, \mathcal{G} = \{(c_1, \ell_1), \cdots, (c_g, \ell_g)\}, \sigma \rangle,$$

where n, m, k and the ℓ_i's are natural numbers, $\mathcal{C}$ is a finite set, the c_i's belong to $\mathcal{C}$ and σ is a sequence whose entries are from $\mathcal{C}$. Following tradition, we will say that every location on the board is occupied by a *dot*, and the colors of these dots is given by the function f. Also, our array indexing starts at one, and rows are counted from bottom to top, and columns from left to right.

Intuitively, a player has a winning strategy in an instance of TwoDots if all the goals can be achieved with k moves, given (or despite) the dots that join the board according to σ. To formalize this, we need to first define moves, and the notion of dots being hit.

There are two types of moves in TwoDots: *regular moves* and *square moves*. We first describe the regular moves, which essentially involve removing paths in the grid occupied by the same color. To define this formally, we will need the notion of adjacent locations and valid sequences.

▶ **Definition 2.1.** Two dots at locations (a, b) and (c, d) are said to be *adjacent* if either $a = c$ and $b = d \pm 1$ or $b = d$ and $a = c \pm 1$, that is, if one of them is to the top, right, left, or bottom of the other. A sequence of locations $\langle t_1, \ldots, t_s \rangle$ is said to be *valid* if t_i is adjacent to t_{i+1} for all $1 \leq i \leq s - 1$, and further, $f(t_i) = f(t_{i+1})$ for all $1 \leq i \leq s - 1$.

[1] Readers unfamiliar with the game may find the definitions in this section either simpler or unnecessary after playing it for some time!

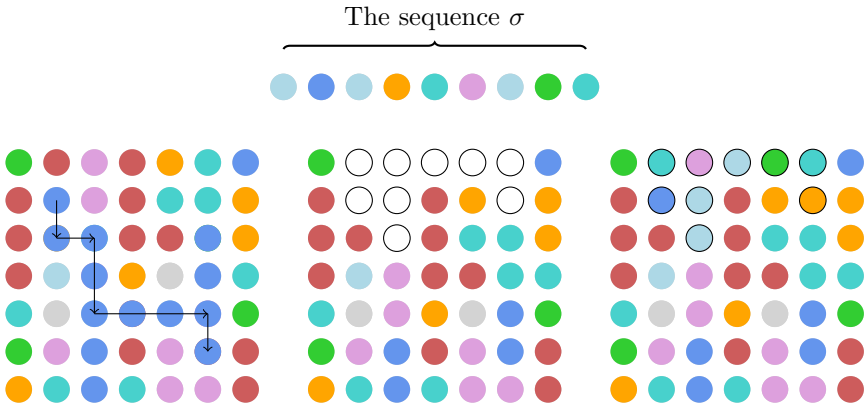


Figure 2 A depiction of the regular move. The first panel shows the set of locations that are committed to the move, the second panel shows the voids created, and the third panel shows how the voids are filled in accordance with σ.

Note that dots aligned diagonally are *not* considered adjacent. A player can use any valid sequence of locations towards a regular move. If a valid sequence had s locations and these locations were occupied by dots of color c, then using this sequence causes the color c to be *hit* s times. Some readers may prefer imagining the player collecting s dots of color c when the sequence is committed. This also creates voids in the locations corresponding to the sequence. The dots "fall down" to fill out the voids — it is useful to think of the board as a vertically oriented object, and the dots therein following the natural laws of gravity. Of course, this simply pushes the voids to the top, which are the filled out with new dots, which are colored according to the first s colors given by the sequence σ. We refer the reader to Figure 2 for an illustration.

Summarizing the above more formally, a regular move given by a valid sequence of locations $\langle t_1, \ldots, t_s \rangle$ causes the following changes:

1. Let c be the color of the dots in the given sequence. Suppose $\mathcal{G}$ contains a pair (c, ℓ). If $s \geq \ell$, the pair (c, ℓ) is purged from $\mathcal{G}$, else the pair is updated to $(c, \ell - s)$.
2. The function f is updated as follows. For every location $t_i = (a_i, b_i)$, we reset $f(a_i, b_i) = f(a_i + h, b_i)$, where h is the largest number such that every location in:

$$\langle (a_{i+1}, b_i), \ldots (a_{i+j}, b_i), \ldots (a_{i+h-1}, b_i) \rangle$$

   is a part of the player's sequence.
3. The updated version of f from the previous step is not well-defined for s locations. These locations are assigned to the first s colors from σ. The voids are filled up by σ in a bottom-to-top, left-to-right order. The sequence σ is reset to the subsequence $\sigma[s+1, :]$.[2]

We now turn to a square move (see Figure 3). If a player identifies four adjacent locations in the form of a square, for instance:

$$\langle (i, j), (i+1, j), (i, j+1), (i+1, j+1) \rangle,$$

[2] This is Pythonic notation, we simply mean here that the first s elements of σ are "chopped off" from the sequence.

such that the said locations are all occupied by dots of the same color, say c, then committing the square causes all dots of color c on the board to be hit. In other words, committing a square replaces every location that has a dot colored c with a void, and these voids are filled just as with a regular move. The color c is hit as many times as it appears on the board, and the goal corresponding to c (if there is one) is updated in the same way as with the regular move. In practice, square moves are handy — they tend to clear out large parts of the board and are often necessary for certain goal types. This move turns out to be crucial to our reductions as well.

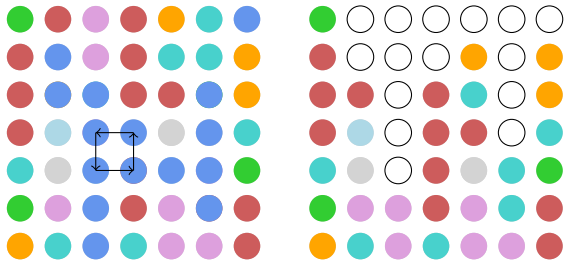


Figure 3 A depiction of the square move, which has the effect of eliminating all the blue dots from the board. The example is rather similar to the above, but note the difference in the number of voids created. The process for filling up the voids is identical, and is therefore omitted.

At the end of k moves, when the game is over, we say that the player has won if the set of goals is empty. The computational problem at the heart of our discussions is the following:

TwoDots
Given an instance of TwoDots, determine if there exists a sequence of at most k moves that meets all the goals.

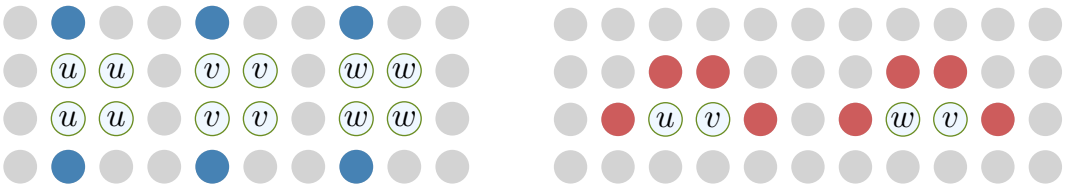
▶ Remark. The sequence σ is important to maintain the invariant of a full board, and the reader may legitimately worry about whether σ is long enough to sustain the voids that can be created by k moves. We assume that if σ falls short, then all future voids are filled by dots of distinct colors and colors that are disjoint from the ones of the board (it is useful to think of these as "dummy colors"). Indeed, in our reductions the "dots from the future" will not be important, and thus we will simply provide a null sequence, and it will be apparent that the new dots will have no impact on gameplay.

2.2 Other Definitions

In this subsection we briefly recall the definitions of the problems that we will be using to show NP-hardness. We refer the reader to [5] for an introduction to reductions, and to [3] for a survey of complexity-theoretic studies of combinatorial games.

Satisfiability. A *literal* is a propositional variable x or a negated variable $\overline{x}$. A *clause* is a collection of literals. A propositional formula in conjunctive normal form, or *CNF formula* for short, is a set of clauses. A CNF formula F is *satisfiable* if there is some truth assignment to the variables that satisfies all the clauses, where a clause is satisfied if at least one of its literals evaluates to true. The special case of 3-SAT, where every clause has at most three literals, is also well-known to be NP-complete.

Exact Cover. An instance of Exact Cover by 3-Sets, abbreviated X3C, consists of an universe $\mathcal{U} = \{u_1, \ldots, u_n\}$ and a family of sets $\mathcal{F} = \{S_1, \ldots, S_m\}$, where each set has three elements. The question is if there is a set cover of size $(n/3)$, that is, a subcollection



(a) The Vertex Gadgets. **(b)** A Tester Gadget for the edges (u, v) and (w, v).

Figure 4 The vertex and tester gadgets used in the reduction from Clique. The colors α and β are depicted, respectively, by blue and red.

$\mathcal{G} \subseteq \mathcal{F}$ such that each element of $\mathcal{U}$ appears exactly once among the sets of $\mathcal{G}$. The problem is well-known to be NP-complete.

Clique. An instance of Clique consists of a graph $G = (V, E)$ and a positive integer k. The question is if there exists a subset S of at least k vertices that form a clique, that is, for every pair of vertices (u, v) such that u and v belong to S, we have that $(u, v) \in E$. This is also a classic NP-complete problem.

3 Hardness in the basic setting

In this section we show the NP-hardness of TwoDots, by a reduction from Clique.

Overview of the Reduction. If n is the number of vertices in the instance of Clique, then we are going to have $(n+2)$ colors for the dots (apart from several more dummy colors). The instance of TwoDots that we generate has two main components:

1. For every $1 \leq c \leq n$, we will have a square on dots of color c, as shown in Figure 4a. These will be the *vertex selector* gadgets.
2. For every edge e, we will have a gadget as shown in Figure 4b, which lapses into something useful only when both vertices involved in the edge e are picked for square moves. These are our *tester gadgets*, together with a carefully defined goal they ensure that the colors chosen for square moves correspond to vertices that form a clique in G.

We allow for $2k + \binom{k}{2}$ moves and set a goal of hitting at least $2k$ dots of color $(n+1)$ (which is the color used in the vertex gadgets), and $4\binom{k}{2}$ dots of color $(n+2)$ (which is the color used in the tester gadgets). It turns out that the only way to achieve these goals is to apply square moves on colors corresponding to the vertices of a clique, and regular moves on the tester gadgets corresponding to the edges of the clique. We now turn to a more formal description of the construction, and the correctness.

The Construction. Let (G, k) be the instance of Clique, and suppose $G = (V, E)$ where $V = \{v_1, \ldots, v_n\}$, and $E = \{e_1, \ldots, e_m\}$. We now describe the instance of TwoDots. The board has four rows and $(3n + 5m)$ columns. The set of colors is as follows.

- We have a set of $4(3n + 5m)$ dummy colors, one for every location of the board.
- We introduce the color c_i corresponding to the vertex v_i, for all $1 \leq i \leq n$.
- We introduce two additional special colors, denoted by α and β.

The mapping f is given by the following, wherein we place squares corresponding to vertices in the first $3n$ columns and the tester gadgets in the remaining columns, all next to each other.

- On the second and third rows, columns $(3i-1)$ and $3i$ are occupied by dots colored c_i. On the first and fourth rows, column $(3i-1)$ is occupied by the color α.
- Suppose the edge $e_j = (v_p, v_q)$. On the second row, the two columns $(3n+5j-2), (3n+5j-3)$ are occupied by dots colored c_p and c_q, respectively.
- Further, on the second row, the columns $(3n+5j-4), (3n+5j)$ are occupied by dots colored β. On the third row, the two columns $(3n+5j-2), (3n+5j-3)$ are occupied by dots colored β.
- On any location (i, j) that is not accounted for by the above, we use a dot with the dummy color corresponding to that location.

We conclude the description of the instance by fixing the number of moves and the goals. The number of moves is given by $2k + \binom{k}{2}$, and we specify two goals, namely (α, k) and $(\beta, 4\binom{k}{2})$. The sequence σ is the null sequence (voids are filled by fresh and distinct colors). This completes the construction.

In the discussion that follows, we say that a tester gadget corresponding to the edge $e = (v_p, v_q)$ has *collapsed* if square moves were executed corresponding to *both* c_p and c_q. Note that once a tester gadget has collapsed, we can hit four dots colored β with one regular move.

The Forward Direction. Suppose S is a clique of size k in G. We execute square moves corresponding to all vertices in S, using up k moves. This makes k pairs of dots of color α adjacent, and we spend the next k moves in eliminating these. Since S was a clique, the square moves also cause $\binom{k}{2}$ of the tester gadgets to collapse (one corresponding to each edge of the clique). The remaining $\binom{k}{2}$ moves are regular moves on the collapsed gadgets, where each move hits four dots colored β, thereby fulfilling the given goal.

The Reverse Direction. The following is immediate from the two goals:

- At least $2k$ moves must be spent in eliminating $2k$ dots of color α, since the configuration of the board forces dots of color α can only be eliminated in pairs, and the only way to make a pair adjacent is to use up a square move on some c_i, $1 \leq k \leq n$.
- At least $\binom{k}{2}$ of the tester gadgets must collapse because of the square moves, otherwise it is not possible to achieve the goal corresponding to β in $\binom{k}{2}$ moves.

Let S be the set of those $1 \leq i \leq n$ for which a square move was executed on color c_i. It is clear that $|S| \geq k$, since anything less will fail the goal with respect to color α. We claim that the set of vertices corresponding to S forms a clique in G. Indeed, note that only those tester gadget collapse that correspond to edges with both endpoints in S. Any missing edge in S will imply that fewer than $\binom{k}{2}$ tester gadgets collapsed, which contradicts the goal with respect to the color β.

This completes the proof of correctness and leads us to our first theorem.

▶ **Theorem 3.1.** TwoDots *is NP-complete.*

We remark that membership in NP is easy to show, since a sequence can be used as a certificate. We now turn to some implications of Theorem 3.1. The first one follows from the fact that our reduction used only four rows. We note that there is no obvious way of

"flipping" the construction to derive an analogous result on the number of columns, since the gameplay is inherently asymmetric. For instance, if the tester gadgets corresponding to edges were to be "stacked" one on top of another, then a single square move could cause some unintentional collapses.

▶ **Corollary 3.2.** TwoDots *remains NP-complete even when the number of rows is a constant.*

Since Clique is W[1]-hard when parameterized by solution size, we also have the following comment on the parameterized complexity of TwoDots when parameterized by the number of moves. We refer the reader to [2] for terminology related to parameterized complexity.

▶ **Corollary 3.3.** TwoDots *is W[1]-hard when parameterized by the number of moves.*

There are (at least) two aspects of this reduction that the reader might find troublesome, or to propose a euphemism, two things that might be identified as providing the source of hardness. One is the convenience of controlling far-out parts of the board via the square move, and the other is the fact that both goals specify a target that is a function of the input size, rather than, say, a constant. Addressing the above, in the next section we show that the hardness persists even if only regular moves were available to the player, and, in a separate result, we establish NP-completeness with only one constant-sized goal.

4 Some Restricted Scenarios

In this section, we describe reductions showing hardness in more restricted scenarios: the first is when the square move is not available to the player, and the second involves only one constant-sized goal. We point out that some charms are lost in both cases: with only regular moves, we encounter a non-constant number of goals of non-constant size, and in the case of the constant sized goal, we are faced with a rather large board. This renders all the three reductions mutually non-subsuming.

4.1 Hardness without the Square Move

Here we propose a simple reduction from X3C, which we recall is the problem of Exact Cover by 3-Sets. We begin with the construction. Let $(\mathcal{U}, \mathcal{F})$ be an instance of X3C, and further, suppose $\mathcal{U} = \{u_1, \ldots, u_n\}$ and $\mathcal{F} = \{S_1, \ldots, S_m\}$. For our instance of TwoDots, we will have a board with three rows and $6m$ columns, and $(n+1) + 18m$ colors. We use $c_1, \ldots, c_n$ and ★ to denote the first $(n+1)$ colors, and the remaining are simply identified as dummy colors (one associated with each location of the board).

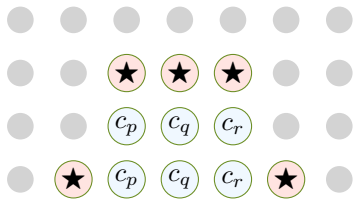


Figure 5 The gadget corresponding to a set $S = \{u_p, u_q, u_r\}$, from an instance of X3C.

We now turn to the map f. We have one block of six columns for every $S_i \in \mathcal{F}$, as seen in Figure 5. More formally, we have:

- For columns $6i - 4$ and $6i$, for all $1 \leq i \leq m$, in the first row we have dots colored ★. For the columns $6i - 3, 6i - 2$ and $6i - 1$, on the third row, we have dots colored ★.

- Suppose the set S_i consists of the elements u_p, u_q, u_r. Then on the first and second rows of columns $6i-3, 6i-2$ and $6i-1$, we have dots colored c_p, c_q and c_r, respectively (see Figure 5).
- On any location (i, j) that is not accounted for by the above, we use a dot with the dummy color corresponding to that location.

We now set the following goals. For every $1 \leq i \leq n$, we have the goal $(c_i, 2)$, and additionally, we have the goal $(\bigstar, 5(n/3))$. Also, the number of moves allowed is $n + n/3$, and σ is the null string. This completes the description of the instance, and we now turn to the proof of correctness. For ease of discussion, we say that the gadget corresponding to a set $S = \{u_p, u_q, u_r\}$ has collapsed if the dots colored c_p, c_q, c_r belonging to that gadget were hit by regular moves.

The Forward Direction. Suppose $\mathcal{G} \subseteq \mathcal{F}$ is a collection of $(n/3)$ sets that covers each element of the universe exactly once. For each $1 \leq i \leq n$, let $g(i)$ denote the set from $\mathcal{G}$ that contains i. We hit the dots c_i in the gadget corresponding to the set $g(i)$. Note that so far we have used n moves, satisfied the goals corresponding to c_i and exactly $(n/3)$ set gadgets (corresponding to the ones in $\mathcal{G}$) have now collapsed. We use the remaining $(n/3)$ moves to clear out the $\bigstar$ rows in the collapsed gadgets, thereby meeting the remaining goal.

The Reverse Direction. Note that to meet the goals corresponding to the colors c_i, $1 \leq i \leq n$, we must use at least n moves. Since there are only $(n/3)$ moves remaining for the goal corresponding to $\bigstar$, we must collapse exactly $(n/3)$ set gadgets. It is easily seen that the collapsed gadgets correspond to to an exact cover, as desired.

These arguments culminate in the following result.

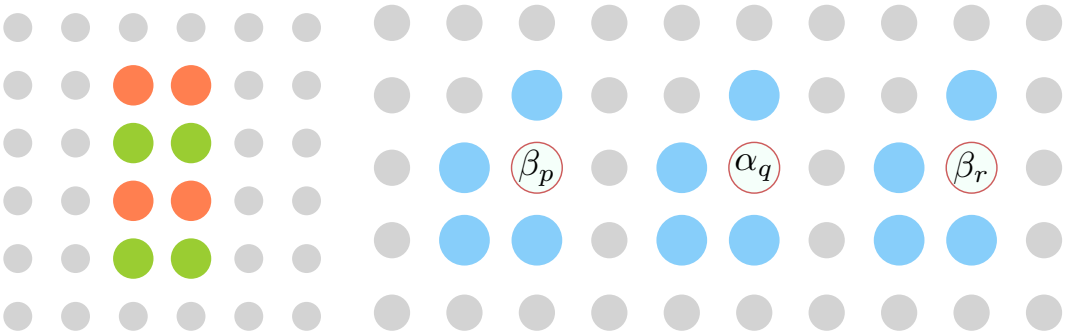
▶ **Theorem 4.1.** TwoDots *is NP-complete, even when played only with regular moves.*

4.2 Hardness with Constant Goals

We now turn to our quest of showing hardness when the game is not demanding when it comes to goals. This is our final reduction, and here we reduce from 3-SAT.

Overview of the Reduction. We present an informal sketch of the construction first. The idea is to have two dots of a special color, denoted by $\bigstar$, separated by a stack of dots corresponding to the clauses of the SAT instance. In particular, we will have a color corresponding to every variable and every clause, and the only way to destroy the column of dots separating the two dots colored $\bigstar$ is to use square moves on every color corresponding to a clause. However, these squares are not directly available on the original board, and they can be created by collapses, as in the previous reductions. As the reader as perhaps guessed by now, these collapses are in correspondence with variables being set appropriately with respect to the clauses.

One difference from the previous reductions is the additional concern that we cannot permit square moves corresponding to contradictory literals such as x and $\overline{x}$. Therefore, the squares corresponding to variables, which trigger the whole sequence are also not directly available — in the variable gadget, a potential square corresponding to a negated (positive) literal must be "sacrificed" in order for the square for the positive (negated) literal to manifest. We refer the reader to Figure 6a for the exact configuration.



(a) The variable gadget, where the two colors shown correspond to the two possible assignments.

(b) A Clause gadget corresponding to the clause $(\overline{x_p}, x_q, \overline{x_r})$. The color corresponding to the clause is depicted in Blue.

Figure 6 The variable and clause gadgets used in the reduction from 3-SAT.

The Construction Let ϕ be an instance of 3-SAT with clauses $C_1, \ldots, C_m$ over the variables $x_1, \ldots, x_n$. We have a board with $(m + 11)$ rows, and $(3n + 9m + 1)$ columns. We introduce the following colors.

- We have two colors α_i and β_i corresponding to each variable x_i and its negation, respectively.
- We introduce the color c_j corresponding to each clause C_j, and an additional color ★.
- We have as many dummy colors as there are locations on the board.

Now we turn to the configuration of the board, which is as follows (see Figure 7).

- The first $(m + 2)$ rows of the first column have dots with the colors $\bigstar, c_1, \ldots, c_m, \bigstar$, appearing in that order (from bottom to top).
- The first $3n$ columns starting from the second column, and the rows $(m+4), (m+5), (m+6), (m + 7)$ are reserved for the variable gadgets (see Figure 6a). In particular, the rows $(m + 4)$ and $(m + 6)$ on columns $3i$ and $(3i + 1)$ contain dots with the color α_i while the rows $(m + 5)$ and $(m + 7)$ on the same columns contain dots with the color β_i.
- The last $9m$ columns, and the rows $(m + 9), (m + 10)$ and $(m + 11)$ are reserved for the clause gadgets (see Figure 6b). In particular, we have the following.
  - If the clause C_j consists of the literals (ℓ_p, ℓ_q, ℓ_r), then we have dots with the colors corresponding to these literals occupying the locations at columns $(3n + 1) + (9j - 6)$, $(3n + 1) + (9j - 3)$ and $(3n + 1) + 9j$ on row $(m + 10)$, respectively.
  - On row $(m + 9)$ and $(m + 11)$, at columns $(3n + 1) + (9j - 6)$, $(3n + 1) + (9j - 3)$ and $(3n + 1) + 9j$, we have dots with the color c_j.
  - On row $(m + 9)$ and $(m + 10)$, at columns $(3n + 1) + (9j - 7)$, $(3n + 1) + (9j - 4)$ and $(3n + 1) + (9j - 1)$, we have dots with the color c_j.
- On any location (i, j) that is not accounted for by the above, we use a dot with the dummy color corresponding to that location.

Our instance allows for $(2n + m + 1)$ moves, the goal is given by $(\bigstar, 2)$, and σ is the null string. This completes our description of the instance, and we are now ready to show the correctness.

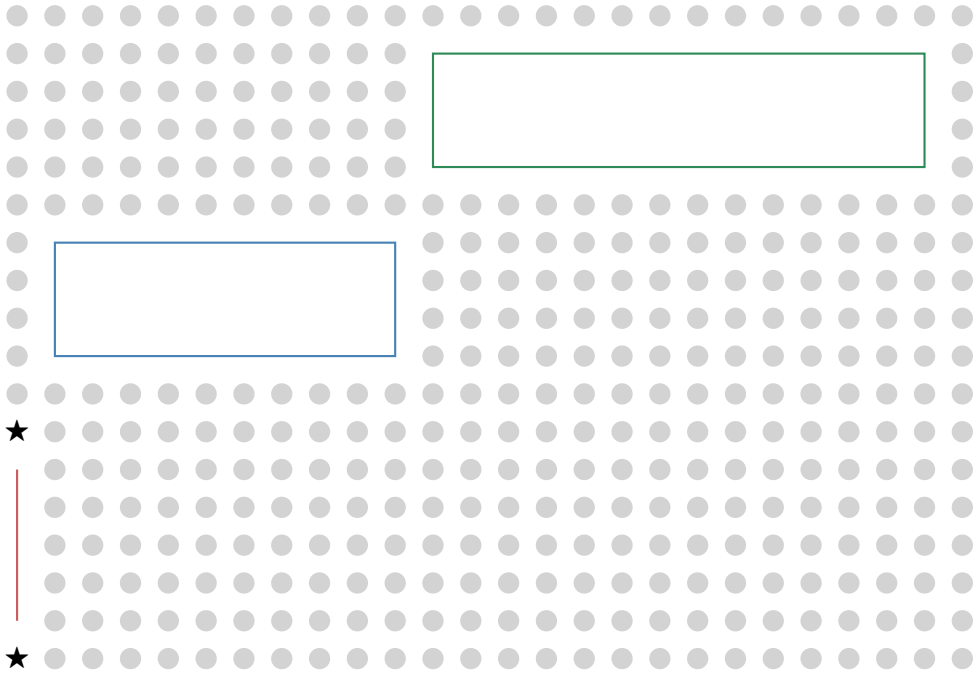


Figure 7 The overall schematic of our reduction from SAT. The bottom-left line in red is a cartoon for the list of clauses, the blue rectangle in the center hosts the variable gadgets from left to right, and the green rectangle in the top right contains the clause gadgets, again from left to right

As usual, to enable the discussion of the correctness, we introduce some terminology. We say that the gadget corresponding to the clause C_j collapses if a square move is executed on a color corresponding to some literal in C_j. Notice that if C_j collapses, then it creates at least one square of color c_j.

The Forward Direction In the forward direction, let $\tau : \{x_1, \ldots, x_n\} \to \{0, 1\}$ be a satisfying assignment for ϕ. For each variable x_i for which $\tau(x_i) = 1$, we execute a regular move on the pair of dots colored β_i such that a square on dots of color α_i is created, and then execute a square move on α_i. Similarly, for each variable x_i for which $\tau(x_i) = 0$, we execute a regular move on the pair of dots colored α_i such that a square on dots of color β_i is created, and as before, we execute the square move on β_i.

Note that we have used up $2n$ moves so far. Since τ is a satisfying assignment, every clause gadget collapses, and we use the next m moves to execute square moves on each color c_j, $1 \leq j \leq m$. At the end of this, the two dots colored $\star$ become adjacent and we use a regular move on them in the last available move to finish the game.

The Reverse Direction In the reverse direction, we first collect some immediate observations. Note that it is imperative for every dot in the first column and rows $2, 3, \ldots, m + 1$ to be hit, for the goal to be met. The only way for this to happen is for the player to execute square moves on each c_j, $1 \leq j \leq m$. This, in turn, is made possible only when every clause collapses. A clause collapse can only be caused by a square move on the α_i's and β_i's. Further, by the structure of the variable gadget, it is clear that for each i, a square move is executed on either α_i or β_i, but never both.

So we let X^+ be the set of variables for which square moves were executed on the corresponding

α_i, and analogously, we let X^- be the set of variables for which square moves were executed on the corresponding β_i. It follows from the last comment in the previous paragraph that these sets of variables are disjoint. Therefore, consider the well-defined assignment τ that sets everything in X^+ to one and everything in X^- to 0, and any remaining variables arbitrarily. We claim that τ is a satisfying assignment.

Indeed, if not, there is some clause C_j that is not satisfied by τ, and the corresponding clause in the game does not collapse by construction. This means, in turn, that there was no way for the dot colored c_j on the first column to be eliminated, which implies that the two dots colored ★ did not become adjacent, contradicting our assumption that we started with a winning sequence. This completes the argument, leading us to the following.

▶ **Theorem 4.2.** TwoDots *is NP-complete when there is one goal demanding two hits.*

▶ Remark. The reduction above could have also been executed with, say, Dominating Set. However, an interesting aspect of this reduction is that it is easily seen to work even if there is no upper bound specified on the number of moves. This was one of the reasons to parameterize by the number of moves, to see if this contained the hardness in the parameterized setting, although Corollary 3.3 answers this in the negative. In this context, we note that an $n^{\mathcal{O}(k)}$ algorithm in the number of moves is easily obtained, by guessing the move at each step, and simulating the game: this search tree has polynomially many branches and depth k.

5 Concluding Remarks

We have shown the NP-completeness of TwoDots in some fairly restricted settings. Somewhat unusually for combinatorial puzzles, these reductions turned out to be rather simple. The question of whether combining restrictions from all the scenarios finally brings us to a tractable setting is the most pertinent one. Other musings include the question of the complexity of the game when the number of columns is a constant, and what happens if we parameterize by the number of columns, or the number of colors. Also, in the version of the game where locations catch fire, possibly there are parallels with the Firefighting problem [4].

References

1 Aaron B. Adcock, Erik D. Demaine, Martin L. Demaine, Michael P. O'Brien, Felix Reidl, Fernando Sanchez Villaamil, and Blair D. Sullivan. Zig-zag numberlink is NP-complete. *JIP*, 23(3):239–245, 2015.

2 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

3 Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *MFCS: Symposium on Mathematical Foundations of Computer Science*, 2001.

4 Fedor V. Fomin, Pinar Heggernes, and Erik Jan van Leeuwen. Making life easier for firefighters. In Evangelos Kranakis, Danny Krizanc, and Flaminia L. Luccio, editors, *FUN*, volume 7288 of *Lecture Notes in Computer Science*, pages 177–188. Springer, 2012.

5 M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.

6 Stefan Langerman and Yushi Uno. Threes!, fives, 1024!, and 2048 are hard. *CoRR*, abs/1505.04274, 2015.

7 Toby Walsh. Candy crush is NP-hard. *CoRR*, abs/1403.1911, 2014.