On the complexity of two dots for narrow boards and few colors

On the Complexity of Two Dots for Narrow Boards and Few Colors

Davide Bilò
University of Sassari, Italy
davide.bilo@uniss.it
 https://orcid.org/0000-0003-3169-4300

Luciano Gualà
University of Rome "Tor Vergata", Italy
guala@mat.uniroma2.it
 https://orcid.org/0000-0001-6976-5579

Stefano Leucci
ETH Zürich, Switzerland
stefano.leucci@inf.ethz.ch
 https://orcid.org/0000-0002-8848-7006

Neeldhara Misra
Indian Institute of Technology, Gandhinagar
mail@neeldhara.com
 https://orcid.org/0000-0003-1727-5388

───── **Abstract** ─────

Two Dots® is a popular single-player puzzle video game for iOS and Android. A level of this game consists of a grid of colored dots. The player connects two or more adjacent dots, removing them from the grid and causing the remaining dots to fall, as if influenced by gravity. One special move, which is frequently a game-changer, consists of connecting a *cycle* of dots: this removes all the dots of the given color from the grid. The goal is to remove a certain number of dots of each color using a limited number of moves. The computational complexity of Two Dots has already been addressed in [Misra, FUN 2016], where it has been shown that the general version of the problem is NP-complete. Unfortunately, the known reductions produce Two Dots levels having both a large number of colors and many columns. This does not completely match the spirit of the game, where, on the one hand, only few colors are allowed, and on the other hand, the grid of the game has only a constant number of columns. In this paper, we partially fill this gap by assessing the computational complexity of Two Dots instances having a small number of colors or columns. More precisely, we show that Two Dots is hard even for instances involving only 3 colors or 2 columns. As a contrast, we also prove that the problem can be solved in polynomial-time on single-column instances with a constant number of goals.

1 Introduction

Two Dots® (http://weplaydots.com/twodots.html) is a popular single-player puzzle video game for iOS and Android. The game has been so much appreciated by the community that, not even after 3 years from its launch, a recently introduced follow-up game, called Dots&Co®

(`https://www.dots.co/dotsandco/`), has already passed the 5 millions of downloads. In its simplest form, the game is played on a vertical grid where each location initially contains 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). In particular, the player selects a path of dots of the same color which can be either simple or it can contain exactly one cycle. In the former case only the selected dots disappear, while, in the latter case, all dots of that color disappear. It turns out that the cyclic move is frequently a game-changer, and plays an important role in our results too. It is clearly a popular heuristic, and the official Two Dots tutorial even offers the helpful tip: *"When in doubt, make squares"*. After a move, all the remaining dots in the area fall down as if influenced by gravity. The game provides a certain number of moves, and demands certain goals to 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 it is removed).

The computational complexity of the game has been analyzed in [16], where the author showed that the problem of deciding whether an instance can be won by the player is NP-complete even in very restricted settings. In particular, the problem remains hard when the board has only four rows, or when there is only one goal of collecting two dots of a particular color, even if there is no restriction on the number of moves. In [16] it is also shown that the problem is W[1]-hard when parameterized by the number of moves. It turns out that all these reductions use a large (i.e., typically linear in the size of the instance) number of both different colors and columns. However, this does not completely match the spirit of the game, where, on the one hand, only few colors are allowed, and on the other hand, the arena of the game has only a constant number of columns, while there can be many rows (even if the player can only see the few down-most ones). Understanding the complexity of the game under these more realistic conditions is explicitly mentioned as open problems in [16].

In this paper, we partially fill this gap by showing that:

- the game is NP-complete even when the instance has three colors, two moves, and two goals;[1]
- the game is NP-complete even when the board has two columns and there is no restriction on the number of moves;
- the game is polynomial-time solvable when the board has only one column, provided that the number of goals is constant;
- the game is NP-complete even when the board has two rows.

Observe that the first two results immediately imply that the problem is not fixed parameter tractable when parameterized w.r.t. the number of colors, or the number of columns, unless P=NP. We leave open the problem of setting the computational complexity of the game when the instance has *both* a constant number of colors and columns.

Other related results

Two Dots belongs to the class of *tile-matching* video games. Tile-matching games allow the player to select a subset of tiles on the board according to some matching rule. Once selected, the tiles are removed from the board and the board configuration is updated automatically following the game-specific rules (for instance, all the remaining tiles might move to fill the voids as if influenced by gravity). Other popular games of this class also exhibit a rich

---

[1] A playable version of this reduction is available at `https://twodots.isnphard.com`.
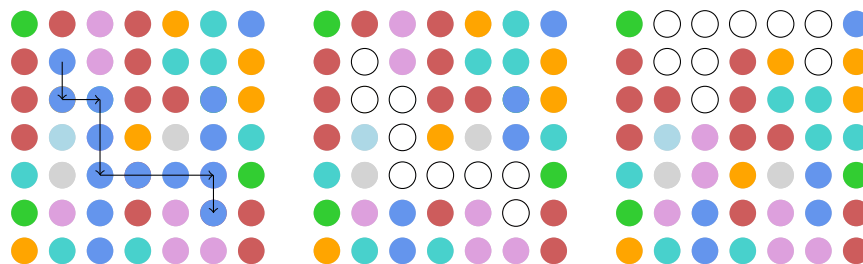
Figure 1 A depiction of a regular move. The first panel shows the set of locations of a move, the second panel shows the voids created, and the third panel shows how dots fall due to gravity.

combinatorial structure and have been studied from the computational complexity perspective. A noteworthy example is that of Candy Crush that has been shown to be NP-complete together with other *match-three* games in [12]. Another game having a somewhat similar mechanic to Two Dots is Flow Free: initially only a small number of dots of the board are colored, each color appearing exactly twice, while all the other positions are filled with *uncolored* dots. A move consists of connecting the two dots of a matching color by a path that traverses only uncolored-dots, which then inherit the color of the path's endpoints. The player is challenged to connect all the matching pairs while coloring all the dots on the board, which is equivalent to finding an embedding of monochromatic, non-intersecting, paths on the game board. This problem is also known as Zig-Zag Numberlink and has been shown to be NP-complete in [1]. The gameplay of Button and Scissors is also similar to the one of Two Dots: the player selects a monochromatic horizontal, vertical, or diagonal path traversing at least two dots (buttons), which are then removed (cut) from the board (the remaining dots are unaffected by gravity). It has been shown that clearing the board is NP-hard [11] even when only two colors are involved or when each color is used by at most 4 dots [6].

More broadly, all these games belong to the class of *casual* games. Casual games are often characterized by a puzzle-like gameplay and simple rules, which make these games easy to play, yet difficult to master. Indeed, the quality and enjoyability of puzzles has even been linked to their computational complexity [7]. It is then not surprising that many of most successful puzzles have been shown to be NP-complete, or even harder. This is the case, e.g., of Tetris [5], the $(n^2 - 1)$-puzzle (a generalized version of the famous 15-puzzle) [17], Rush Hour [9, 8], Peg-Solitaire [18, 13], Trainyard [3], Clickomania (also known as Same Game) [4, 2], 2048 [15], and many others [14].

Organization of the paper

The paper is organized as follows: Section 2 provides the problem definition and basic notation used through the paper, in Section 3 we show that the problem is NP-complete when the the number of colors is constant, and in Section 4 we address levels of the game with a constant number of columns. Finally, the results for levels with a constant number of rows are reported in Section 5.

2 Preliminaries

An instance of `Two Dots` consists of the following:
1. A $m \times n$ grid in which each position (i, j) is occupied by exactly one dot whose color belongs to a set $\mathcal{C}$.
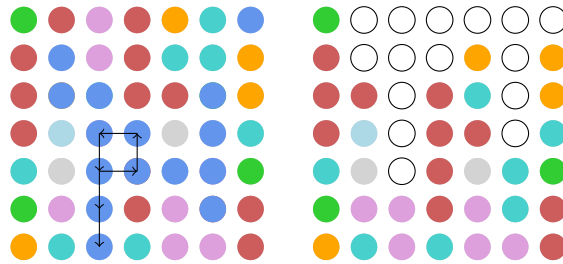
■ **Figure 2** A depiction of a cyclic move, which has the effect of eliminating all the blue dots from the board. The example is rather similar to the above, but note the difference in the number of voids created.

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 $\ell \in \mathbb{N}$.

Intuitively, a player has a winning strategy in an instance of `Two Dots` if all the goals can be achieved within k moves. To formalize this, we need to first define moves, and the notion of dots being collected.

There are two types of moves in `Two Dots`: *regular moves* and *cyclic moves*. We first describe the regular moves, which essentially involve removing simple paths in the grid occupied by the same color. Recall that, two dots are adjacent if they occupy two neighboring position on the grid, either horizontally or vertically (dots aligned diagonally are *not* considered adjacent). Any move (either regular or cyclic) consists of a sequence of locations $\langle t_1, \ldots t_s \rangle$, with $s \geq 2$, such that all locations contain a dot of the same color and, for every $i = 2, \ldots, s$, t_i is adjacent to t_{i-1}.

- In a regular move all the locations of the sequence $\langle t_1, \ldots t_s \rangle$ are unique (see Figure 1).
- In a cyclic move all the locations of the subsequence $\langle t_1, \ldots t_{s-1} \rangle$ are unique and t_s coincides with t_j, for some $j \in \{1, \ldots, s-4\}$ (see Figure 2). Informally speaking, the locations induce a cycle with a (possibly empty) dangling path from t_j consisting of the locations in $\{t_1, \ldots, t_j\}$.

A regular move creates voids in all the locations corresponding to the sequence. A cyclic move creates voids in all the locations of the grid containing dots whose color match the color of the dots in the selected sequence.[2] All the removed dots are *collected* by the player. Then 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, pushing the voids to the top.[3] We refer the reader to Figure 1 for an illustration.

At the end of k moves, when the game is over, we say that the player has won if, for each goal (c, ℓ), the number of dots of color c collected by the player is at least ℓ.

We wish to determine whether a `Two Dots` level (i.e., an instance) can be won by the player, namely whether there exists a sequence of at most k moves that meets all the goals.

---

[2] Our reductions work for this simplified model which somehow contains "all the hardness" of the game. In the actual game, dots that are enclosed in the cycle of a cyclic move become bombs which, after falling down, explode and destroy their 8-neighborhood. Our reductions still work in this general model but we would then need to introduce additional gaps in our gadgets.

[3] This is a standard approach to generalize the game, but it differs slightly from the model used in [16], where new dots join the board to fill the voids. We point out that the results in [16] also work in our case.

3   Hardness of levels with three colors, two moves, and two goals

In this section, we show that `Two Dots` is NP-complete even when the number of colors is bounded by three. Since it is clear that `Two Dots` is in NP (a certificate being the sequence of moves of a solution), we now focus on showing that `Two Dots` is NP-hard. A playable version of the reduction is available at `https://twodots.isnphard.com`.

We reduce from the EXACT COVER BY 3-SETS problem (`X3C` for short). In an `X3C` instance we are given: (i) a set $\mathcal{I} = \{I_1, I_2, \ldots, I_{3n}\}$ of $3n$ items; and (ii) a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of m subsets of $\mathcal{I}$, each subset having cardinality exactly 3. The problem is that of determining whether there exists a collection $\mathcal{S}' \subseteq \mathcal{S}$ of n sets such that $\bigcup_{S \in \mathcal{S}'} S = \mathcal{I}$, i.e., each item belongs to exactly one set in $\mathcal{S}'$. This problem is well-known to be NP-complete (see, for example, [10]). W.l.o.g. we will assume that n is an odd number.

The overview of our reduction is shown in Figure 3. We focus on describing the construct in the gray box, ignoring the first and last few columns. The middle columns correspond to items in groups of five — thus the first five columns encode the first item, and so forth. The dots are arranged in what we refer to as *wires*. There is one horizontal wire made of red dots and m horizontal wires made of blue dots. The red wire is a *check-wire* while the other m wires consisting of blue dots are *set-wires*. There are two empty rows between any pair of consecutive wires. To complete the board, we connect together the left (resp. right) endpoints of the set-wires using a single column of blue dots. Then, we introduce several more columns — the exact number of which we will specify later — on the extreme left and the extreme right of the board. These columns are populated with blue dots on rows corresponding to the set wires and are connected to the two blue columns joining the set-wires with a single dot each, on the top-right and on the bottom-left position, respectively (see Figure 3). All remaining dots in the grid are occupied by green dots.

The player is asked to eliminate a sufficiently large number of blue dots and all the red dots in two moves. The layout of the check-wire is such that all the red dots cannot all be eliminated in the first move. Further, we need to use at least one move to meet the goal for the blue dots. Therefore, in a winning strategy, the first move must involve the blue dots and achieve two things: (i) the move should clear the desired number of blue dots, and (ii) the move should result in the "alignment" of the red dots on the check-wire. The wires are set up in such a way that the dots on the check-wire align only when the dots removed from the set-wires correspond to the sets of a solution of the `X3C` instance.

We now describe the check-wire and set-wires in greater detail. First, let us establish some notation. Given a `X3C` instance with $3n$ items and m sets, the board in the reduced instance of `Two Dots` will have $5n + 2T$ columns and $4(m + 1)$ rows, where T is a parameter that we will fix later. In our discussion, we use $p_1, \ldots, p_T$ and $q_1, \ldots, q_T$ to label the first and last T columns, respectively. The remaining columns (indexed from $T + 1$ to $T + 5n$) are labeled by $c_1, \ldots, c_{5n}$ and use C to refer to this subset of columns. On the other hand, we index the rows simply by their numbers, with the topmost row being the first.

The set-wires.   For each set $S_j \in \mathcal{S}$ there is a two-cell tall set-wire traversing all columns in C. The j-th wire is on rows $r_j = 4(j + 1)$ and $r_j - 1$.

The wire is constructed as follows: For each item $I_i \in \mathcal{I}$ we consider the sub-grid consisting of 10 cells on rows r_j and $r_j - 1$, and on columns c_i to $c_i + 4$. If $I_i \notin S_j$ the lower row of this sub-grid is filled with blue dots while the top row remains empty (see the highlighted sub-grid corresponding to set S_2 and item I_4 in Figure 3). If $I_i \in S_j$ then the we place a blue dot on (i) all the cells of the bottom row of the sub-grid except for the one on column $c_i + 2$, and (ii) the cells on the top row of the sub-grid that are on columns $c_i + 1$, $c_i + 2$ and
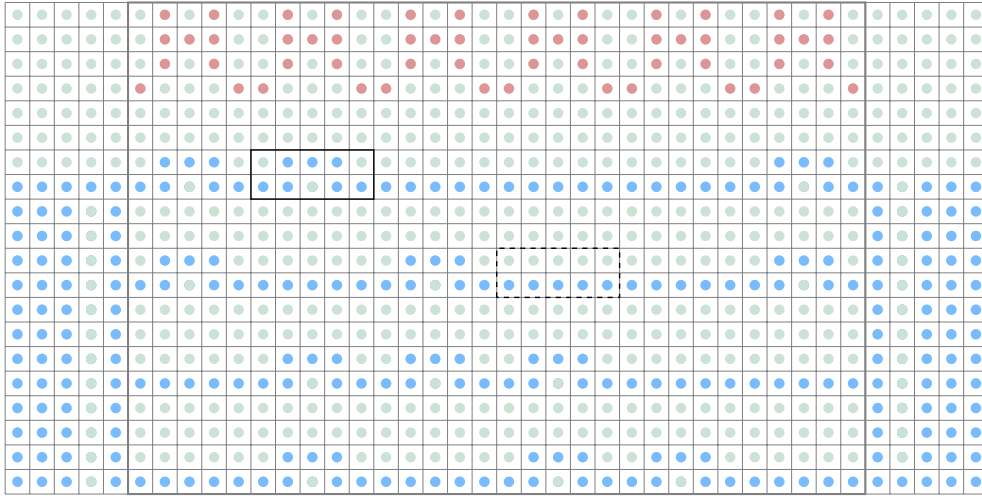
Figure 3 Overview of the reduction.

$c_i + 3$ (see the highlighted sub-grid corresponding to set S_1 and item I_2 in Figure 3).

Notice that if all the dots of the set-wire corresponding to, say, the set S_j are removed, then all the dots above row r_j will fall by exactly 1 cell, except for the ones on columns $c_i + 1$ and $c_i + 3$ where i is such that $I_i \in S_j$: in these columns, the dots above row r_j will fall by exactly 2 cells.

Finally, since the number of items contained in each set is exactly 3, notice that the number of blue dots in each set-wire is exactly $b := 5 \cdot (3n - 3) + 7 \cdot 3 = 15n + 6$.

The check-wire.   The check-wire is a four-cell tall wire that is initially placed at top of the board. It is constructed simply by repeating the same pattern of 9 red dots every 5 columns, i.e., column c_{i+5} has the same layout of column c_i. The pattern is shown in Figure 4 (a).

Suppose that all the blue dots contained the set-wires corresponding to solution $\mathcal{S}'$ of the X3C instance are removed by the player, from top to bottom. This would cause all the dots of the check-wire to fall by exactly n rows, except for the dots on column $c_i + 1$ and $c_i + 3$ for $i = 1, \ldots, 3n$ that will fall by exactly $(n + 1)$ cells. This will cause the dots on the items-wire to arrange in the configuration shown in Figure 4 (b). Notice that, in this configuration, all the $9 \cdot 3n = 27n$ dots of the items-wire can be removed by the player using a single move.

The reduction from X3C to Two Dots

We are now able to prove our result:

▶ **Theorem 1.** *Two Dots is* NP-complete *even when the numbers of colors, moves, and goals are bounded by 3, 2, and 2, respectively.*

Proof. Let $A = \langle \mathcal{I} = \{I_1, \ldots, I_{3n}\}, \mathcal{S} = \{S_1, \ldots, S_m\} \rangle$ be an instance of X3C and consider the corresponding instance B of Two Dots as described above, where T is such that the number η of blue dots in the first $T - 2$ columns (resp. the last $T - 2$ columns) is greater than the number of remaining blue dots.

Assume, without loss of generality, that $\mathcal{S}$ is not itself a set cover for $\mathcal{I}$ and remember that n is odd. We will show that at least 2η blue dots and all the red dots can be removed from B using at most two moves if and only if A admits an exact cover by 3-sets.
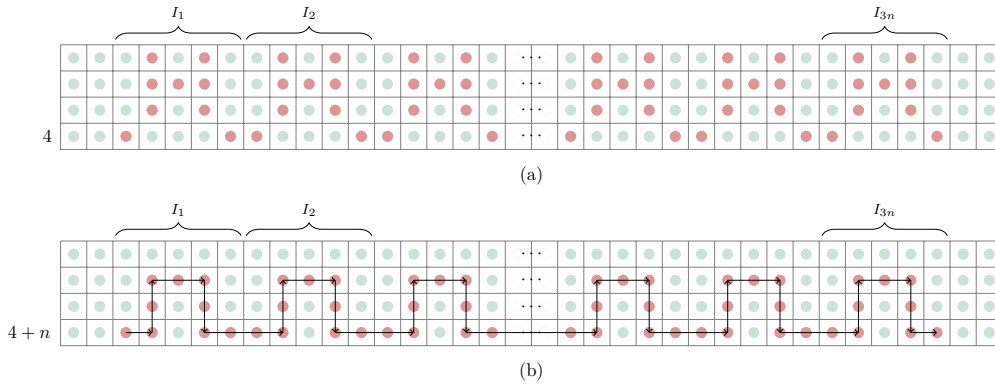
Figure 4 (a) Initial setup of the check-wire. (b) The check-wire once it gets aligned.

The Forward Direction. Let $\mathcal{S}' = \{S_{j_1}, S_{j_2}, \ldots, S_{j_n}\}$ exact cover by 3-sets for A where $j_k \in \{1, \ldots, m\}$ for $k = 1, \ldots, n$. We assume, w.l.o.g., that $j_1 < j_2 < \cdots < j_n$. A winning sequence of moves for the instance of B consists of:

1. connecting, in order and in a single move: all the blue dots in the fist $T - 2$ columns, the single dot in the $(T - 1)^{th}$ column, all the dots of the set-wires on rows $r_{j_1}, \ldots r_{j_n}$ in a zig-zag fashion, the single dot in the $(T + 5n + 1)^{th}$ column, and finally all the dots in the last $T - 2$ columns. Notice that the above move is feasible since n is odd.

2. connecting, in a single move, all the red dots. This is possible since $\mathcal{S}'$ is an exact cover, and hence the previous move will cause the check-wire to align.

It is easily checked that both goals are satisfied (notice that the first and last $T - 2$ columns contain 2η blue dots).

The Reverse Direction. Suppose now that there is a solution to the instance B of Two Dots. From the fact that: (i) we are permitted only two moves; (ii) we have to clear all the red dots; and (iii) all the red dots are not aligned in the initial state, it follows that the first move has to meet the goal for the blue dots and also align the red dots on the check-wire so that the second move can be used to eliminate all of them in one move. This means that the first move cannot be a cycle-move, that it must involve dots that belong to both the first and the last $T - 2$ columns of the board, and that it must traverse set-wires entirely and in a zig-zag fashion. Let $\mathcal{S}'$ be the sets corresponding to the sets-wires whose dots have been removed in the first move. Suppose, towards a contradiction, that $\mathcal{S}'$ is not an exact cover for A. This means that at least one of the following conditions is true: (i) there exists an item I_j that is not covered by $\mathcal{S}'$; or (ii) there exists one item I_j that belongs $t \geq 2$ sets in $\mathcal{S}'$. In the former case the number of blue dots removed from the columns associated with item I_j is the same, and hence it will not be possible to connect all the dots in the check-wire in a single move. In the latter case, the number of blue dots removed from the 2nd and 4th column associated with item I_j exceeds the corresponding number of removed blue dots for the 1st, 3rd, and 5th column by $2t$. Hence, after the first move, the red dots in the check-wire are not aligned and therefore it is not possible to meet the read goal using a single additional move.   ◀

4   Boards with a constant number of columns

In this section, we address the complexity of `Two Dots` on boards that have a constant number of columns. More precisely, we show that the problem is NP-complete if the board has two or more columns, while it is polynomial-time solvable in the one-column case. Interestingly, our hardness result holds even when the player has an unlimited number of moves and only one goal to achieve.

4.1   Hardness of levels with two columns, one goal and unlimited moves

We proceed here by a reduction from `3-SAT`. Let $C_1, \ldots, C_m$ be a set of clauses over the variables $x_1, \ldots, x_n$. We assume, without loss of generality, that every clause consists of exactly three literals. The overall structure of the `Two Dots` instance that we construct is given in Figure 5. We describe the components starting from the bottom. First, we stack up a collection of *clause gadgets*, one corresponding to each variable of the `3-SAT` instance. Then, after a suitable gap, we introduce the *variable gadgets*, one corresponding to each variable of the instance. Finally, we have a *formula-check* gadget, which is the basis for the only goal that we have in this instance. We introduce one color for every literal and one for every clause of the `3-SAT` instance. Let the colors associated with the literals x_i and $\overline{x_i}$ be p_i and q_i, respectively; while we denote the color associated with the clause C_j by ℓ_j. We also have one special color that we denote by $\int$. We now describe each gadget separately and then explain the equivalence of the instances. In the following, when we speak of gaps in the board, we may assume these to be dots of "dummy" colors, which are newly introduced colors distinct from the colors mentioned already, and also distinct from each other.

The clause gadgets.   Consider a clause C_j. The gadget corresponding to a clause is shown in part (a) of Figure 6. Let a, b, c be the colors corresponding to the literals of C_j. The first row is a gap row, and the next seven rows[4] consist of the following:

- Dots colored ℓ_j occupy the first column on all seven rows;
- Dots with colors a, b and c occupy the second column on the third, fifth and seventh rows, respectively; and
- Dots colored ℓ_j occupy the remaining rows on the second column.

After these rows, we introduce another gap row. Finally, the first column of the last three rows are occupied by dots colored a, b and c, respectively; while the second column on the last three rows are occupied by dummy dots. Note that because of the way the seven rows described above are "sandwiched" between gap rows, the only way to obtain a ℓ_j-colored square is to make a square move with at least one of a, b or c.

The variable gadgets.   For a variable x_i, the variable gadget consists of four rows, alternatingly occupied by dots of colors p_i and q_i on both columns (see Figure 6(c)). To be specific, the first and third rows have dots colored p_i on both columns, while the second and fourth rows have dots colored q_i on both columns. Observe that within the scope of this gadget, any valid gameplay can involve a square move on either p_i or q_i, but not both.

---

[4] Recall that our convention is to count rows from the top.
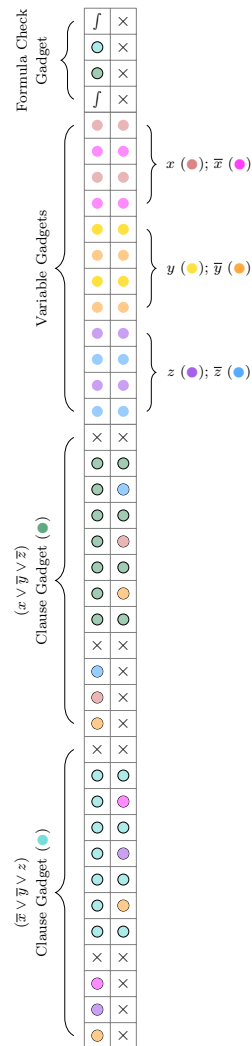
Figure 5 Overview of the reduction. The grid cells marked × are filled with distinct colors different from the ones used to represent variables and clauses. The goal of the game is to hit two dots of color ʃ and the number of moves are unbounded.

(a)   (b)   (c)   (d)   (e)
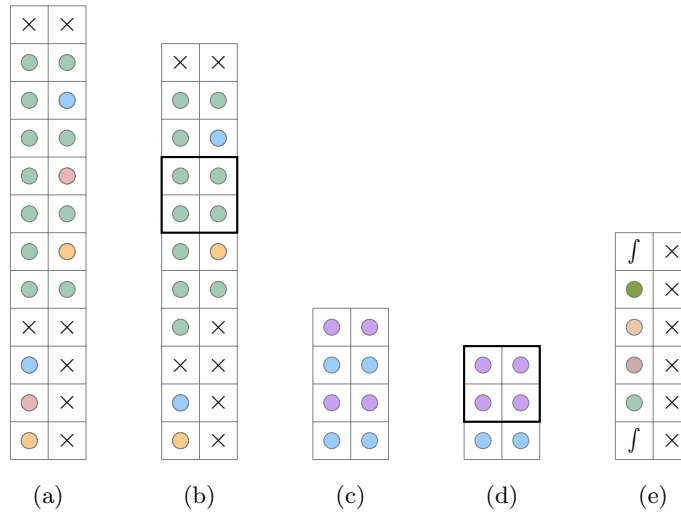
■ **Figure 6** (a) Initial setup of the clause gadget. (b) The clause gadget in its aligned state. (c) Initial setup of the variable gadget. (d) The state of the variable gadget after one move on one of the literals. (e) The formula-check gadget.

The formula-check gadget. The formula-check gadget is depicted in Figure 6(e). It consists of $(m + 2)$ rows, where the dots occupying the first and last row of the first column have color $\int$ and the interim rows comprise of one dot each of color ℓ_j, $1 \leq j \leq m$. For all rows, we have dummy dots occupying the second column. The only goal in the game will be to hit two dots colored $\int$.

We are now ready to prove our main theorem for this section:

▶ **Theorem 2.** *Two Dots is* NP-complete *on boards that have only two columns, even when the player has to achieve only one goal with an unlimited number of moves at his disposal.*

Proof. We proceed by a reduction from 3-SAT. Let an instance $\mathcal{I}$ of 3-SAT comprise of the clauses $C_1, \ldots, C_m$ over the variables $x_1, \ldots, x_n$, where every clause consists of exactly three literals. Let B denote the instance of Two Dots constructed as described above. Recall that the goal is to hit two dots colored $\int$ and there is no bound on the number of moves. We now establish the equivalence of the instances.

The Forward Direction. Let $\tau : V \to \{0, 1\}$ be a satisfying assignment for the instance $\mathcal{I}$. For any variable x for which $\tau(x) = 1$, we eliminate the row containing dots colored q_i and perform a square move on the dots of color p_i, which becomes feasible once the q_i-colored dots are removed from either row of the variable gadget. On the other hand, for any variable x for which $\tau(x) = 0$, we eliminate the row containing dots colored p_i and perform a square move on the dots of color q_i. Observe that after each square move on a variable gadget, the design of the clause gadgets ensures that the number of dots hit in both columns is equal. Therefore, after these moves are complete, an ℓ_j-colored square is created for every $1 \leq j \leq m$. Making these moves in any order leaves us with a board where the $\int$-colored dots become adjacent, and the goal can be met with one final move.

The Reverse Direction. A winning gameplay involves a move that hits at least two $\int$-colored dots on board. Recall that the only $\int$-colored dots are available in the formula check

gadget and that they are separated by m dots corresponding to the colors of the clauses. Since all other adjacent locations are occupied by dummy dots, it follows that the only way to arrive at a configuration where the two $\int$-colored dots are adjacent is to play a ℓ_j-cyclic move for all $1 \leq j \leq m$. For any such j, consider the clause gadget corresponding to C_j and let a, b, c denote the colors of the literals that appear in C_j. A cycle with dots colored ℓ_j can only manifest if there was a square move involving one of the colors a, b or c. We now propose an assignment based on these moves: set the variable x_i to 1 if the given gameplay involved a square move on the color p_i and set x_i to 0 if the gameplay involved a square move on the color q_i . If the gameplay did not have a square move on either p_i or q_i, then set the value of x_i arbitrarily. Note that this is a well-defined assignment since no valid gameplay can involve a square move on both p_i and q_i, by the design of the variable gadget and the fact that the variable gadget is the only part of the overall construction where dots of these colors are adjacent. To see that this is a satisfying assignment, proceed by contradiction: if a clause C_j is not satisfied, then it is easy to see that the choice of square moves amongst the variable gadget which led us to our assignment were such that no ℓ_j-squares were generated, which contradicts our assumption that we started with a winning gameplay.   ◄

4.2   A polynomial-time algorithm for levels with one column

Let B be an instance of `Two Dots` where the board consists of one column with n dots. For $1 \leq i \leq n$, let $c(i)$ denote the color of the ith dot in B. Moreover, for $i \leq j$, let $B_{i,j}$ denote the subsequence of dots starting at the ith row and ending at the jth row. By a slight abuse of notation, we also use $B_{i,j}$ to denote the natural instance of `Two Dots` associated with this (truncated) board. As a warm-up, and since this is instrumental to our general algorithm, we begin by considering the case in which the goal is to remove *all* the dots in the board (i.e., to *clear* the board) using the minimum number of moves.

4.2.1   Clearing the board

Let $C(i, j)$ be the minimum number of moves needed to clear $B_{i,j}$, or $+\infty$ if there exists no such sequence of moves. We now describe a dynamic programming algorithm to compute all the values $C(i, j)$ (and, in particular, $C(1, n)$).

   If $j - i + 1 \leq 0$, or if $j - i + 1 = 1$, then clearly $C(i, j) = 0$ and $C(i, j) = +\infty$, respectively. We therefore consider the case in which $j \geq i + 1$. Notice that, in order to clear the board $B_{i,j}$, the dot on the first row of $B_{i,j}$ must be hit with a move connecting (at least) one other dot. Let h be the smallest index of a row in $B_{i,j}$ that contains one such dot. We distinguish two cases:
1. The move hitting the dot on the first row of $B_{i,j}$ connects exactly 2 dots.
2. The move hitting the dot on the first row of $B_{i,j}$ connects 3 or more dots.

   If the former case we "guess" the location h of the dot that partners with the first dot of $B_{i,j}$. This decomposes our instance into two sub-instances corresponding to the boards $B_{i+1,h-1}$ and $B_{h+1,j}$. In formulas:

$$C_1(i, j) = \min_{i < h \leq j \text{ such that } c(h) = c(i)} \{C(i + 1, h - 1) + C(h + 1, j) + 1\},$$

where last term accounts for to the move that is used to hit the ith dot and the hth dot.

   In the latter case we still guess the location h, but we now decompose the board into two different sub-instances, namely $B_{i+1,h-1}$ and $B_{h,j}$ (i.e., we still include the hth dot in the second sub-instance). The hth dot can then be combined with another dot in row $h' > h$

belonging to the same move that is used to hit both the ith and the hth dot. This can be repeated recursively until the last two dots hit by the move are considered, which falls into the former case, and thus accounts for the whole multi-dot move. In formulas:

$$C_2(i,j) = \min_{i<h\leq j \text{ such that } c(h)=c(i)} \{C(i+1,h-1)+C(h,j)\}.$$

Overall, our recurrence is given by $C(i,j) = \min\{C_1(i,j), C_2(i,j)\}$.

4.2.2 The general case

For the sake of simplicity we describe our algorithmic approach for the case when the game has only one goal, and our task is to determine the minimum number of moves that are needed to achieve said goal. In particular, suppose that the single goal demands the elimination of ℓ red dots. Nevertheless, our approach can be easily generalized for the case of multiple goals.

We now describe a dynamic programming routine to check if there is a sequence of at most k moves that hits at least ℓ red dots. To this aim, let $T(i,j,\delta)$ be the minimum number of moves needed to gain at least δ red dots in $B_{i,j}$, or $+\infty$ if there is no such sequence of moves. By our definition, we have that, if $\delta \leq 0$, then $T(i,j,\delta) = 0 \ \forall i,j$. If $\delta > 0$ and $j <= i+1$, then $T(i,j,\delta) = +\infty$. Otherwise, we consider the following three cases:

1. There is a solution that does not hit the ith dot.
2. There is a solution that hits the ith dot along with one other dot.
3. There is a solution that hits the ith dot along with two or more other dots.

To describe our recurrence corresponding to these cases, we denote by $\nabla(i',j')$ the number of red dots in $B_{i',j'}$. We start by addressing the first case, in which the ith dot can be simply ignored from the current board:

$$T_1(i,j,\delta) = T(i+1,j,\delta).$$

In the second case, we "guess" the location h of the dot that partners with the ith dot in an optimal move. To this end, we consider separately the subsequences of dots corresponding to $B_{i+1,h-1}$ and to $B_{h+1,n}$. This two sub-instances are handled differently as, in order to connect the ith dot with the hth dot, all the dots in $B_{i+1,h-1}$ need to be removed (while this is not true for $B_{h+1,n}$. We can write:

$$T_2(i,j,\delta) = \min_{i<h\leq j \text{ such that } c(h)=c(i)} \{C(i+1,h-1)+T(h+1,j,\delta-\nabla(i,h))+1\}.$$

In the last case, we have the following analogous recurrence:

$$T_3(i,j,\delta) = \min_{i<h\leq j \text{ such that } c(h)=c(i)} \{C(i+1,h-1)+T(h,j,\delta-\nabla(i,h-1))\}.$$

Overall, our recurrence is given by $T(i,j,\delta) = \min\{T_1(i,j,\delta), T_2(i,j,\delta), T_3(i,j,\delta)\}$, and we can state the following.

▶ **Theorem 3.** *Two* `Dots` *admits a polynomial time algorithm for a constant number of goals when the board consists of only one column.*

Figure 7 The edge gadget for the case of boards with two rows. Here we are representing an edge e (●) incident on vertices u (●) and v (●).

5   Boards with a constant number of rows

Here, we state and prove the following result, strengthening the NP-hardness result for four rows given in [16]. Unlike for the case of two columns, the reduction in this setting employs the use of many goals and many colors.

▶ **Theorem 4.** `Two Dots` *is* NP-complete *when the board has only two rows and can be solved in polynomial time for boards that have one row.*

Proof. The hardness for the case of two rows is by a reduction from `Vertex Cover`. Let the $(G = (V, E), k)$ be an instance of `Vertex Cover` where $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. Our board consists of two rows and $2n + 5m$ columns. For every vertex v_i, introduce four dots of color c_i such that they form a square on the adjacent columns $2i - 1$ and $2i$. For every edge $e_j = (v_p, v_q)$, introduce the edge gadget shown in Figure 7, which involves three dots that have color d_j and one dot each of color c_p and c_q. The remaining five positions in the grid are filled with dummy dots that have colors distinct from the colors that correspond to edges and vertices. The goal is to hit at least two dots of color d_j for each $1 \le j \le m$ in at most $m + k$ moves.

We now argue the equivalence of these instances. In the forward direction, given a vertex cover $S \subseteq V$ of size k, perform the square moves on the colors corresponding to vertices in S. This uses up the first k moves. Since S is a vertex cover, this causes at least two dots of color d_j to become adjacent in the edge gadget corresponding to e_j. The remaining m moves can be used to now meet the demands of the game. In the other direction, assume we have a valid gameplay that meets all the goals. Note that at least m moves must be used to hit dots of color d_j. Let S denote the set of colors on which square moves were employed in the remaining moves. Note that this is a set of at most k colors, each of which corresponds to a vertex in the graph G. We claim that this subset is a vertex cover. Indeed, if not, observe that the edge gadget corresponding to any uncovered edge (say e_j) remains unchanged and therefore the goal for the color d_j cannot be met, contradicting our assumption that we started with a winning gameplay.

We now turn to the case of boards with one row, where we claim that a natural greedy algorithm solves `Two Dots` in polynomial time. First, note that the goal of hitting k dots of color c can be met if and only if the total number of dots colored c present in intervals of length at least two is at least k. Further, it is easily checked that in an optimal play, every move hits colors for which there is a non-trivial goal left (note that this is not true in the general game, where moves involving colors that have no goals associated with them can also help with meeting the goals of the game). Finally, observe that we may employ a greedy strategy here to meet any particular goal, where we proceed by hitting maximal intervals of the longest length of a particular color first.   ◀

6   Conclusions

In this paper we have settled the computational complexity of several restrictions of Two Dots involving narrow boards and/or few colors. Some problems which are still open and that we regard as interesting are those of understanding the computational complexity of Two Dots for (i) boards with only two colors, and (ii) boards with a constant number of columns and colors, which nicely captures the spirit of the game. Finally, we remark that – by carefully positioning the wire gadgets, and employing some other small modifications – our reduction involving 3 colors, 2 moves, and 2 goals, can be adapted to require only 1 goal.

References

1   Aaron B. Adcock, Erik D. Demaine, Martin L. Demaine, Michael P. O'Brien, Felix Reidl, Fernando Sánchez Villaamil, and Blair D. Sullivan. Zig-zag numberlink is np-complete. *JIP*, 23(3):239–245, 2015. `doi:10.2197/ipsjjip.23.239`.

2   Aviv Adler, Erik D Demaine, Adam Hesterberg, Quanquan Liu, and Mikhail Rudoy. Clickomania is hard, even with two colors and columns. *The Mathematics of Various Entertaining Subjects: Research in Games, Graphs, Counting, and Complexity*, 2:325, 2017.

3   Matteo Almanza, Stefano Leucci, and Alessandro Panconesi. Trainyard is np-hard. *Theoretical Computer Science*, 2017. `doi:10.1016/j.tcs.2017.09.039`.

4   Therese C. Biedl, Erik D. Demaine, Martin L. Demaine, Rudolf Fleischer, Lars Jacobsen, and J. Ian Munro. The complexity of clickomania. *CoRR*, cs.CC/0107031, 2001. URL: `http://arxiv.org/abs/cs.CC/0107031`.

5   Ron Breukelaar, Erik D. Demaine, Susan Hohenberger, Hendrik Jan Hoogeboom, Walter A. Kosters, and David Liben-Nowell. Tetris is hard, even to approximate. *Int. J. Comput. Geometry Appl.*, 14(1-2):41–68, 2004. `doi:10.1142/S0218195904001354`.

6   Kyle Burke, Erik D. Demaine, Harrison Gregg, Robert A. Hearn, Adam Hesterberg, Michael Hoffmann, Hiro Ito, Irina Kostitsyna, Jody Leonard, Maarten Löffler, Aaron Santiago, Christiane Schmidt, Ryuhei Uehara, Yushi Uno, and Aaron Williams. Single-player and two-player buttons & scissors games - (extended abstract). In Jin Akiyama, Hiro Ito, Toshinori Sakai, and Yushi Uno, editors, *Discrete and Computational Geometry and Graphs - 18th Japan Conference, JCDCGG 2015, Kyoto, Japan, September 14-16, 2015, Revised Selected Papers*, volume 9943 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2015. `doi:10.1007/978-3-319-48532-4_6`.

7   Davide Eppstein. Computational complexity of games and puzzles. https://www.ics.uci.edu/ eppstein/cgt/hard.html, accessed on the 23rd of February 2018.

8   Henning Fernau, Torben Hagerup, Naomi Nishimura, Prabhakar Ragde, and Klaus Reinhardt. On the parameterized complexity of the generalized rush hour puzzle. In *Proceedings of the 15th Canadian Conference on Computational Geometry, CCCG'03, Halifax, Canada, August 11-13, 2003*, pages 6–9, 2003. URL: `http://www.cccg.ca/proceedings/2003/22.pdf`.

9   Gary William Flake and Eric B. Baum. Rush hour is pspace-complete, or "why you should generously tip parking lot attendants". *Theor. Comput. Sci.*, 270(1-2):895–911, 2002. `doi:10.1016/S0304-3975(01)00173-6`.

10   M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

11   Harrison Gregg, Jody Leonard, Aaron Santiago, and Aaron Williams. Buttons & scissors is np-complete. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, Kingston, Ontario, Canada, August 10-12, 2015*. Queen's University, Ontario,

Canada, 2015. URL: `http://research.cs.queensu.ca/cccg2015/CCCG15-papers/48.pdf`.

12 Luciano Gualà, Stefano Leucci, and Emanuele Natale. Bejeweled, candy crush and other match-three games are (np-)hard. In *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26-29, 2014*, pages 1–8. IEEE, 2014. `doi:10.1109/CIG.2014.6932866`.

13 Luciano Gualà, Stefano Leucci, Emanuele Natale, and Roberto Tauraso. Large peg-army maneuvers. In Erik D. Demaine and Fabrizio Grandoni, editors, *8th International Conference on Fun with Algorithms, FUN 2016, June 8-10, 2016, La Maddalena, Italy*, volume 49 of *LIPIcs*, pages 18:1–18:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. `doi:10.4230/LIPIcs.FUN.2016.18`.

14 Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation*. A K Peters, 2009.

15 Stefan Langerman and Yushi Uno. Threes!, fives, 1024!, and 2048 are hard. In Erik D. Demaine and Fabrizio Grandoni, editors, *8th International Conference on Fun with Algorithms, FUN 2016, June 8-10, 2016, La Maddalena, Italy*, volume 49 of *LIPIcs*, pages 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. `doi:10.4230/LIPIcs.FUN.2016.22`.

16 Neeldhara Misra. Two dots is np-complete. In Erik D. Demaine and Fabrizio Grandoni, editors, *8th International Conference on Fun with Algorithms, FUN 2016, June 8-10, 2016, La Maddalena, Italy*, volume 49 of *LIPIcs*, pages 24:1–24:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. `doi:10.4230/LIPIcs.FUN.2016.24`.

17 Daniel Ratner and Manfred K. Warmuth. Nxn puzzle and related relocation problem. *J. Symb. Comput.*, 10(2):111–138, 1990. `doi:10.1016/S0747-7171(08)80001-6`.

18 Ryuhei Uehara and Shigeki Iwata. Generalized Hi-Q is NP-complete. *IEICE Transactions (1976-1990)*, 73(2):270–273, 1990.