On the PSPACE-completeness of Peg Duotaire and other Peg-Jumping Games

Davide Bilò
Dipartimento di Scienze Umanistiche e Sociali, University of Sassari, Italy.
davide.bilo@uniss.it
https://orcid.org/0000-0003-3169-4300

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

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

Guido Proietti
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, University of L'Aquila, Italy, and Istituto di Analisi dei Sistemi ed Informatica, CNR, Roma, Italy.
guido.proietti@univaq.it
https://orcid.org/0000-0003-1009-5552

Mirko Rossi
Dipartimento di Ingegneria dell'Impresa, University of Rome "Tor Vergata", Italy.
r.mirko25@gmail.com

Abstract

Peg Duotaire is a two-player version of the classical puzzle called Peg Solitaire. Players take turns making peg-jumping moves, and the first player which is left without available moves loses the game. Peg Duotaire has been studied from a combinatorial point of view and two versions of the game have been considered, namely the single- and the multi-hop variant. On the other hand, understanding the computational complexity of the game is explicitly mentioned as an open problem in the literature. We close this problem and prove that both versions of the game are PSPACE-complete. We also prove the PSPACE-completeness of other peg-jumping games where two players control pegs of different colors.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases peg duotaire, pspace-completeness, peg solitaire, two-player games

Digital Object Identifier 10.4230/LIPIcs.FUN.2018.8

1 Introduction

Peg-Jumping games are games with one or more players that are played on boards of different shapes. Each position of the board can host at most one peg, and a move consists of *jumping* a peg over an (horizontally or vertically) adjacent peg into an empty position. The move causes the peg that is jumped over to be removed from the board (see Figure 1). Arguably, the most popular game in this class is the single-player puzzle called *Peg Solitaire* (also

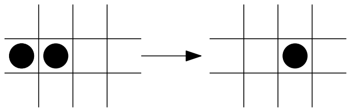


Figure 1 A move in peg-jumping games.

known as *Hi-Q*), in which the aim is to find a sequence of moves which reduces an initial placement of pegs into a single peg (and thus, the goal is that of clearing the board). A classical instance of the game has a cross-shaped board full of pegs except for the central position. The Peg Solitaire is an ancient game and its history dates back to at least the 17th century (see [2] for a comprehensive overview on the game).

Several other single-player peg-jumping games have been considered. For example, in the *Solitaire-Reachability* [15], the goal is, given an initial configuration of pegs, to find a sequence of moves that places any peg on a given target position (it is not required to remove all the other pegs). Another prominent game in this class, with a slightly different flavor, is the *Solitaire-Army* problem: given a *desert* region on a (usually infinite) board, and a target position in this region, one wishes to find an initial configuration of pegs outside of the desert that allows a peg to reach the target position through a valid sequence of moves. In its classical formulation, introduced by J.H. Conway in 1961, the desert is a half-plane, and the challenge is to understand what is the farthest distance in the desert that allows the target position to be reached. Conway devised an elegant potential argument to show that distance 5 cannot be reached on any finite board [18]. Other desert shapes have also been considered, such as square- and rhombus-shaped deserts [8].

In this paper, we focus on 2-player peg-jumping games, mainly on *Peg Duotaire*, a game introduced in [22], in which two players alternatively make a peg move and the winner is the last player to move. Two versions of Peg Duotaire have been considered: the *single-hop* Duotaire [22, 13], where each move consists of a single-hop jump, and the *multi-hop* Duotaire [21], where a series of (single-hop) jumps with the same peg can be made on a given turn. Both variants are impartial games, and they have been studied from a combinatorial point of view, while the problem of understanding the computational complexity of Peg Duotaire is mentioned as an open problem in the book by Hearn and Demaine (Section A.4 in [17]) and [21].

Our results

We study the problem of deciding whether the first player in a Peg Duotaire instance can force a win. As our main result, we show that this problem is PSPACE-complete for both versions of the game, namely the single- and the multi-hop variant. This closes the open problem given in [17] and [21].

We also consider another peg-jumping game, namely a 2-player version of Solitaire-Reachability. In this game, pegs are partitioned into white pegs (controlled by the first player) and black pegs (controlled by the second player). A move of the first (resp., the second) player consists of a jump involving only white (resp., black) pegs. However, pegs of a given color can prevent jumps of pegs of the other color since they occupy positions of the board. Moreover, each player has a target position that wants to reach with a peg (and the two target positions might coincide). As a natural extension of Solitaire-Reachability, we assume that the winner is the first player that reaches its target position.[1] However, different types of winning conditions can be considered here. For example, we can assume –as usual in the combinatorial game community– that the winner is the player that makes that last move, or a combination of the two mentioned rules, e.g., a player wins by either reaching his target position or by leaving his opponent with no available moves. We prove that all these variants are PSPACE-complete.

Related Results

Despite of the simplicity of their rules [10, 11, 7, 5], peg-jumping games exhibit a non-trivial combinatorial richness, and for this reason attracted the attention of many researchers over their long history [6, 3, 4]. From a computational point of view, it has been shown that single-player Peg Solitaire is NP-complete when the goal is to clear the entire board [23], or when the task is to decide whether a given target position can be reached [15]. On the other hand, deciding whether a given configuration can be transformed into a single peg is polynomial-time solvable for rectangular boards of fixed (constant) height, since solvable instances form a regular language [21, 22].

As far as 2-player peg-jumping games are concerned, single-hop Duotaire was introduced in [22], and then studied in [13, 21], while the multi-hop variant has been introduced in [21], where, besides other results, it is shown that even in the one-dimensional case, the set of instances for which the first player wins cannot be described by a context-free language.

Another work which is close in spirit to our is [16], where the PSPACE-completeness has been proved for another 2-player peg-jumping game called *Konane*, an ancient Hawaiian game in which pegs are of two different colors, and a player moves a peg of his color by jumping it over a peg of the opposite color in order to capture it. A peg may make multiple successive jumps in a single move, as long as they are in a straight line (while no turns are allowed within a single move). The first player that is unable to move wins. Due to the differences of the rules, the reduction in [16] cannot be easily adapted to prove PSPACE-completeness of the games we consider here. Finally, the present work contributes to the rich literature investigating the computational complexity of combinatorial games [17, 14, 19, 1, 20, 9].

2 Single-Hop Duotaire

2.1 Overview

In this section we focus on Single-Hop Duotaire and we prove that the problem of deciding whether the first player can force the win is PSPACE-complete.

Our reduction is from *directed vertex geography (DVG)*. In this problem, we want to decide whether the first player can force a win in the following game. We are given a directed graph G and a distinguished vertex $s \in V(G)$ that initially contains a *token*. Two players take turns performing the following move: first the token is moved from its current vertex u to a neighboring vertex v, traversing the edge $(u, v) \in E(G)$, and then vertex u is deleted from the graph. The first player who has no legal move loses the game (and the other player wins).

The DVG problem is known to be PSPACE-complete even when G is planar, bipartite, all the vertices have maximum degree 3, maximum indegree 2, and maximum outdegree 2

[1] In this case, we can assume that a player with no available moves can skip his turn, and that the game can end with a draw, whenever no player can move and no target position has been reached yet.

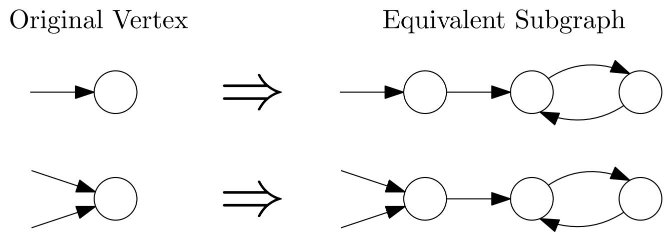


Figure 2 Vertex transformation. Original verteces with outdegree 0 (on the left), and equivalent subgraphs (on the right).

[12]. We will furthermore assume that vertex s has outdegree 2 and no incoming edges, while all the vertices in $V(G) \setminus \{s\}$ have either (i) indegree 1 and outdegree 2, (ii) indegree 2 and outdegree 1, or (iii) indegree 1 and outdegree 1. It turns out that these assumptions can be easily guaranteed by performing suitable transformations of the input graph, as we will discuss in the sequel.

The idea is to consider a planar embedding of G on a grid from which we build an equivalent instance of single-hop Duotaire where the token is simulated by a specific *token peg* (see Figure 7 for an example of the input graph G and of the associated single-hop Duotaire instance). We will simulate the behavior of the vertices of G using suitable *gadgets*: these gadgets take the token peg as an input, which encodes the act of placing the token on the associated DVG vertex (as a result of a previous move), and route it to a specific output, which corresponds to selecting the next position of the token in DVG (and hence to selecting an outgoing edge). The token peg will be transported from a vertex output (representing one endpoint of an edge) to a vertex input (representing the other endpoint) using a *wire* gadget.

Moreover, in an isolated area of the board, we place two adjacent pegs that allow the players to play a one-time extra move that we call *dummy* move. Except for this dummy move, all the other moves available to the players at any given point in time will involve the token peg.

2.2 Transforming the input DVG instance

Here we show how an instance of DVG on a planar, bipartite graph G in which all the vertices have maximum degree 3, maximum indegree 2, and maximum outdegree 2, can be transformed in order to further ensure that:

- s has outdegree 2 and no incoming edges.
- all the vertices in $V(G) \setminus \{s\}$ have either (i) indegree 1 and outdegree 2, (ii) indegree 2 and outdegre 1, or (iii) indegree 1 and outdegree 1.

First of all, we delete from G all the edges entering in s, and we iteratively remove all the other vertices of indegree 0. Then, while s has exactly one outgoing edge (s, s'), we perform the following operation: we delete s from G, we move the token to s', and we rename s' to s. It is easy to see that each such operation yields an equivalent instance in which the roles of the two players are exchanged: Player 1 can force a win in the instance preceding the operation iff Player 2 can win in the resulting instance (recall that Player 1 is always the first player to move, and that PSPACE is closed under complement).

After the transformation, G contains exactly 1 vertex with indegree 0 (i.e., s), which must also have outdegree 2. All the remaining vertices of G are either of one of the forms in (i), (ii), (iii), or they have outdegree 0 and indegree in $\{1, 2\}$. In the latter case, they can be replaced with the equivalent subgraphs shown in Figure 2.

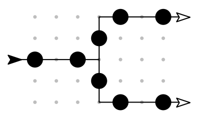


Figure 3 The gadget for a vertex with indegree 1 and outdegree 2.

2.3 Gadgets

Here we describe all the gadgets. Each gadget is meant to be played in one or more prescribed ways and is designed to ensure that any player deviating from the intended play will necessarily lose the game.

2.3.1 Vertices with outdegree two

In our reduction there are two kinds of vertices having outdegree 2, namely the starting vertex s (having indegree 0), and vertices having indegree 1.

Let us consider the case of vertices with indegree 1 first, which are implemented as shown in Figure 3. Players will be able to play the gadget whenever the token peg reaches the position marked with the black arrow. W.l.o.g., we assume that, once the token peg is in place, it is Player 1's turn. The intended play of the gadget follows the solid lines via alternating moves of the players. In particular, notice that Player 2 moves the token peg into the center of the gadget, where the solid lines meet. At this point, Player 1 can choose to jump over either the peg immediately above, or the peg immediately below. This corresponds to choosing which of the two outgoing edges of the associated vertex in the DVG instance the token traverses next. The following moves are straightforward and will bring the token peg to one of the two positions marked with the white arrows. Notice that this last move is performed by Player 1, and hence the turn will be up to Player 2.

We now argue that any deviation from the above strategy, will cause the deviating player to lose. In particular, all the deviations in this gadget consist of using one of the pegs of the gadget to jump over the token peg. However, if a player plays such a move he will bring the board in a configuration where the only available move is the dummy move. The opponent can then use this dummy move to win the game.

On the converse, if any player plays the dummy move instead of a move involving the token peg, the opponent can respond making a move that jumps over the token peg, thus reaching a configuration where no move is left available to the other player (thus winning the game).

As far as the starting vertex s is concerned, it suffices to implement it in the same way of vertices with indegree 1 we just described, with the only exception that a peg is initially placed in the position marked by the black arrow.

2.3.2 Vertices with outdegree 1

We first discuss vertices with outdegree 1 and indegree 2, which are implemented as shown in Figure 4 (a). The indented ways to play this gadget carries the token peg from any of the two input positions marked with the black arrows, to the output position marked with the white arrow. Notice that during the corresponding sequence of moves, the players will need to jump over the token peg along the solid black line. When this happens, the old token peg is removed from play, nevertheless, instead of thinking of the resulting state of the board as a configuration with no token peg, we promote the jumping peg to become the new token peg.

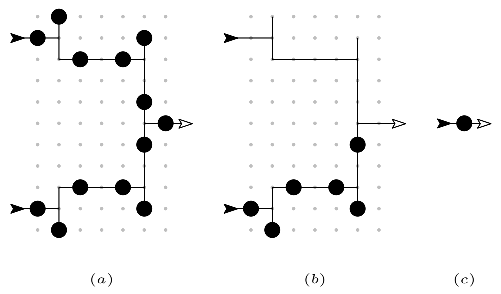


Figure 4 Gadgets for vertices with outdegree 1 and (*c*) indegree 1; (*a*) indegree 2. Picture (*b*) shows how the gadget in (*a*) looks like once players played it, which intuitively corresponds to a vertex already visited by the token in the associated DVG instance.

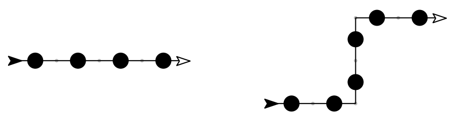


Figure 5 Wire gadgets.

As in the gadget encoding a vertex with outdegree 2, also in this case the first player making a move is also the player making the final move –placing the token peg in the output position (i.e., playing a gadget changes the turn of the next player to play).

We now argue that any player that deviates from the prescribed strategy is bound to lose. As in the previous case, notice that if a player makes a move that brings the token peg outside of the solid lines, or in the opposite direction w.r.t. the intended play, then the opponent can respond by playing the dummy move and winning the game. Notably, this also ensures that a token peg cannot be brought from its initial input position to the other input position (thus traversing the solid lines in the opposite direction).

We encode vertices having indegree and outdegree 1, with the gadget of Figure 4 (c), whose correctness is straightforward.

Wires Wire gadgets are used to encode directed edges in the DVG instance. Such a gadget receives the token peg as an input (which coincides with one of the outputs of the vertex gadget associated with the tail of the encoded edge), and carries it to its output (which coincides with the input of the vertex gadget associated with the head of the encoded edge), through an even number of alternating moves. This ensure that the player making the first move in the wire gadget will also be the next player to play after the wire gadget has been completely traversed. Some examples of wire gadgets are shown in Figure 5. Notice that by repeating the shown pattern, one can lengthen or shorten wires as needed, as well as perform 90, 180, and 270-degrees turns.

This is useful as the planar embedding of the graph G in the DVG instance will determine how to lay wires on the board. It might however happen that such an embedding results in wires with an odd number of moves. In this case, one can restore the desired parity by replacing any straight portion of a wire consisting of 4 moves (see Figure 6 (a)) with the gadget shown in Figure 6 (b), which uses the same input and output positions but requires 9 moves to be traversed.

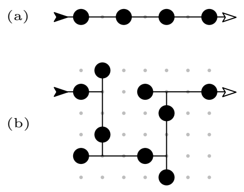


Figure 6 Picture (*b*): changing-parity gadget. Exactly 9 moves are needed to traverse it. The gadget can be used to change the parity of the length of a wire by replacing a portion of a 4-move straight wire (shown in (*a*)).

2.4 Putting all together

As we already described, we build the instance of single-hop Duotaire from a planar embedding of G, by replacing each vertex with its corresponding vertex gadget, and by connecting vertex gadgets through wires in the way prescribed by the planar embedding (see Figure 7 for an example).

When we embed all the gadgets on the board, we shall guarantee that every output of a gadget can be connected to the input of the consecutive gadget through a wire. We point out that all our gadgets are designed in such a way that this is always possible. Indeed, every gadget satisfies the following property: if an input is in the i-th row and the j-th column, where both i and j are even, then each output of the gadget is in i'-th row and j'-th column, with both i' and j' even. This is true for the wire gadget also, which allows us to connect an arbitrary pair of even-even positions.

We now show that if a player has a winning strategy in a DVG instance, then he can also force a win in the corresponding single-hop Duotaire instance. Let us consider the case in which Player 1 has a winning strategy first, and assume that all the gadgets are played in one of the intended ways (as otherwise the deviating player will lose if his opponent plays optimally). Remember that, initially, the token peg is placed on the input position of the gadget corresponding to vertex s (i.e., the black arrow of Figure 3). The two outputs of this vertex gadget correspond to the edges outgoing s in the DVG instance. Player 1 can then play the gadget in such a way that the token peg is carried to the output corresponding to the first edge traversed in his winning strategy, say $e = (s, u)$. This forces the players to traverse the wire corresponding to edge e until the input position of the gadget corresponding to vertex u is reached. Notice that, by our choice of the wire lengths, the turn is now up to Player 2. Since gadgets are always played in the intended way, Player 2 must also bring the token peg to one of the output positions of the gadget, which corresponds to an edge in DVG, say (u, v). Suppose that v is a vertex whose gadget has never been reached by the token peg so far; when the token peg reaches the corresponding input of the v's gadget (on Player 1's turn), Player 1 can respond by moving the peg towards another vertex gadget according to the DVG move prescribed by his winning strategy. Player 1 continues to play according to this scheme until he routes the peg toward a vertex w whose gadget was already traversed. Notice that vertex w must have indegree 2 (and outdegree 1), hence this is the situation depicted in Figure 4 (b) (up to symmetries). Since there are exactly 6 leftover moves along the solid lines (and any deviation causes the deviating player to lose), plus the dummy move, Player 1 is then able to win the game. In other words, any player attempting to move the token peg to a vertex that was already traversed will lose the game. A similar argument applies to the case in which Player 2 has a winning strategy in DVG.

The previous discussion, and the fact that a winning strategy for a single-hop Duotaire instance (if any) can be found by a DFS traversal of the (implicit) game tree (whose height is at most the number of pegs in the instance), allow us to state the following:

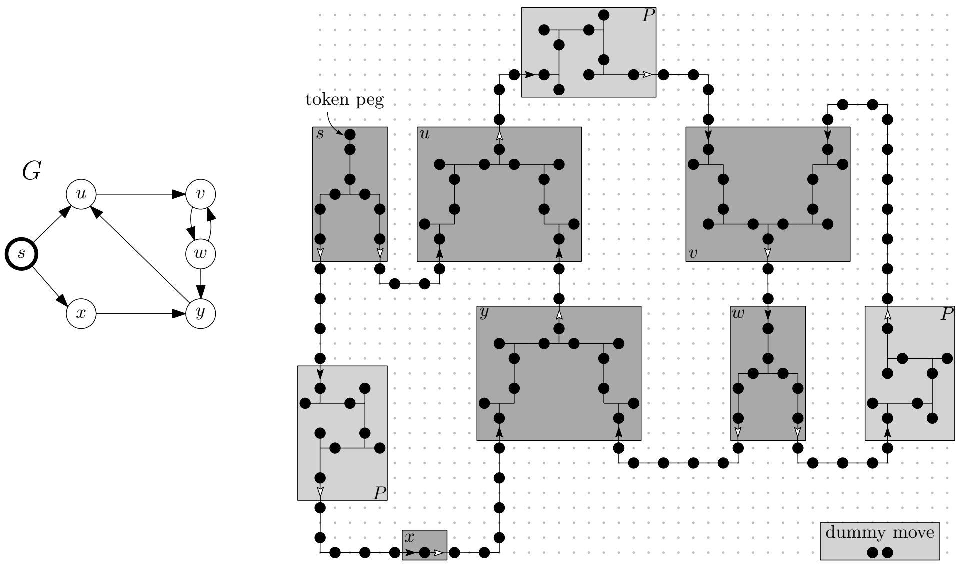


Figure 7 A DVG instance and its corresponding Single-Hop Duotaire one. Gadgets of Figure 6 are used to guarantee that every wire needs an even number of move to be traversed.

▶ **Theorem 1.** *Deciding whether the first player can force a win in single-hop Duotaire is* PSPACE-*complete.*

3 Multi-Hop Duotaire

In this section we prove that the problem of deciding whether the first player can force the win in the multi-hop Duotaire is PSPACE-complete.

Our reduction is from DVG on planar bipartite graphs of maximum degree 3. On the one hand, similarly to the reduction for single-hop Duotaire, both planarity and maximum-degree bounds are useful for embedding the instance on a board. On the other hand, bipartiteness is needed to uniquely associate each vertex with exactly one of the two players. Indeed, if the token is placed on a vertex of the left-hand side of the (vertex) bipartition and, w.l.o.g., it is Player 1's turn, then the player can only shift the token along an edge, if any, to reach a vertex of the right-hand side of the bipartition. Similarly, Player 2 can only shift the token along an edge, if any, to reach a vertex of the left-hand side of the bipartition.

The idea of the reduction is to have a token peg placed on each vertex gadget; however, each token peg needs to be *activated* by another token peg before it can be moved. At the beginning of the game, only the token peg contained in the vertex gadget representing s is active by default. Each token peg is *owned* by a specific player: in the intended play, the token pegs associated with the vertices of the left-hand side of the bipartition can be moved only by Player 1, while the remaining token pegs can be moved only by Player 2. When a token peg, say t, is active, the player owning t is first forced to jump over the token peg that activated t, and then, thanks to suitable *control* pegs, the player is forced to continue moving t along an edge until t reaches a new vertex gadget, if possible, and activates the token peg of that gadget. Therefore, once a player has moved his token peg away from a vertex gadget, no token peg is left in the gadget, and the player that tries to move his token peg inside a previously visited vertex gadget, will lose the game, due to the presence of one dummy move as in the single-hop Duotaire. Similarly to the reduction for single-hop Duotaire, every move other than the dummy move involves token pegs.

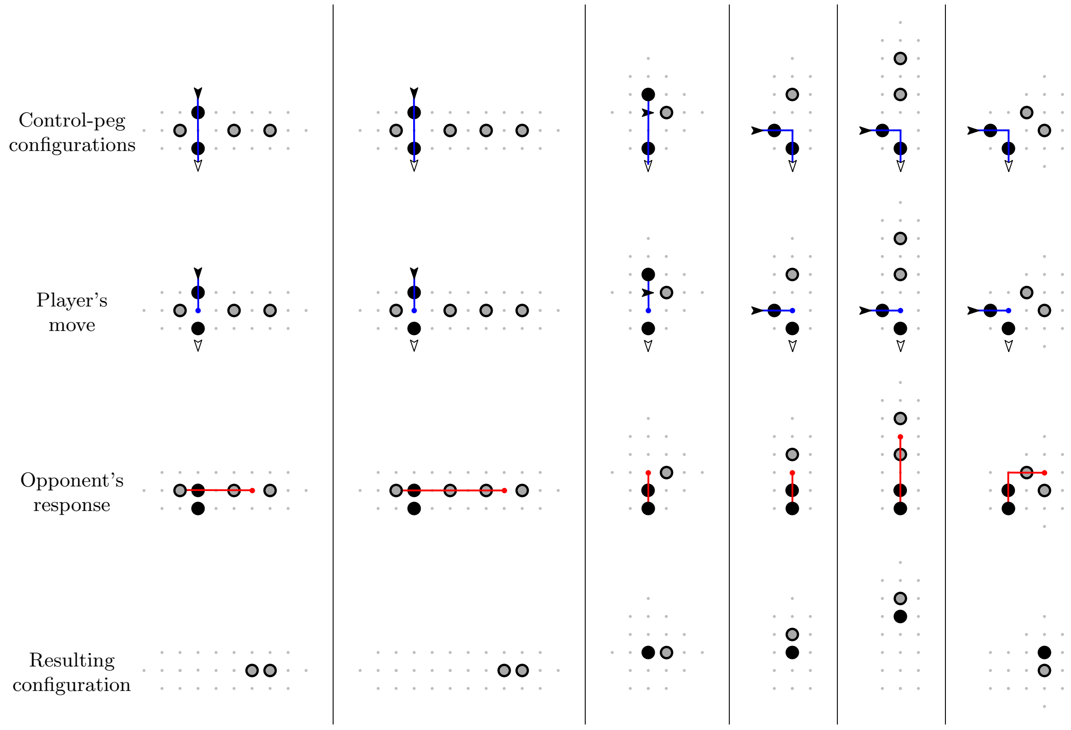


Figure 8 The 6 ways of embedding control pegs to monitor each pair of consecutive black pegs. The dots represent positions of the board nearby the control pegs that need to be empty. The picture shows the opponent's response when the moving player jumps only over the first of 2 consecutive black pegs. In all the 6 cases, the countermove of the opponent generates a dummy move. In the first 2 embeddings, the moving player can jump over the first black peg of the pair, and then also over a control peg. We observe that in this case the token peg has reached an empty area of the board, and the opponent wins the game by playing the dummy move.

3.1 Gadgets

Other than token pegs and (black) pegs that describe the main structure of the gadget, each gadget contains control (gray) pegs that are used to force players to behave in the desired way. Control pegs are embedded on the board to monitor each pair of consecutive black pegs whose corresponding Manhattan distance is equal to 2. We use 6 types of embeddings (see Figure 8). Basically, each configuration forces the player that is jumping over the first of 2 consecutive black pegs to jump also over the other black peg within the same move.

Wires. Wire gadgets are used to encode edges of the DVG instance as well as the vertex s. Each such gadget receives a token peg as an input, and carries it to its output through a single multi-hop move. This ensures that the player that is moving the token peg will also traverse the entire wire. Three examples of wires are shown in Figure 9. To avoid that the player moving the token peg would not traverse the entire wire, control pegs have been added all along the wire. Observe that the embedding of control pegs that monitor a pair of consecutive black pegs on straight wires is independent of the position of the other control gadgets placed along the wire. Finally, notice that a wire can also make 90-degree turns both clockwise and counterclockwise: indeed, the right-most drawing in Figure 9, being symmetric, can be traversed in both directions.

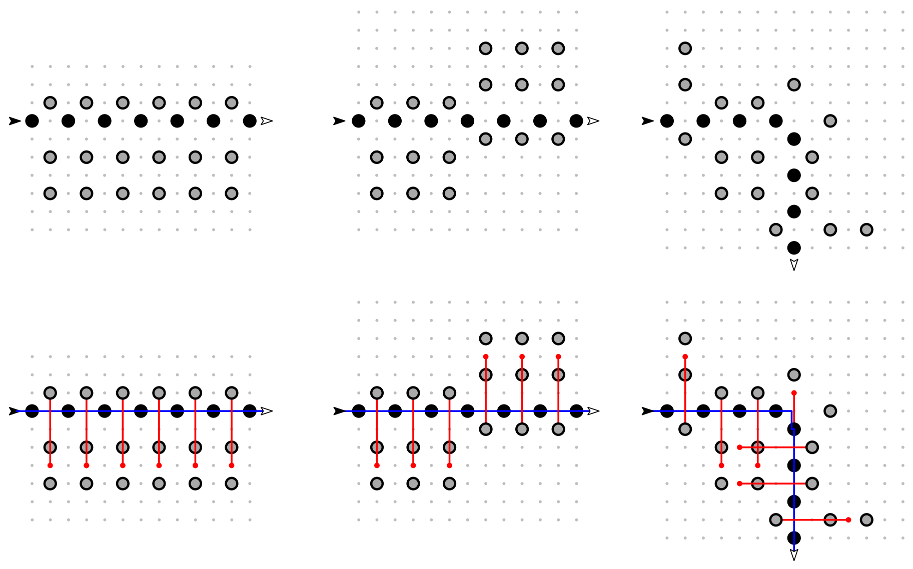


Figure 9 Straight wires and 90-degree turns.

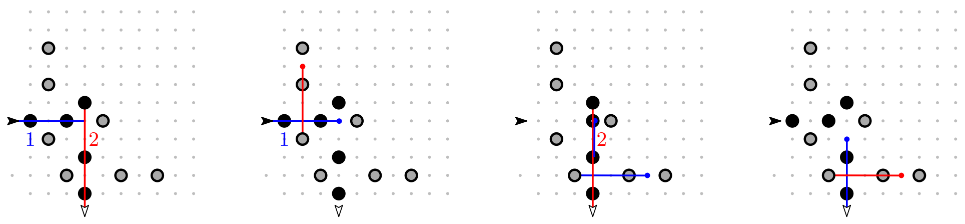


Figure 10 Vertex gadget. The rightmost drawing shows that the player that enters the gadget from its output loses.

Vertices. Vertex gadgets (see Figure 10) are used to encode vertices of the DVG instance that are different from s. Each such gadget receives the token peg of a player as an input, and outputs the token peg owned by the other player in exactly 2 moves. This is done thanks to the presence of control pegs which force the player that is entering the vertex gadget with a token peg he owns, to terminate his move exactly when his token peg is aligned with the token peg owned by the other player. Thus, the gadget forces the opponent to move his token peg to the output of the gadget. We observe that no token peg remains inside a vertex gadget once it has been visited. Furthermore, if we think of the game as if it were played on a chessboard, we also observe that if the input token peg comes from a black (resp., white) square of the chessboard, then the token peg contained in the vertex gadget is on a white (resp., black) square. Finally, we observe that a player that cheats and tries to enter the vertex gadget from its output position rather than from its input position, will lose the game.

Branches and one-way gadgets. A branch (see Figure 11) is used to model both the merge of two distinct wires into a single wire (i.e., vertices with indegree 2), as well as the split of one wire into two distinct wires (i.e., vertices with outdegree 2). The gadget takes one token peg in one of the two possible input positions, and forces the moving player to exit from the gadget either in the (unique) output position, or in the other input position. Branches are not sufficient by themselves to model wire splits and merges, and they need to be used together with *one-way* gadgets.

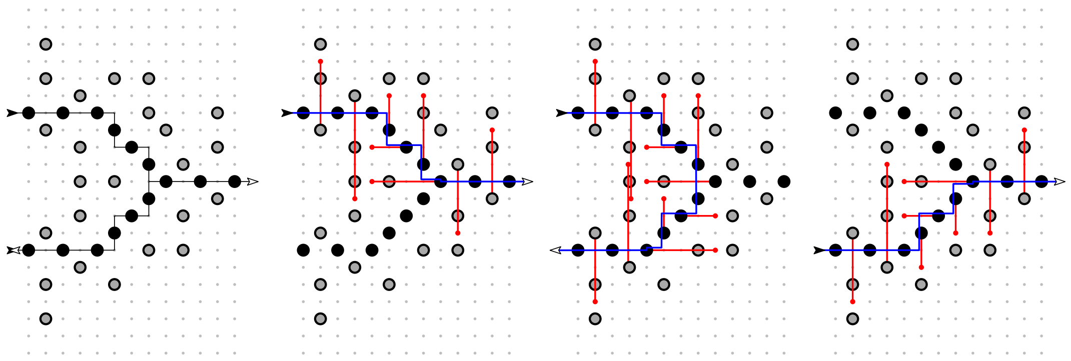


Figure 11 A branch.

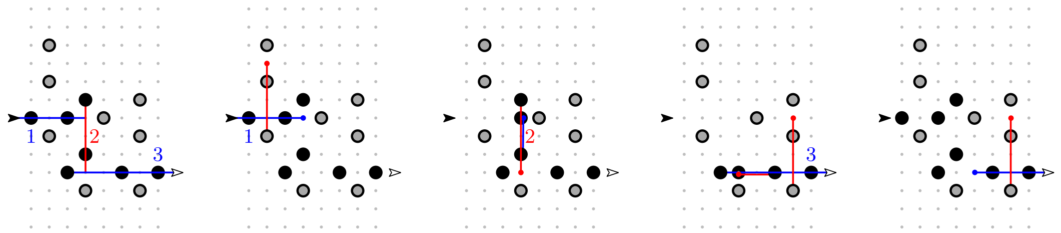


Figure 12 One-way gadget. The rightmost drawing shows that the player that enters the gadget from its output loses.

One-way gadgets (see Figure 12) are used to avoid that players can cheat by visiting branches starting from their corresponding output positions rather than from their corresponding input positions. The one-way gadget takes a token peg as an input, and outputs the token peg with 3 multi-hop moves. A one-way gadget contains two additional token pegs, one token peg for each player, each of which must be activated before it can be moved. Similarly to the vertex gadget, a one-way gadget outputs a token peg that is different from the one that entered the gadget. However, differently from the vertex gadget, the one-way gadget outputs a token peg owned by the same player that entered the gadget. A one-way gadget is designed in such a way that a player that cheats and tries to enter such a gadget from its output rather than its input, will lose the game.

The merge of two wires can now be modelled with a branch whose output is attached with the input of a one-way gadget. Similarly, the split of a wire into two wires can be modelled by using a branch and two one-way gadgets, whose inputs are attached to the output and to any of the 2 inputs of the branch, respectively.

3.2 Putting all together

The token pegs are placed over the (chess)board in such a way that, w.l.o.g., Player 1 owns the token pegs that are placed on black squares, while Player 2 owns the token pegs that are placed on white squares. We observe that this induces the position of each vertex gadget on the board, according to the vertex-player association (we recall that each player has been associated with a specific side of the bipartition). Therefore, when it is Player 1's turn, the (unique) active token peg is on a black square, and the player can only move such a token peg to activate a token peg placed on a white square, if possible. Similarly, when it is Player 2's turn, the (unique) active token peg is on a white square, and the player can only move such a token peg to activate a token peg placed on a black square, if possible.

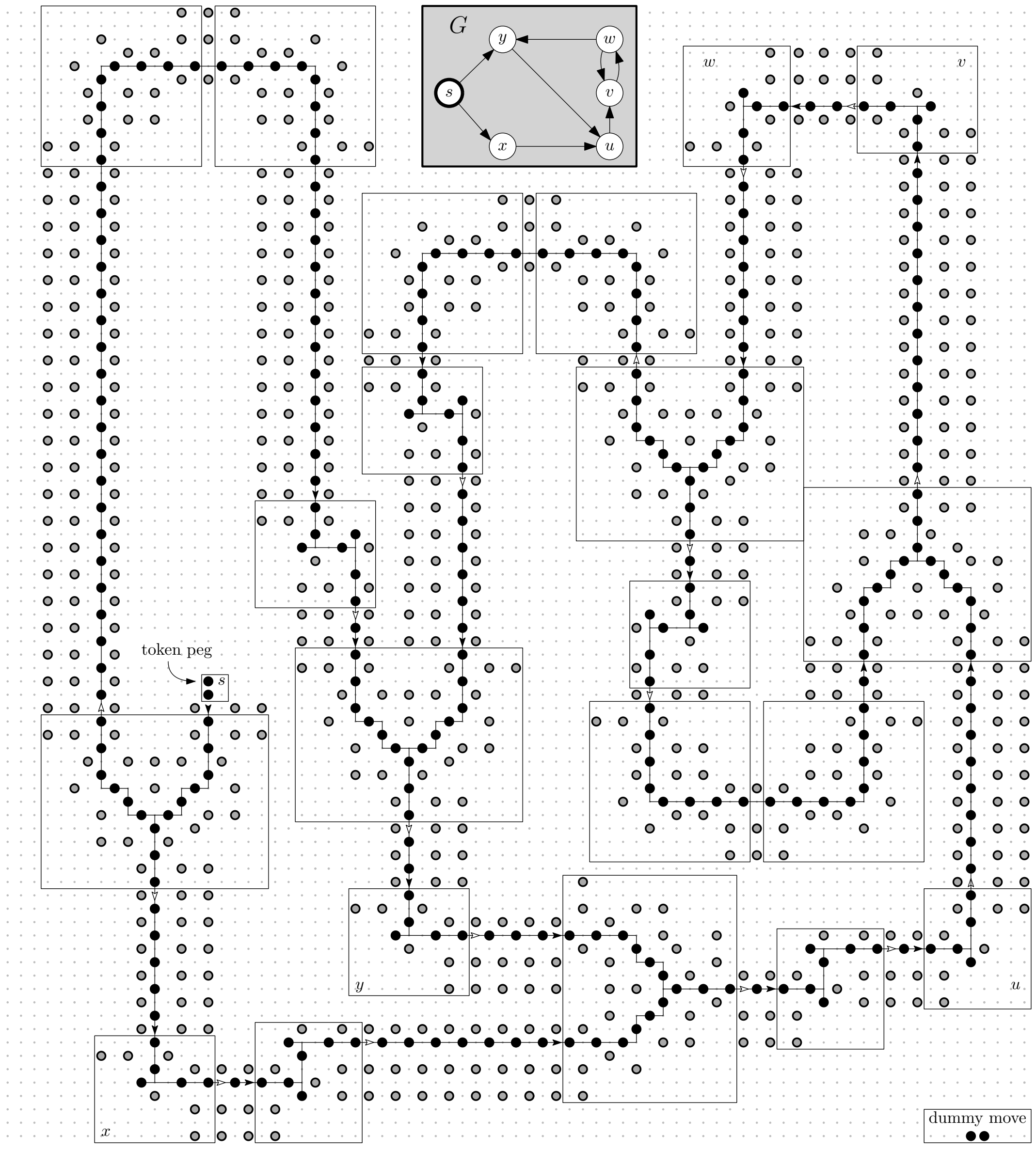
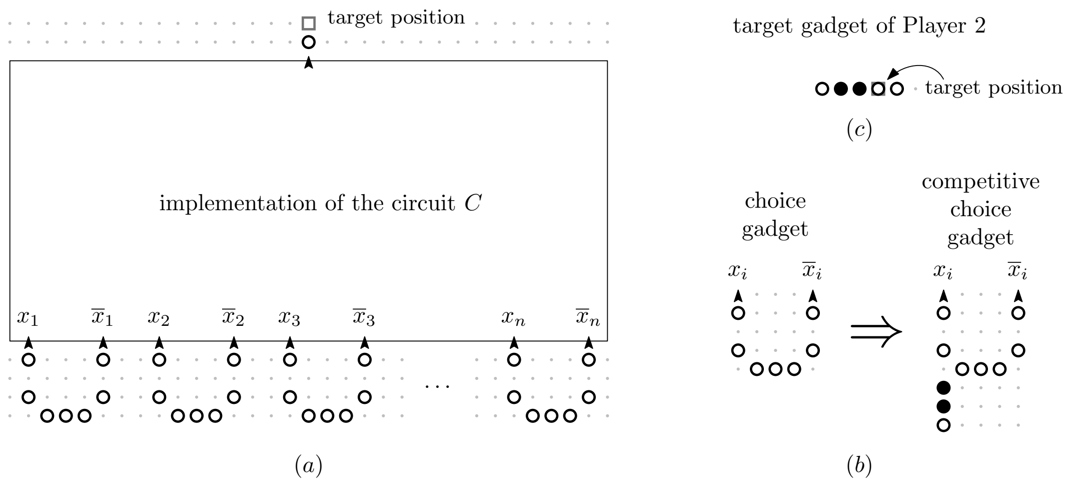


Figure 13 A DVG instance and its corresponding Multi-Hop Duotaire one. For the sake of readability, some (redundant) one-way gadgets have been removed.



target position
implementation of the circuit C
x_1 $\overline{x}_1$ x_2 $\overline{x}_2$ x_3 $\overline{x}_3$ x_n $\overline{x}_n$
...
(a)
target gadget of Player 2
target position
(c)
choice gadget
competitive choice gadget
x_i $\overline{x}_i$
(b)

Figure 14 The three ingredients of our reduction for the 2-player version of Solitaire-Reachability. Picture (a) shows (the form of) the peg configuration resulting from the application of the transformation given in [15] to a circuit C (equivalent to the formula F). There is a choice gadget for each variable. Picture (b): the competitive choice gadget. Picture (c): the target gadget of Player 2. If Player 1 is forced to make the only available move in the gadget, then Player 2's target position becomes reachable.

Similarly to the single-hop Duotaire, the embedding shall guarantee that each output of any gadget can be connected to the corresponding input of the consecutive gadget. This could be a problem when the output is, for example, in an even-even position while the input is in a odd-odd position.[2] However, the one-way gadget can be used (also) to restore the desired parity. See Figure 13 for an example multi-hop Duotaire instance obtained from a corresponding DVG instance.

By using similar arguments to the ones we discussed for single-hop Duotaire, we can now state the following:

▶ **Theorem 2.** *Deciding whether the first player can force a win in multi-hop Duotaire is* PSPACE*-complete.*

4 2-player Solitaire Reachability

In this section we prove that the 2-player version of Solitaire Reachability discussed in the introduction is PSPACE-complete. We present the reduction when the winning rule is the following: a player wins when either he reaches his target position, or his opponent has no available moves. Next, we show how to adapt the reduction for other winning conditions.

The main idea of the reduction is borrowed from the PSPACE-completeness reduction of the *bounded 2-player constraint logic* presented in [17]. The reduction is from *Positive Conjunctive Boolean Formula Game* (POS CNF), where one wants to understand whether the first player can force a win in the following game. We are given a *monotone* boolean formula F in CNF, i.e., a formula containing positive literals only. The two players alternate choosing some variable of F that has not yet been chosen, and decide whether to assign either true or false to that variable. The game ends after all variables of F have been chosen.

[2] Notice that the cases even-odd and odd-even cannot occur because of the vertex-player association.

The first player wins if and only if F is true; therefore, the second player wins if and only if F is false. Observe that since F is monotone, the first player has convenience to set the chosen variables to true, while the second player will always set his variables to false.

It is well known that any CNF Boolean formula can be transformed, in polynomial time, into an equivalent planar Boolean circuit of NAND gates only, which in turn can be converted into an instance of Solitaire-Reachability [15]. More precisely, we use this fact to convert F into a configuration of white pegs and a target position having the form showed in Figure 14 (a), where there is a *choice gadget* for each input variable. Each choice gadget allows to set the corresponding variable either to true or false. The reduction given in [15] implies the following: there exists a sequence of moves placing a peg in the target position if and only if the chosen Boolean assignment satisfies F.

In our reduction we first replace each choice gadget with a *competitive choice gadget* (see Figure 14 (b)). The competitive choice gadget allows Player 1 to set the corresponding variable either to true or false, unless Player 2 forces Player 1 to set the variable to false. Therefore, the player that plays the gadget first essentially decides the assignment of the variable. To complete the description of the reduction, we add the target gadget of Player 2 (see Figure 14 (c)), in which the target position of the player is occupied by a peg of his opponent, and sufficiently many dummy moves that can be performed only by Player 2.

Clearly, since F is monotone, both players have convenience to first play all the competitive choice gadgets, and, once the truth assignment has been chosen, Player 1 has a sequence of moves that allows him to win the game only if the truth assignment satisfies the formula. Conversely, if the truth assignment does not satisfy the formula, then Player 1 runs out of moves before Player 2, frees the target position of his opponent, and Player 2 wins the game. Therefore, Player 1 can force a win in our instance if and only if he can force a win in the POS CNF instance.

Our reduction can be adapted to other winning conditions:

- The only way a player can win is reaching the target position. In this case, we can assume that a player with no available moves can skip his turn, and that the game can end with a draw, whenever no player can move and no target position has been reached yet. The reduction is exactly the same.
- There is no target position and the first player that has no available moves loses the game. The reduction is the same but we remove the target gadget of Player 2, and we add a number of sufficiently large moves for Player 1, that are triggered only if a (white) peg is placed in the old target position of Player 1.

References

1 Matteo Almanza, Stefano Leucci, and Alessandro Panconesi. Trainyard is np-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 2:1–2:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FUN.2016.2.

2 John D. Beasley. *The Ins and Outs of Peg Solitaire*. Oxford University Press, Oxford Oxfordshire ; New York, 1985.

3 John D. Beasley. Solitaire: Recent Developments. *arXiv:0811.0851 [cs, math]*, nov 2008. arXiv: 0811.0851. URL: http://arxiv.org/abs/0811.0851.

4 John D. Beasley. John and Sue Beasley's Webpage on Peg Solitaire, oct 2015. URL: https://web.archive.org/web/20151010215541/http://jsbeasley.co.uk/pegsol.htm.

5 George I. Bell. A Fresh Look at Peg Solitaire. *Mathematics Magazine*, 80(1):16–28, feb 2007. URL: `http://www.jstor.org/stable/27642987`.

6 George I. Bell, Daniel S. Hirschberg, and Pablo Guerrero-Garcia. The minimum size required of a solitaire army. *arXiv:math/0612612*, 2006. arXiv: math/0612612. URL: `http://arxiv.org/abs/math/0612612`.

7 Arie Bialostocki. An application of elementary group theory to central solitaire - ProQuest. URL: `http://search.proquest.com/openview/5b321e9dd162151b018ab7d63304daf2/1?pq-origsite=gscholar`.

8 Bela Csakany and Rozalia Juhasz. The Solitaire Army Reinspected. *Mathematics Magazine*, 73(5):354–362, dec 2000.

9 Erik D. Demaine, Fermi Ma, Ariel Schvartzman, Erik Waingarten, and Scott Aaronson. The Fewest Clues Problem. In *8th International Conference on Fun with Algorithms (FUN 2016)*, volume 49, pages 12:1–12:12, 2016.

10 Edsger W. Dijkstra. The checkers problem told to me by M.O. Rabin, 1992. URL: `http://www.cs.utexas.edu/users/EWD/ewd11xx/EWD1134.PDF`.

11 Martin Gardner. *The Unexpected Hanging and Other Mathematical Diversions*. University of Chicago Press, Chicago, nov 1991.

12 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

13 J. P. Grossman. Periodicity in one-dimensional peg duotaire. *Theor. Comput. Sci.*, 313(3):417–425, 2004. `doi:10.1016/j.tcs.2002.11.003`.

14 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`.

15 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`.

16 Robert A. Hearn. *Amazons, Konane, and Cross Purposes are PSPACE-complete*, page 287–306. Mathematical Sciences Research Institute Publications. Cambridge University Press, 2009. `doi:10.1017/CBO9780511807251.015`.

17 Robert A. Hearn and Erik D. Demaine. *Games, puzzles, and computation*. AK Peters Wellesley, 2009.

18 Ross Honsberger. A problem in checker jumping. *Mathematical Gems II*, pages 23–28, 1976.

19 David Lichtenstein and Michael Sipser. GO is polynomial-space hard. *J. ACM*, 27(2):393–401, 1980. `doi:10.1145/322186.322201`.

20 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`.

21 Cristopher Moore and David Eppstein. One-Dimensional Peg Solitaire, and Duotaire. *arXiv:math/0008172*, 2000. arXiv: math/0008172. URL: `http://arxiv.org/abs/math/0008172`.

22 Bala Ravikumar. Peg-solitaire, string rewriting systems and finite automata. *Theor. Comput. Sci.*, 321(2-3):383–394, 2004. `doi:10.1016/j.tcs.2004.05.005`.

23 Ryuhei Uehara and Shigeki Iwata. Generalized Hi-Q is NP-Complete. *IEICE TRANSACTIONS (1976-1990)*, E73-E(2):270–273, feb 1990.