Seminar Advanced Algorithms and Data Structures
Student Report

On the PSPACE-completeness of Peg Duotaire and other Peg-Jumping Games
Written By: D. Bilò, L. Gualà, S. Leucci, G. Proietti, M. Rossi
Advisor: Stefano Leucci
Buddy: Luca Mondada

Jonas Passweg

October 2018

1 Introduction

The paper [1] analyzed here, proves PSPACE-completeness of three Peg-Jumping games, by identifying three PSPACE-complete problems that can be reduced to the respective games. This is an open problem in the literature, solved by this paper.

To best understand the proof, and why it shows PSPACE-completeness, I want to start by clarifying what games we are talking about. I will then try to introduce the proof by explaining key concepts, followed by explaining the puzzle pieces, that will together form the reduction. Note that the three Peg-Jumping games are two-player games and that proving the PSPACE-completeness of one of these Peg-Jumping games means proving that the problem of deciding whether the first player can force a win in this game is PSPACE-complete.

1.1 Peg-Jumping games

Peg-Jumping games consist of pegs on a board. A peg can be thought of as a playing piece that completely fills one position on the board. Allowed moves are jumps of a peg over an adjacent peg into an empty position. 3 such moves can be seen in figure 2[1]. A peg, that is jumped over, is removed from the board. Furthermore in the paper, the board is (implicitly) restricted to a grid, so that every position has only 4 adjacent positions, except for positions at the edge which have less. Many variations of these games exist. A compiled list of all games mentioned in the paper can be found in table 5 on page 10.

I will shortly present 2 of the 3 games analyzed in depth in the paper. I will then show that they are in PSPACE and I will go through their PSPACE-completeness proof based on a reduction. Before that, I want to make sure that we understand why these reductions work. The three games are Single-Hop Peg Duotaire, Multi-Hop Peg Duotaire and a 2-player version of Solitaire Reachabilty. We will concentrate on the former 2 games because their reductions are similar and therefore understanding one helps understanding the other.

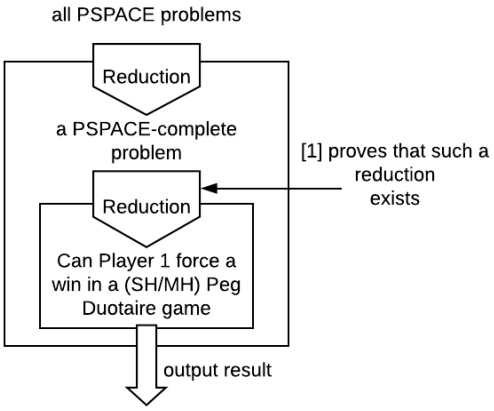

Figure 1: Idea for Reduction, SH = Single-Hop, MH = Multi-Hop

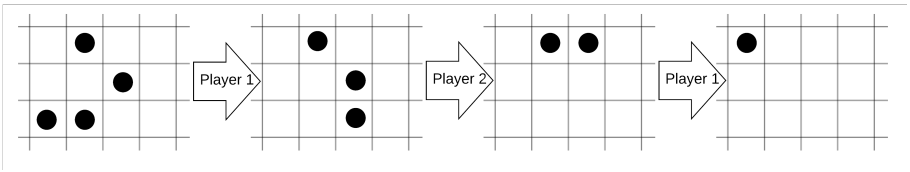

Figure 2: Single-Hop Peg Duotaire instance. Player 1 wins since player 2 has no moves left. Each arrow is between two board configurations, left the one before the move and right the one after.

1.2 Reduction

The difficulty in the paper is to show the reduction of a PSPACE-complete problem to each Peg-Jumping game. As you may remember, to be PSPACE-complete a problem has to be solved using an amount of memory that is polynomial in the length of the input and every other PSPACE problem must be reducible to it in polynomial time. If we, show that a PSPACE-complete problem is reducible to a PSPACE problem, then the latter must also be PSPACE-complete since every PSPACE problem can then be reduced to the first, this then to the latter, resulting in a reduction to the latter problem, seen in figure 1. This is exactly what we are going to do to prove the PSPACE-completeness of the Peg-Jumping games.

2 Single-Hop Peg Duotaire

In the game Single-Hop Peg Duotaire, 2 players take turns each alternatingly making one move. All the pegs have the same color, and the first player not able to make a move, looses the game. The initial configuration and number of the pegs is arbitrary. An instance of this game can be seen in figure 2.

[1]For now, one can ignore the player number in the arrow.

2.1 PSPACE of Single-Hop Peg Duotaire

Apart from the reduction we have to show that Single-Hop Peg Duotaire is PSPACE. This is summarized in the paper and does not need a lot of explanation. Consider the implicit game tree of an instance of Single-Hop Peg Duotaire. Its height is bounded by the number of pegs in this given instance since one vanishes with every move. Therefore, an algorithm can do a depth-first-search on the tree and will find any winning strategy, if it exists. This algorithm is allowed to store the entire path, since the storing of a board configuration is linear in the number of pegs and all the possible moves on a configuration are enumerable and also linear in the number of pegs. More on that can be found in [2].

2.2 Reduction with Directed Vertex Geography

The reduction to Peg Duotaire will be done with something known as directed vertex geography (DVG), which is a known PSPACE-complete problem. In other words we are going to simulate a DVG instance with the help of a Peg Duotaire game in such a way, that iff player 1 can force a win in the DVG instance, it can also force a win in the Peg Duotaire instance.

A DVG instance consists of a graph with directed edges and a single starting vertex. A move is following an edge in the correct direction from the current vertex[2] to a new vertex. To prohibit repetitions we then remove the vertex at which the edge we followed started and the new vertex becomes the current vertex. The player with no move i.e no outgoing edges left, has lost.

We need to further transform our DVG graph to be able to simulate it with a Peg Duotaire game. However, how this is achieved is not important for the proof itself. Let's therefore assume that the graph of our DVG problem has these properties (section 2.3) and is still PSPACE-complete.

2.3 Summary: DVG graph to be modelled by Single-Hop Peg Duotaire game

- graph is planar and bipartite
- one vertex s is set as the starting point.
- s has outdegree 2 and no incoming edges
- all other vertices have either
  - indegree 1 and outdegree 2
  - indegree 2 and outdegree 1
  - indegree 1 and outdegree 1

Note that the DVG graph only needs to be bipartite for the reduction of the Multi-Hop Peg Duotaire version.

2.4 Gadgets

The idea behind reducing the DVG problem to a Single-Hop Peg Duotaire problem, is to embed the planar DVG graph on a peg board. Each vertex and each edge of the DVG graph will be replaced by a predefined peg configuration on the board, corresponding to the specification of the vertex/edge. These board configurations are called gadgets. Like edges in the DVG graph, the edge gadgets connect the vertex gadgets in the Single-Hop Peg Duotaire instance.

[2]The current vertex is trivially equal to the starting vertex at the beginning of the DVG instance.

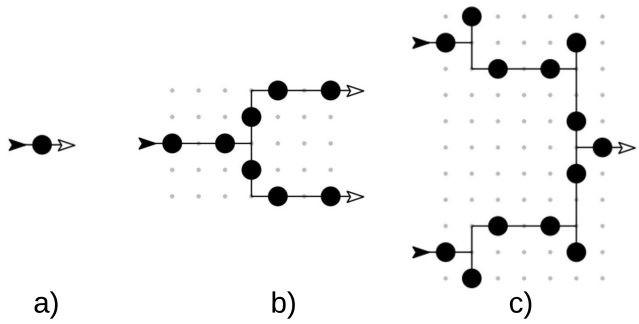

Figure 3: a) gadget for $v_{(1,1)}$; b) gadget for $v_{(1,2)}$; c) gadget for $v_{(2,1)}$. $v_{(a,b)}$ is a vertex with indegree a and outdegree b. Reprinted from [1]

If a DVG graph is piece-wise replaced by these gadgets, the resulting Single-Hop Peg Duotaire game has the following property: iff player 1 can force a win in the DVG instance, it can also force a win in the Single-Hop Peg Duotaire instance.

Since gadgets are on a peg board, they are designed to be playable. Playable means here, that players can alternatively make a jump to move a peg from the start of the gadget to the end. Doing so is playing a gadget.

Let's look at figure 3.b for more details. The dots represent the pegs, the black arrow the starting point and the white arrows the ends of the gadget. There is no peg on the black arrow, making the gadget unplayable. However after a peg is placed there, the players can make alternating moves to move this new peg to one of the white arrows.

Using this, we can connect an end of one of the gadget with the start of the next, which results in a chain of gadgets. Note that if the first gadget of the chain is playable, then so is the whole chain.[3] We start by looking at the simplest gadget and work through all of them.

2.4.1 Gadget for vertices with indegree 1, outdegree 1

In figure 3.a you can see the gadget for a vertex with indegree 1 and outdegree 1. Trivially jumping with an additional peg from the black arrow to the white arrow is playing the vertex gadget. This needs exactly one move.

2.4.2 Gadget for vertices with indegree 1, outdegree 2

A gadget of a vertex with one incoming and two outgoing edges is seen in figure 3.b. Remember that each vertex gadget is connected to several edge gadgets. Therefore choosing which end i.e white arrow is reached determines which edge gadget is playable. This corresponds to choosing the outgoing edge in a DVG instance.

This gadget is therefore designed such that the player making the first move inside the gadget, also can choose which end is reached. Alternating moves along the line achieves this, since the decision happens on an odd move, counting for this vertex gadget only.

Therefore gadgets need to guarantee, that if player 1 makes the first move in this vertex gadget, player 2 must make the first move in the next vertex gadget. This is done by giving each vertex gadget an odd number of moves and each edge gadget an even number of moves to be played.

[3]Note that we ignore having multiple starting and ending points for this claim.

2.4.3 Gadget for the starting vertex

The starting vertex gadget works like the gadget for a vertex with indegree 1 and outdegree 2 just mentioned before but it is playable at the start. This means that there is a peg on its black arrow at the start.

2.4.4 Gadget for vertices with indegree 2, outdegree 1

This vertex gadget moves the peg from one of the black to the white arrow (See figure 3.c). Notice, that we change the peg that travels along the line on jump 2 and 5.

2.4.5 Gadgets for edges

At last, we also need a modelling for the edges in the DVG graph, called edge gadgets or wires. We already know, that the number of moves must be even, in order for the vertex gadgets to be played alternatingly among the players. The pattern of a peg followed by an empty space can be repeated to achieve shorter and longer paths or turns of 90 degrees in either direction. Note that both ends need each to have an arrow i.e. an empty field and a peg adjacent to it.

2.5 Putting the gadgets together

We can now build a peg game by replacing each vertex with its corresponding gadget and connect them with wires. Since the DVG graph is planar[4], we can do this without the wires crossing each other on the peg board.

Furthermore every gadget was designed as to end on an even row and column, if it also starts on an even row and column. Additionally there is a way to change a 4 move segment of a wire with a 9 move segment of the same size, to restore parity if needed. With that in mind we are confident that there is a arrangement of gadgets on a board, such that it corresponds to the DVG graph.

2.6 Prevent illegal moves and the dummy move

Until now we assumed that players make the move we want. However we will see, that only those moves do not lead to a loss. Since we always only have one gadget that is playable or currently being played, there are obviously no more than 2 possible moves a player can choose from. Except for the third move in the gadget for a vertex with indegree 1 and outdegree 2, one of these moves will lead to a unplayable configuration on the board i.e. no moves are left. We call this move an illegal move. The other move is the one we wish the player to take.

An additional isolated dummy move i.e. two pegs out of reach of any other pegs, is introduced, which gives the player the opportunity to:

- counter an illegal move with the dummy move and therefore win
- counter a dummy move with an illegal move if the dummy move was misused and therefore also win

Furthermore we want to prohibit repetition. This is simple, since playing a gadget correctly, will leave it nearly empty and will therefore render it unplayable. Note that this is also the case for the gadget for the vertex with indegree 2 and outdegree 1, due to the last jump sharing the same peg. Note that the jumps 2 and 5 in this gadget are the ones that ensure, that a player cannot

[4] A planar graph is, simply put, a graph that can be drawn on a plane such that the edges do not cross.

move from one starting position to the other. More generally, this and the fact that already played gadgets are unplayable ensures that we never get to move from an ending to a starting point.

2.7 Finalizing reduction

The initial goal was to show, that only a winning strategy in a DVG instance, will also lead to a win for a "perfect" player in the corresponding Single-Hop Peg Duotaire instance. We have already shown, how illegal moves are handled, and can therefore assume, that every vertex gadget and edge gadget is played correctly. Since every vertex gadget has at least one outgoing wire, the loosing player (the one not having a winning strategy), will eventually move towards an already played vertex gadget. Since every wire has an even number of moves, it is the same player that will also finish with the last move of the wire. The winning player can now play the dummy move and win. Note that this also works when the last player moves to the other starting point of an already played gadget for a vertex of indegree 2 and outdegree 1, as it only adds an even number (6) of moves.

Theorem 1 *Deciding whether the first player can force a win in Single-Hop Peg Duotaire is PSPACE-complete.*

3 Multi-Hop Peg Duotaire

Now we want to prove the same complexity classification for Multi-Hop Peg Duotaire. The difference here is, that an arbitrary number of jumps with the same peg counts as one move, removing the restriction of having one jump per turn. Note that the Single-Hop jumps in figure 2 may all be made during one move in the Multi-Hop version.

3.1 PSPACE of Multi-Hop Peg Duotaire

The argument for Multi-Hop Peg Duotaire being in PSPACE is the same as with the Single-Hop version, except for a few modifications.

- Now an arbitrary amount of pegs on the board can be removed in one turn. The length of the implicit game tree is still bound by the number of pegs.
- The moves are enumerable since the ordering of the moves can be created with the lexicographical ordering of the Single-Hop moves.

3.2 Reduction with Directed Vertex Geography

Again, the proof is provided by a reduction from the DVG problem. With that, we show the PSPACE-completeness of deciding whether the first player can force a win in the Multi-Hop Peg Duotaire instance. The DVG graph has the same properties as in the Single-Hop version, seen in section 2.3.

3.3 Idea for Directed Vertex Geography reduction

In the Multi-Hop version we no longer have the guarantee, that each move is exactly one jump. Without thinking about a bad player, we will not understand why changing the gadgets is needed. Therefore we already now assume to have a player that does not behave as wished.

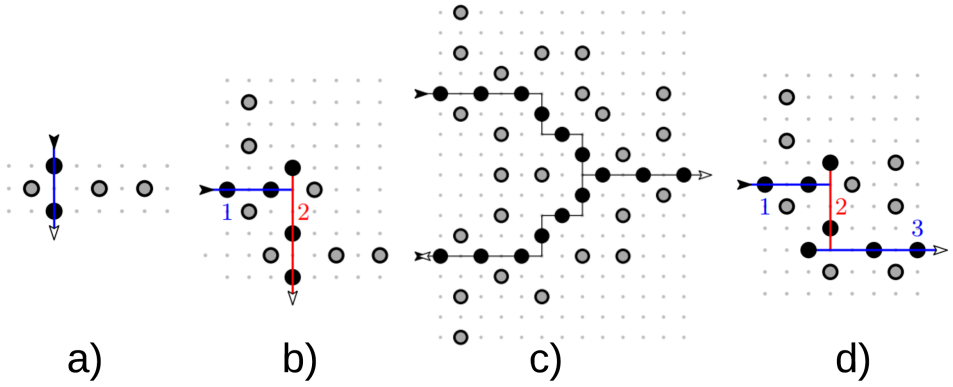

Figure 4: a) controlling pegs (grey) configuration; b) vertex gadget for Multi-Hop version; c) branch gadget; d) One-Way gadget. Reprinted from [1]

Two concepts are introduced to guarantee that any player makes the exact amount of jumps we wish for. Firstly, controlling pegs blocks the player of ending the turn with to few jumps, by giving the other player a possibility to then force a win. Secondly, jumping over a peg into the empty position p and then needing to take another peg to jump over the one at position p guarantees that a player needs to end the turn at this position and therefore limiting the maximal amount of jumps in one turn.

3.4 Gadgets

Let's for now assume, that the controlling pegs work as intended. The gadgets are more confusing than with the Single-Hop version. Most notably, a vertex gadget before also included the logic for how many edge were incoming and outgoing in the corresponding vertex of the DVG graph. Now the task of a vertex gadget is to switch the player. The branching is handled by a separate gadget.

3.4.1 Gadget for edges

Wires (gadgets for edges) are again bidirectional and controlling pegs are between each 2 consecutive pegs i.e. they block the player from ending the turn between those two pegs. A wire therefore moves a peg from one end to the other end of the wire in one move.

3.4.2 Gadget for the starting vertex

The gadget for the starting vertex is now only two adjacent pegs, since it does not have to handle the branching logic. Note that this it the only instance of two adjacent pegs at the start of the game.

3.4.3 Gadget for other vertices

Each vertex different from the starting vertex is modelled by two Multi-Hop moves. The incoming move with peg p in the new vertex gadget is halted, since another peg needs to jump over p to get

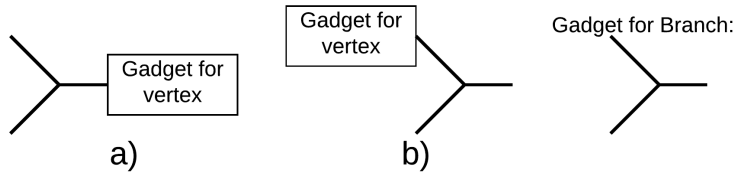

Figure 5: How branch gadgets can be connected to a vertex gadget. a) gadget which corresponds to a vertex with indegree 2 in the DVG graph; b) gadget which corresponds to a vertex with outdegree 2 in the DVG graph. Note that the One-Way gadgets are omitted

out of the vertex gadget. This switches the player. Every vertex gadget is now the same and can be seen in figure 4.b.

Both Multi-Hop jumps are forced by controlling pegs. Furthermore a player cannot enter the vertex from the other side as the other player will be able to force a win.

3.4.4 Gadget for branches

The gadget for branches has 2 inputs and 1 output. It forces the player who enters from one of the two inputs to either move to the one output or the other input with the help of controlling pegs. However every end/start can be reached for every other start/end. This allows us to simulate two outgoing edges by placing the branch gadget after the vertex gadget (seen in figure 5.b) or two incoming edges by placing the branch gadget before the vertex gadget (seen in figure 5.a).

We must limit the possibility of leaving the branch gadget through any given exit. This is done with the One-Way gadget, which can be added at every connection point of the branch gadget. In practise a maximum of 2 are needed per branch gadget.

A smaller version of a branch gadget without the controlling pegs looks just like the vertex gadget from the Single-Hop version with indegree 1, outdegree 2. However bigger branch gadgets are used so that the controlling pegs of the two inputs do not collide. This is seen in figure 4.c.

3.4.5 One-Way gadget

The one-way gadget has one entrance and one exit and cannot be played from exit to entrance. It is an extension of the vertex gadget and is seen in figure 4.d. To pass through a one-way gadget players need 3 moves instead of 2. This allows the current player before the gadget to keep playing afterwards.

3.5 Controlling pegs

In the paper 6 different implementations of controlling pegs are presented, each needed for different configurations of the normal pegs. These are placed in between two pegs so that player 1, jumping over the first peg, places his/her peg in between those two, will also have to be jumping over the second peg. Else player 2 will be able to force a win by creating another dummy move with the controlling pegs. Additionally player one cannot jump over the controlling peg, as this leads to no more playable moves. One implementation can be seen in 4.a. Note that this can also be done such that the second jump is in a 90 degrees angle to the first jump, to get a turn in a wire for example.

3.6 Note on bipartition

The property bipartite of the graph is needed in the Multi-Hop version. This is due to the fact that one move is always made using the same peg. If the game was on a chess board, a white peg would always jump to white fields and vice-versa. Similarly to a gadget in the Single-Hop version needing to be playable such that players can make moves in it, every move in the Multi-Hop version needs to be activated, for the other player to start this move. An activation is (again similarly to the Single-Hop version) placing a peg next to the start of the next move, so that the player can jump over the newly placed peg. Coming back to the chess board, for a peg to activate another, one must be initially on a white field and the other on a black field. Therefore only "black" pegs can activate "white" vertices and vice versa. This is why we need a bipartition to assure that a "white" peg will always move to a "black" vertex and a "black" peg to a "white" vertex.

3.7 Putting the gadgets together

With the help of bipartition and using the arguments of Single-Hop Peg Duotaire, we can connect all the gadgets to embed a whole DVG graph. One-way gadgets can be added to fix a shift by 1 diagonal field to fix parity.

The dummy move is still present and achieves the same as in the Single-Hop version, i.e. it prevents an illegal move by providing a counter move.

3.8 Finalizing reduction

With arguments similar to the Single-Hop version we get:

Theorem 2 *Deciding whether the first player can force a win in Multi-Hop Peg Duotaire is PSPACE-complete.*

4 Conclusion

The paper proves the PSPACE-completeness of three Peg-Jumping games by explaining gadgets that, like puzzle pieces, can be put together to simulate a directed vector geography instance. With the Single-Hop version of Peg Duotaire the focus is on showing that one can design those gadgets, so that the players are restricted in the amount of moves they can choose from. The Multi-Hop version differs in that we need to force the players through a vertex in an odd number of moves. The player choosing the outgoing edge from a vertex in a DVG instance is also choosing the corresponding wire in the Peg Duotaire instance. This is done with the help of controlling pegs. The key take-home message, next to the new complexity classifications of peg games, is for me how designing smaller parts of a reduction with the same key concepts translates into a simplified assembling of those parts. This is the reason why the description of the gadgets is so extensive.

However the paper misses to explicitly state that their Peg Duotaire versions are played on rectangular boards. In practice boards in hexagonal or triangular shapes with different numbers of positions adjacent to a peg, do exist [3, 4].

This paper solves the question whether these games are PSPACE-complete and helps understanding the complexity of such games, analyzed repeatedly in the scientific literature. We may in turn use the new complexity classification to prove the PSPACE-completeness of any problem that we can reduce Single- or Multi-Hop Peg Duotaire to.

References

[1] Davide Bilò, Luciano Gualà, Stefano Leucci, Guido Proietti, and Mirko Rossi. On the PSPACE-completeness of Peg Duotaire and other Peg-Jumping Games. In Hiro Ito, Stefano Leonardi, Linda Pagli, and Giuseppe Prencipe, editors, *9th International Conference on Fun with Algorithms (FUN 2018)*, volume 100 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[2] Ian Miguel S. Armagan Tarim Christopher Jefferson, Angela Miguel. Modelling and solving english peg solitaire. *Computers & Operations Research*, 33:2935–2959, 2006.

[3] George I. Bell. Peg solitaire on hexagonal boards. `http://www.gibell.net/pegsolitaire/Hexagon/index.html`, 2016. Accessed: 2018-10-20.

[4] George Bell. A fresh look at peg solitaire. 80:16–28, 02 2007.

5 Appendix

Name	#p[5]	pegs	goal	known proofs
Peg Solitaire	1	same color	clear board of pegs	NP-complete [23 as referenced in [1]] and PSOL[6]
Single-Hop Peg Duotaire	2	same color[7]	opponent left without move	force win is PSPACE-complete (this paper)[8]
Multi-Hop Peg Duotaire	2	same color[7]	opponent left without move	force win is PSPACE-complete (this paper)
Solitaire-Reachability	1	same color	reach target from current position	NP-complete [15 as referenced in [1]]
Solitaire-Army	1	same color	find initial configuration that fills target in desert	distance 5 cannot be reached on any finite board (Conway [18 as referenced in [1]])
2-player version of Solitaire-Reachability	2	2 colors	reach target, opponents pegs are obstacles	force win is PSPACE-complete (this paper)
Konane (ancient Hawaiian game)	2	2 colors	last move looses (jump only over opponents pegs)[9]	PSPACE-complete [16 as referenced in [1]]

Table 1: Peg-Jumping Games from [1]

[5]=number of players

[6]polynomial-time solvable for rectangular board of fixed size (solvable instances form a regular language) [21,22 as referenced in [1]]

[7]impartial game: possible moves and payoffs are symmetric between players

[8]instances for which the first player wins cannot be described by a context-free language

[9]can have multiple jumps as long as it's in a straight line