Maastricht University

Games, Puzzles and Treewidth

Citation for published version (APA):

Van Der Zanden, T. C. (2020). Games, Puzzles and Treewidth. In F. Fomin, S. Kratsch, & E. van Leeuwen (Eds.), *Treewidth, Kernels, and Algorithms* (pp. 247-261). Springer, Cham. https://doi.org/10.1007/978-3-030-42071-0_17

Document status and date:
Published: 20/04/2020

DOI:
10.1007/978-3-030-42071-0_17

Document Version:
Publisher's PDF, also known as Version of record

Document license:
Taverne

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.umlib.nl/taverne-license

Take down policy
If you believe that this document breaches copyright please contact us at:

repository@maastrichtuniversity.nl

providing details and we will investigate your claim.

Download date: 02 Oct. 2023

Games, Puzzles and Treewidth

Tom C. van der Zanden(✉)

Department of Data Analytics and Digitalisation, Maastricht University, Maastricht, The Netherlands
T.vanderZanden@maastrichtuniversity.nl

Abstract. We discuss some results on the complexity of games and puzzles. In particular, we focus on the relationship between bounded treewidth and the (in-)tractability of games and puzzles in which graphs play a role. We discuss some general methods which are good starting points for finding complexity proofs for games and puzzles.

Keywords: Complexity · Games · Puzzles · Treewidth

1 Introduction

This article was written on the occasion of the 60th birthday of Hans Bodlaender. As one of his PhD students, I have come to know Hans not only for his appreciation of deep theoretical research on graphs, but also for his enjoyment of games and puzzles and for fun (but nevertheless serious) research. Our collaboration started during Hans' algorithms master course, in which there was an assignment on solving the puzzle game Bloxorz. This resulted in our first joint paper, in which we showed Bloxorz PSPACE-complete [24]. Happy 60th, Hans!

The complexity of games and puzzles is a widely studied topic, and there are far too many results out there to even begin to give an overview. Instead, we will focus on a few common techniques for showing hardness and in particular on the relation to treewidth. One of the reasons game and puzzle complexity is so popular, is, of course, that it is fun: trying to build gadgets with elements from Super Mario or arguing whether a game remains NP-hard even when general relativity is considered [9] provide light-hearted insights into complexity. Games and puzzles also make excellent and visceral examples for teaching: Sudoku is a perfect example of a problem in NP – it is clearly easy to verify the correctness of a solution while, intuitively, it seems much harder to find a solution (and indeed, the problem is NP-complete [27]).

1								6
		6		2		7		
7	8	9	4	5		1		3
			8		7			4
				3				
	9				4	2		1
3	1	2	9	7			4	
	4			1	2		7	8
9		8						

Fig. 1. An instance of a Sudoku puzzle, a common example of an NP-complete puzzle.

Decision versions of many classical pen-and-paper puzzles (such as Sudoku, Fig. 1) are easily seen to be in NP and most often, are also NP-complete. Other examples include Nonogram and Kakuro (see e.g. [13], Appendix A.7 for

© Springer Nature Switzerland AG 2020
F. V. Fomin et al. (Eds.): Bodlaender Festschrift, LNCS 12160, pp. 247–261, 2020.
https://doi.org/10.1007/978-3-030-42071-0_17

an overview). These puzzles are essentially constraint satisfaction problems, so it is natural they would be NP-complete.

In many puzzles, the player has to move pieces around on a board in order to reach a target configuration. For example, in Peg Solitaire, pegs are arranged on a grid, and the player is allowed to move a peg from one hole to another by jumping over another (adjacent) peg, after which the peg that has been jumped over is removed from the board. Deciding whether the board can be cleared (save for one peg) is NP-complete [22]. Here, membership in NP is obvious: since every move reduces the number of pegs by one, a solution has bounded length.

Fig. 2. A Rush Hour puzzle by ThinkFun. The goal is to move the red car off the right side of the board.

More often, puzzles which involve moving pieces on a board are PSPACE-complete. A necessary[1] condition for this is that the length of a solution is not polynomially bounded (and, save for a few exceptions, if the length of a solution is not polynomially bounded, the problem is PSPACE-complete). An example of such a puzzle is Rush Hour [10] (Fig. 2), in which cars (rectangles of size 1×2), that may only move backwards and forwards, are arranged vertically and horizontally on a board and the goal is to move a specific car to its destination. One possible approach to proving hardness [13] constructs a Rush Hour instance in which the player is forced to move the cars in such a way that corresponds to painstakingly checking all possible assignments to a quantified boolean formula[2]. These types of puzzles are essentially reconfiguration problems [17].

Another interesting source of problems to analyse are (platform) video games. Here, the decision question is whether the player can reach the end of the level. The simplest such games are in P (since they reduce to a reachability question that can be solved using, for instance, breadth-first search). However, depending on what elements are present in a level, such games may be NP-complete (e.g., Super Mario [1]) or even PSPACE-complete (e.g., Zelda [2]).

Of course, the complexity strongly depends on what features are present in the game. Several metatheorems exist characterizing which features make a video game hard (see e.g., [11,25]). For example, NP-hardness can be obtained if there is a time limit and a way to force the player to visit several locations (reduction from Hamiltonian Path), or if the game features one-way passages (such as a *long fall* feature, where the player can survive a fall higher than they can jump) and a way to enable passages at a later stage of the level (such as a button that opens a door or an enemy that can be killed to enable safe passage later). For PSPACE-hardness, reversible state changes are required, such as blocks that can be pushed back and forth (e.g., Sokoban [7]) or pressure plates that can *force* the player to close doors (and a means to open them again).

[1] Unless NP = PSPACE.

[2] E.g., a formula of the form $\exists_{x_1} \forall_{x_2} \exists_{x_3} \cdots \forall_{x_n} \phi(x_1, x_2, \ldots, x_n)$, where ϕ is an unquantified boolean formula over binary variables $x_1, \ldots, x_n$.

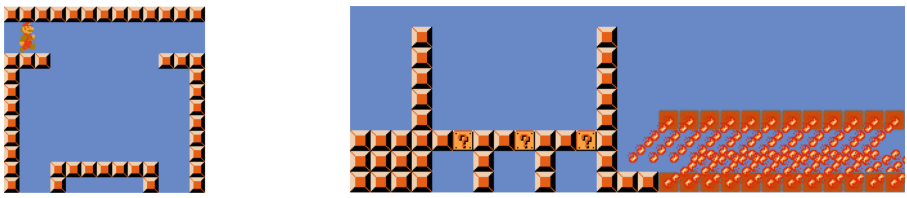
Finally, two-player games tend to be either PSPACE- or EXPTIME-complete. Games which allow an unbounded number of moves tend to be EXPTIME-complete, such as Chess [12] and Go [20]. On the other hand, if the number of moves is bounded, two-player games tend to be PSPACE-complete. Examples include Reversi [18] and Generalized Geography [21].

In this article, we will survey several known results on the hardness of games and puzzles, with a focus on techniques and frameworks for proving hardness and, where applicable, the relation to (bounded) treewidth.

2 Hardness of Video Games

2.1 NP-Hard Video Games

One possible method to show NP-hardness of a video game is to give a direct reduction from Satisfiability. Aloupis et al. [1] use this approach for Super Mario. Variable and clause gadgets are shown in Fig. 3. In the variable gadget, the player may go either left or right to assign a particular variable to true or false. The fall is long enough that the player cannot jump back up to (also) make the other assignment. The clause gadget is entered from one of the three entrances on the bottom (corresponding to one of the three literals in a 3-SAT clause); hitting the item block from below will release a star powerup to the area above. This powerup can later be used to traverse the flames on the right.



(a) Variable gadget (b) Clause gadget

Fig. 3. Gadgets showing the hardness of Super Mario due to Aloupis et al. [1].

To complete the reduction, we still need a (complicated) *crossover* gadget, i.e., a gadget which allows to paths to cross each other without the player being able to switch from one path to the other. This gadget is needed to make appropriate connections between the variable and clause gadgets. Aloupis et al. [1] give such a gadget. The construction allows (and forces) the player to first traverse all variable gadgets and make a choice for each one (and thus, unlocking the various clause gadgets) and then traverse the check paths of all the clause gadgets.

To simplify the proof, one might consider using Planar 3-SAT, that is, 3-SAT wherein the incidence graph of variables and clauses is planar – hoping that using this problem might help eliminate the need for a crossover gadget. Unfortunately, this is not the case, because we also need additional paths to visit all the variables (to set them) and clauses (to check them).

Recently [19], the following very interesting satisfiability variant was shown NP-complete:

Sided Linked Planar 3-SAT-3
Given: A CNF formula ϕ with at most 3 literals per clause and at most 3 literals per variable, such that the incidence graph, augmented with a cycle that first visits all variables and then visits all clauses, is planar *and* admits an embedding such that for each variable, the edges going to its positive literals occur on a different side of the cycle than the edges going to its negative literals.
Question: Does ϕ have a satisfying assignment?

This is a very useful satisfiability variant, since the cycle visiting all variables and clauses can be used to perform the setting and checking exactly as required in the previous proof. Using this satisfiability variant, the crossover gadget can be eliminated from the reduction for Super Mario, and, in fact, from many other proofs as well.

This proof illustrates just one example of what can cause a video game to be NP-hard: in this case, it is the fact that the existence of a long drop (which forces us to make a choice in going left or right) combined with the existence of a game element that unlocks a path for later traversal (essentially, this is Metatheorem 3 of [11]). Viglietta's metatheorems [25] capture several other possible elements that can make a game NP-hard:

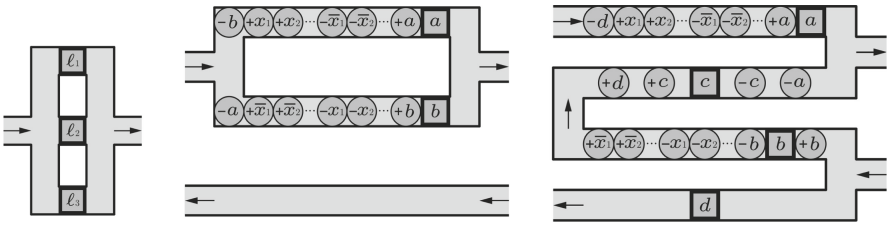
Metatheorem 1 ([25]). *If a video game features one of the following combinations of elements, it is* NP*-hard:*

- *Location traversal (spots in the level that must be traversed, e.g., items that must be collected), combined with single-use paths.*
- *Tokens, toll roads that consume a token to traverse and location traversal. Tokens may be either cumulative (any number can be held) or collectible (one may be held at a time).*
- *Cumulative tokens and toll roads.*
- *Pressure plates*[3]*, which correspond one-to-one to doors that they may open or close.*

2.2 PSPACE-Hard Video Games

Many video games are PSPACE-hard. This can be the case when they feature movable elements (such as Sokoban [7]) or if the game features elements that

[3] A pressure plate, as opposed to a button, is a game element that the player cannot avoid triggering if traversed.



(a) Clause gadget (b) Existential quantifier (c) Universal quantifier

Fig. 4. Construction showing PSPACE-hardness of games with pressure plates (depicted as circles; a circle labelled $+x$ (resp. $-x$) opens (resp. closes) door x) and doors (depicted as squares) due to Viglietta [25].

give the player the option to open doors or can force them to close them again. In the previous section, we stated the metatheorem that for pressure plates which correspond one-to-one with doors it is possible to obtain NP-hardness. This is tight, since there exist NP-complete games featuring these elements: since each door is controlled by at most one pressure plate, once it is opened or closed its state can never change again.

If instead we consider a game in which each door may be controlled by two pressure plates, one which may open it and one which may close it, we obtain PSPACE-hardness:

Metatheorem 2 [25]**.** *If a game features pressure plates and doors, each pressure plate controls at most one door and each door is controlled by at most two pressure plates, then the game is* PSPACE*-hard.*

The proof (due to Viglietta [25]) is by reduction from Quantified Boolean Formula Satisfiability. Figure 4 shows the gadgets used in the construction: (a) a clause gadget, which may be traversed if at least one of the doors corresponding to one of the three literals of the clause is open. (b) an existential quantifier gadget, which can be traversed from top left to top right by picking either the top or bottom path (which opens doors corresponding to either a true or false assignment to that variable) (c) a universal quantifier, which must be traversed in the following way: the player enters from the top left, passes the top path (making a true assignment), then leaves from the top right. The player then returns (after checking the rest of the formula) on the bottom right and then must traverse the middle two paths (making a false assignment), after which the player must again exit from the top right to verify the assignment again, before returning on the bottom right and being able to exit on the bottom left.

One drawback of this scheme is that pressure plates must be able to act on doors anywhere in the level. This cannot always be realized easily. On the positive side, it is not necessary to build a crossover gadget (since crossings are provided by the pressure plates working on arbitrarily distant doors). Another framework, the *door framework* [2], gets rid of the requirement that buttons may

operate arbitrarily distant doors, and states that a game is PSPACE-hard if it is possible to build a crossover gadget and a door gadget, which contains three distinct paths: an open path (which may open the door when traversed), a close path (which forcibly closes the door when traversed) and a door path (which may be traversed only when open).

Many games feature *buttons*, a game element which is similar to, but different from, a pressure plate. When encountering a button, the player has the option of pressing it or not (in contrast to a pressure plate, which is activated whether the player would like it to or not). Buttons which act on only one door at a time are trivial (since a player would always press an "open" button and would never press a "close" button), so it is not possible to obtain a hardness metatheorem for these. Instead, Viglietta [25] considers k-buttons: a button which may act on at most k doors at once. For $k \geq 2$, a player may be incentivized to press a button that would close a door, provided it also opens another.

One could ask what is the minimum k for which a game with k-buttons is hard. Viglietta [25] showed that for $k = 2$ such games are NP-hard, and for $k = 3$, PSPACE-hard. Hans and I improved this, showing that $k = 2$ already suffices for PSPACE-hardness:

Metatheorem 3 ([24]). *A game featuring 2-buttons is* PSPACE*-hard, even if each door may be acted upon by at most 2 buttons.*

Bloxorz [16] (Fig. 5) is a puzzle game that features a third type of element, a *switch*, that may toggle a (trap-)door between open and closed (i.e., repeated presses of a switch will cause the state of the door to cycle between open and closed). The unique feature of Bloxorz is that the player is a $1 \times 1 \times 2$ block, for which two types of moves are possible: if the block is standing up, a tilting move is possible, which causes the block to lay on one of its 1×2 sides. If the block is lying on a 1×2 side, it can either do a tilting move (causing the block to stand up again), or a rolling move, rolling over to another of its 1×2 sides. These special types of moves enable some interesting gadget constructions.

Each switch in Bloxorz may act on only one trapdoor (and each trapdoor may be acted upon by only one switch). However, we can exploit the fact that the block is 1×2 to build what are, essentially, 2-switches (if the block falls down on a 1×2 side on two switches, they are both triggered). Thus, it is possible to

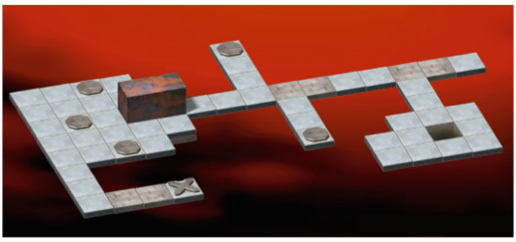


Fig. 5. An example Bloxorz level

show that Bloxorz is PSPACE-complete [24]. The proof can be distilled to the following metatheorem:

Metatheorem 4 (Consequence of [24]). *If a game features 2-switches, it is* PSPACE*-hard, even if each door may be controlled by at most one switch.*

Note that for this metatheorem, it is not relevant whether the 2-switches behave like pressure plates (in the sense that they are always forcibly triggered) or buttons (in the sense that the player can choose to trigger them or not).

3 Constraint Logic Framework

Many hardness constructions for games and puzzles are quite involved, and require the creation of complicated crossover gadgets or lengthy arguments about simulating the behaviour of a Turing machine. The Constraint Logic framework, introduced by Hearn and Demaine [13], provides several games and puzzles based around the notion of *constraint graphs*, each of which is complete for a different complexity class. These aim to be convenient starting points for reductions and simplify hardness proofs (for instance, by eliminating the need to construct a dedicated crossover gadget).

Of particular interest is the PSPACE-complete variant, called Nondeterministic Constraint Logic (NCL), which has proven very useful for a wide variety of hardness proofs. However, the Constraint Logic framework includes many games, each of which capture the essence of a different complexity class and its relation to games: for instance, it is possible to define a two-player game on constraint graphs. The edges are partitioned between the players, the players take turns flipping an edge from their set and each player has a target edge that they must flip to win. In a bounded setting (each edge may be flipped at most once), this game is PSPACE-complete (capturing, e.g., the hardness of Reversi), while in general it is EXPTIME-complete (capturing, e.g., the hardness of Chess). We will now formally introduce Constraint Logic.

Definition 1. *A* constraint graph *is an undirected graph $G = (V, E)$, together with:*

- *For each vertex $v \in V$, a vertex weight $w(v)$,*
- *For each edge $e \in E$, an edge weight $w(e)$.*

Given a constraint graph G, a *configuration* is an assignment of orientations to its edges. A configuration is *valid* if for each vertex, the total weight of edges pointing in towards that vertex is at least that vertex' weight.

The Constraint Graph Satisfiability problem asks whether a constraint graph G admits a valid configuration. It can be viewed as the constraint logic equivalent of 3-SAT.

Constraint Graph Satisfiability
Given: A constraint graph G.
Question: Does G have a valid configuration?

Constraint Graph Satisfiability is NP-complete, even for very restricted constraint graphs:

Theorem 1 ([13]). *It is* NP*-complete to decide whether a constraint graph G admits a valid configuration, even if G is planar, has maximum degree* 3*, each vertex has weight equal to* 2 *and the edge weights are in* $\{1, 2\}$*.*

In fact, for all discussions on constraint logic, it suffices to consider vertex weights 2 and edge weights $\{1, 2\}$. Going forward, we shall omit the vertex weights. Following the convention in [13], red edges shall have weight 1 and blue edges (drawn thicker) shall have weight 2.

Given a (valid) configuration for a constraint graph, we can obtain another configuration by flipping the direction of one edge. A *move* is an edge flip between two valid configurations.

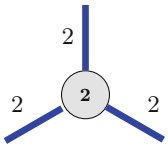
Nondeterministic Constraint Logic (NCL)
Given: A constraint graph G, a valid configuration C for G and a target edge $e \in G$.
Question: Is there a sequence of moves on G, starting from C, that eventually reverses e?

We may also consider the configuration-to-configuration variant of NCL, which asks whether a starting configuration C_1 can be reconfigured (through moves) to a target configuration C_2. All complexity results discussed in this section hold analogously for this variant.

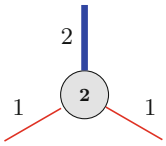
NCL is PSPACE-complete, even for very restricted constraint graphs. We may consider graphs constructed using only the two vertex types shown in Fig. 6. The *OR vertex* has three incident blue (weight 2) edges, and thus in any valid configuration at least one of them needs to point inward. It resembles an OR gate in the sense that if we identify one edge as the "output", it can point outward if and only if at least one of the two other edges is pointing inward. The *AND vertex* has two incident red (weight 1) edges and one incident blue (weight 2) edge. It is satisfied if and only if both red edges point inward or the blue edge points inward. Thus, we can think of the blue edge as its "output", able to point outward only if both red edges are pointing inward. The fact that NCL is PSPACE-complete for such restricted graphs makes it a very powerful tool for constructing hardness proofs.

Theorem 2. Nondeterministic Constraint Logic *is* PSPACE*-complete, even for planar constraint graphs that use only AND and OR vertices.*

To prove hardness by reduction from NCL, we thus only need to show how AND and OR vertices may be constructed. Figure 7 shows how the AND and OR vertices may be constructed in Rush Hour. In the AND vertex, car C can move down if and only if cars A and B move out; in the OR vertex, if either car A or B moves out, car C can move in. These constructions are essentially the only elements necessary for the proof, it only remains to be shown that they

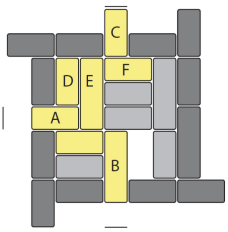


(a) OR vertex

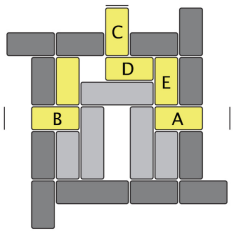


(b) AND vertex

Fig. 6. The two vertex types from which a restricted constraint graph is constructed: (a) OR vertex and (b) AND vertex. Following the convention set in [13], as a mnemonic weight 2 edges are drawn blue (dark grey) and thick, while weight 1 edges are drawn red (light grey) and thinner. From [23]. (Color figure online)



(a) AND vertex



(b) OR vertex

Fig. 7. AND and OR vertices, as constructed in a reduction to Rush Hour, Hearn and Demaine [13].

can (given a planar constraint graph) be arranged in the plane and connected accordingly; this greatly simplifies the original PSPACE-completeness proof [10].

Bounded Width. Of course, given that a graph is involved, a natural question is what happens if the unweighed graph underlying the constraint graph has bounded treewidth. It can easily be seen that Constraint Graph Satisfiability is polynomial-time solvable on graphs of bounded treewidth. Surprisingly, NCL is PSPACE-complete, even for graphs of bounded treewidth (and actually, even for graphs of bounded bandwidth):

Theorem 3 ([23]). *There is a constant c such that* Nondeterministic Constraint Logic *is* PSPACE*-complete, even on planar constraint graphs of bandwidth at most c that use only AND and OR vertices.*

This result is closely related to a result of Wrochna [26] on reconfiguration problems in bounded bandwidth graphs. Wrochna shows that a Turing machine (with polynomially bounded tape) can be simulated in a bounded width structure. Since the tape is linear and, locally, we only need to take into account a bounded number of cases (depending on the number of states of the TM and the size of the alphabet), it is actually quite natural that this should be the case.

An amusing consequence of this result is that Rush Hour is PSPACE-complete even when played on a board of constant width.

Interestingly, some games on graphs that are PSPACE-complete do become tractable if the graph has bounded treewidth. For instance, Generalized Geography can be solved in linear time on graphs of bounded treewidth [3]. The main difference is that Generalized Geography is a bounded two-player game, whereas Nondeterministic Constraint Logic is single player, unbounded. The fact that moves are reversible means that information can flow back and forth in the graph, passing through separators. Essentially, the separator property of bounded treewidth graphs is useless if there is a mechanism that can pass information through them. In contrast, in Generalized Geography, once a vertex has been picked by one of the players, it cannot be used again. It is an interesting open problem to settle the conjecture that Bounded Two-Player Constraint Logic is also tractable in bounded treewidth graphs, and that the unbounded variant remains EXPTIME-complete.

For more information on Constraint Logic, and for many interesting reductions from Constraint Logic to various games and puzzles, I refer to the excellent book [13] by Hearn and Demaine.

4 Partition and Packing Problems

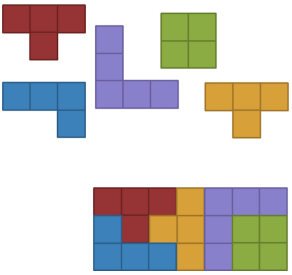


Fig. 8. A simple Polyomino Packing puzzle and a possible solution.

So far, we have looked at video games, pen-and-paper puzzles and sliding piece puzzles. A final class of puzzles that we are going to look at are packing puzzles and jigsaws. Besides fun applications (puzzles such as Tangram, polyomino packing or jigsaws), packing problems also have many practical applications of real-world importance, such as loading packages into a delivery truck or loading products onto pallets [14].

A key tool for showing the hardness of packing problems is the 3-Partition problem:

3-Partition
Given: A collection A of $3n$ positive integers $a_1, a_2, \ldots, a_{3n}$.
Question: Is there a partition of A into n triples such that the numbers in each triple sum to $\frac{1}{n}\Sigma_{i=1}^{3n} a_i$?

The 3-Partition problem is *strongly* NP-hard, i.e., it remains NP-complete even if each integer a_i is bounded by a polynomial in n. This is very useful, since in a reduction we can construct puzzles whose size is proportional to the integers in the input and still obtain a hardness result.

Reductions from 3-Partition can sometimes give very easy hardness proofs. Demaine and Demaine [8] give an excellent overview of many types of packing and jigsaw puzzles, most of which are shown hard using 3-Partition.

An interesting type of puzzles are Polyomino Packing puzzles. In Polyomino Packing we are given a collection of polyominoes (shapes consisting of unit squares, glued together on their edges) and are asked to build a target shape (which is a subset of a square grid). Figure 8 shows an example of a Polyomino Packing puzzle, in which we are asked to pack 5 polyominoes into a 3×7 rectangle.

Even for very restricted instances, Polyomino Packing is already NP-complete:

Theorem 4 ([8]). *It is (strongly)* NP*-complete to decide whether n given rectangular pieces sized $1 \times x_1, 1 \times x_2, \ldots, 1 \times x_n$, where the x_i's are positive integers bounded above by a polynomial in n, can be exactly packed into a specified rectangular box of area $x_1 + x_2 + \ldots + x_n$.*

The proof due to Demaine and Demaine [8] is by reduction from 3-Partition and is quite simple and elegant: given an instance of 3-Partition with $3n$ integers $a_1, a_2, \ldots, a_{3n}$, each integer a_i is translated to a rectangle of size $1 \times (a_i + n)$. We then ask whether these rectangles can be packed into a $n \times \left(3n + \frac{1}{n}\Sigma_{i=1}^{3n} a_i\right)$ rectangle.

Bounded Width. An interesting question is what happens when we bound the *width* in some way. For instance, we might ask ourselves what the complexity of Polyomino Packing is when the target shape is a $k \times n$ rectangle, where k is a (bounded) width parameter. There is a very simple hardness proof showing hardness even for $k = 2$:

Theorem 5. Polyomino Packing *is* NP*-hard, even when the target shape is a $2 \times n$ rectangle.*

This can be seen by reduction from 3-Partition. We may construct a $2 \times \left(n + 1 + \Sigma_{i=1}^{3n} a_i\right)$ rectangle into which n slots of size $1 \times \left(\frac{1}{n}\Sigma_{i=1}^{3n} a_i\right)$ are cut. We then ask whether this "rectangle with slots" can be packed together with $3n$ rectangle polyominoes of sizes $1 \times a_1, 1 \times a_2, \ldots, 1 \times a_{3n}$ into a box of size $2 \times \left(n + 1 + \Sigma_{i=1}^{3n} a_i\right)$.

The reason having bounded width does not help in this case is that, if we try to do separator-based dynamic programming (where a separator might divide the $2 \times n$ box into two $2 \times n/2$ boxes), we need to remember which pieces we have used on one side of our separator. A very similar phenomenon appears in Subgraph Isomorphism in planar and bounded treewidth graphs, where having bounded treewidth also does not really help [4].

Exact Complexity. Besides studying classification into complexity classes such as NP and P, it is also interesting to ask what the exact complexity of a problem is. That is, since Polyomino Packing is NP-hard, we know it most likely cannot be solved in polynomial time. It then becomes interesting to ask what the slowest-growing f is such that Polyomino Packing into a $2 \times n$ box can be solved in time $2^{O(f(n))}$.

Using a dynamic programming approach, and the observation that placing polyomino pieces into a $2 \times n$ box partitions the remaining area into connected components that are easy to characterize, it is possible to obtain the following result:

Theorem 6 ([6]). Polyomino Packing *can be solved in* $2^{O(n^{3/4} \log n)}$ *time if the target shape is a* $2 \times n$ *rectangle.*

Of course, a natural question is whether we can also obtain a (matching) lower bound on f. Using the Exponential Time Hypothesis [15], doing so is sometimes possible. The Exponential Time Hypothesis (ETH) states that there exists no algorithm solving n-variable 3-SAT in $2^{o(n)}$ time. Assuming this hypothesis, and by designing efficient reductions (that do not blow up the instance size too much), it is possible to derive conditional lower bounds on the running time of an algorithm.

The blow-up of the reduction from 3-SAT to 3-Partition is rather large and does not lead to a tight lower bound for the $2 \times n$ case. However, many packing problems exhibit an interesting behaviour: their optimal running time is $2^{\Theta(n/\log n)}$ (under the ETH). This holds for, for instance, Subgraph Isomorphism on planar graphs [4,5] and also for Polyomino Packing if we increase k to 3:

Theorem 7 ([6]). Polyomino Packing *cannot be solved in* $2^{o(n/\log n)}$ *time, even if the target shape is a* $3 \times n$ *rectangle.*

Using treewidth-based techniques, we can obtain a matching $2^{O(n/\log n)}$-time algorithm [6]. Interestingly, this means that $4 \times n$ (or $n \times n$) Polyomino Packing is essentially not any harder than $3 \times n$ Polyomino Packing. $2 \times n$ Polyomino Packing is somewhat easier, but still NP-hard.

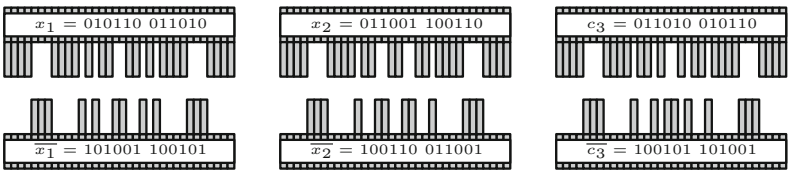


Fig. 9. Top: polyominoes corresponding to variables x_1, x_2 and clause c_3. Bottom: the complementary polyominoes, that mate with the polyominoes above them to form a $3 \times \Theta(\log n)$ rectangle. Note that the polyominoes are depicted compressed horizontally. Due to [6].

The lower bound can be obtained by a direct reduction from 3-SAT. If we number the variables in the instance $1, 2, \ldots, n$ and the clauses $n+1, \ldots, n+m+1$, we can pick a unique bitstring corresponding to each variable and clause, being derived from the binary representation of its index. We can then use these bitstrings to construct *corresponding polyominoes*, which consist of a solid row on top and a row on the bottom which has a square whenever the bitstring has a

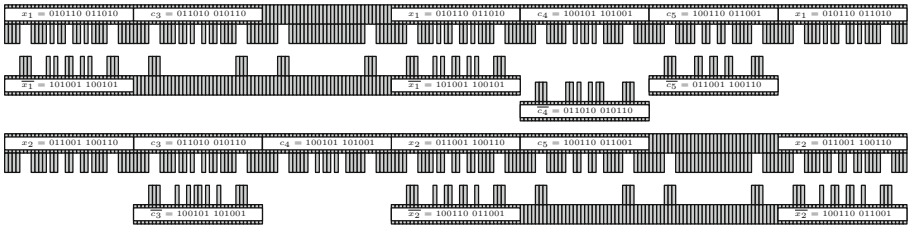


Fig. 10. Example of the reduction for the formula $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$. Top-to-bottom, left-to-right: formula encoding polyomino for x_1, variable-setting polyomino for x_1, clause-checking polyomino for c_4, clause checking-polyomino for c_5, formula-encoding polyomino for x_2, clause-checking polyomino for c_3, variable-setting polyomino for x_2. The polyominoes are arranged in a way that suggests the solution $x_1 = \mathit{false}$, $x_2 = \mathit{true}$. Due to [6].

1 in that position. Since we need only $\log(n+m)$ bits to represent each number, each corresponding polyomino has only $O(\log n)$ squares.

For each clause and variable we can also define a complementary polyomino, which mates with the corresponding polyomino to form a $3 \times \Theta(\log n)$ rectangle. Figure 9 shows an example of corresponding and complementary polyominoes.

Using corresponding and complementary polyominoes as building blocks, it is possible to build up larger polyominoes that form an instance of $3 \times n$ Polyomino Packing that is solvable if and only if the formula is satisfiable. The instance will have a size of $O(n \log n)$ squares, leading to the claimed lower bound.

We will not go into the full details of the reduction here; instead, we refer to [6]. However, we sketch some details. For every variable, a formula-encoding polyomino is built, together with a variable-setting polyomino. In any solution, there are two possible placements of the variable-setting polyomino with respect to the corresponding formula-encoding one. Either overlapping the left side of the formula-encoding polyomino, corresponding to a false assignment, or overlapping the right, corresponding to a true assignment. The places where the variable-setting polyomino does not overlap the formula-encoding polyomino contain gaps into which polyominoes corresponding to clauses satisfied by the assignment can fit. Figure 10 shows an example of this construction.

This technique offers an alternative to 3-Partition-based proofs, giving tighter lower bounds than would be obtained through 3-Partition. Many games and puzzles are expressive enough to allow bitstrings to be used to succinctly encode variables and clauses. The example of $2 \times n$ Polyomino Packing is an example where we do not have sufficient expressivity and have to fall back to a direct reduction from 3-Partition. An interesting open problem is to settle the exact complexity of 3-Partition (and that of $2 \times n$ Polyomino Packing), as current upper and lower bounds are not tight (the chain of reductions that establishes the hardness of 3-Partition is quite long, blowing up the size of the instance at several steps; however, we do not know the precise value of the lower bound that follows).

5 Conclusions

In this survey, we have discussed several complexity results on various (types of) games and puzzles. We have seen examples of pen-and-paper puzzles, video games, jigsaws and packing puzzles, some two-player games and reconfiguration puzzles. In several cases, we have seen surprising connections to treewidth: reconfiguration puzzles (and problems) remain PSPACE-hard on bounded (tree-)width structures, while other PSPACE-hard games (such as Generalized Geography) become tractable. We have seen that for Polyomino Packing, the problem similarly remains NP-hard even for boards of width 2 and that the hardness does not increase above width 3.

It would be an interesting open problem to further study the complexity of Constraint Logic variants and games under bounded width. The case for Nondeterministic Constraint Logic is well understood, but it would be interesting to see if the tractability result for Generalized Geography generalizes to a result for Bounded Two-Player Constraint Logic. Going further, it would be interesting to investigate the complexity of other Constraint Logic variants in bounded width graphs.

References

1. Aloupis, G., Demaine, E.D., Guo, A., Viglietta, G.: Classic nintendo games are (NP-)hard. arXiv preprint arXiv:1203.1895 (2012)
2. Aloupis, G., Demaine, E.D., Guo, A., Viglietta, G.: Classic nintendo games are (computationally) hard. Theoret. Comput. Sci. **586**, 135–160 (2015)
3. Bodlaender, H.L.: Complexity of path-forming games. Theoret. Comput. Sci. **110**(1), 215–245 (1993)
4. Bodlaender, H.L., Nederlof, J., van der Zanden, T.C.: Subexponential time algorithms for embedding H-minor free graphs. In: Chatzigiannakis, I., Mitzenmacher, M., Rabani, Y., Sangiorgi, D. (eds.) 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, vol. 55, pp. 9:1–9:14 (2016). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik
5. Bodlaender, H.L., van der Zanden, T.C.: Improved lower bounds for graph embedding problems. In: Fotakis, D., Pagourtzis, A., Paschos, V.T. (eds.) CIAC 2017. LNCS, vol. 10236, pp. 92–103. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57586-5_9
6. Bodlaender, H.L., van der Zanden, T.C.: On the exact complexity of polyomino packing. In: Ito, H., Leonardi, S., Pagli, L., Prencipe, G. (eds.) 9th International Conference on Fun with Algorithms (FUN 2018), Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, vol. 100, pp. 9:1–9:10 (2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik
7. Culberson, J.C.: Sokoban is PSPACE-complete. In: Lodi, E., Pagli, L., Santoro, N. (eds.) International Conference on Fun with Algorithms (FUN 1998), pp. 65–76. Carleton Scientific, Waterloo (1998)
8. Demaine, E.D., Demaine, M.L.: Jigsaw puzzles, edge matching, and polyomino packing: connections and complexity. Graph. Comb. **23**(1), 195–208 (2007)

9. Demaine, E.D., Lockhart, J., Lynch, J.: The computational complexity of portal and other 3D video games. In: Ito, H., Leonardi, S., Pagli, L., Prencipe, G. (eds.) 9th International Conference on Fun with Algorithms (FUN 2018), Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, vol. 100, pp. 19:1–19:22 (2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik
10. Flake, G.W., Baum, E.B.: Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants". Theoret. Comput. Sci. **270**(1–2), 895–911 (2002)
11. Forišek, M.: Computational complexity of two-dimensional platform games. In: Boldi, P., Gargano, L. (eds.) FUN 2010. LNCS, vol. 6099, pp. 214–227. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13122-6_22
12. Fraenkel, A.S., Lichtenstein, D.: Computing a perfect strategy for n × n chess requires time exponential in n. J. Comb. Theory, Ser. A **31**(2), 199–214 (1981)
13. Hearn, R.A., Demaine, E.D.: Games, Puzzles, and Computation. AK Peters/CRC Press, Natick (2009)
14. Hodgson, T.J.: A combined approach to the pallet loading problem. AIIE Trans. **14**(3), 175–182 (1982)
15. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**, 512–530 (2001)
16. Clarke, D.: DX Interactive. Bloxorz. https://damienclarke.me/#bloxorz
17. Ito, T., et al.: On the complexity of reconfiguration problems. Theoret. Comput. Sci. **412**(12), 1054–1065 (2011)
18. Iwata, S., Kasai, T.: The othello game on an n × n board is PSPACE-complete. Theoret. Comput. Sci. **123**(2), 329–340 (1994)
19. Pilz, A.: Planar 3-SAT with a clause/variable cycle. arXiv preprint arXiv:1710.07476 (2017)
20. Robson, J.M.: The complexity of Go. In: 9th World Computer Congress on Information Processing, pp. 413–417 (1983)
21. Schaefer, T.J.: On the complexity of some two-person perfect-information games. J. Comput. Syst. Sci. **16**(2), 185–225 (1978)
22. Uehara, R., Iwata, S.: Generalized Hi-Q is NP-complete. IEICE Trans. (1976-1990) **73**(2), 270–273 (1990)
23. van der Zanden, T.C.: Parameterized complexity of graph constraint logic. In: Husfeldt, T., Kanj, I. (eds.) 10th International Symposium on Parameterized and Exact Computation (IPEC 2015), Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, vol. 43, pp. 282–293 (2015). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik
24. van der Zanden, T.C., Bodlaender, H.L.: PSPACE-completeness of bloxorz and of games with 2-buttons. In: Paschos, V.T., Widmayer, P. (eds.) CIAC 2015. LNCS, vol. 9079, pp. 403–415. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18173-8_30
25. Viglietta, G.: Gaming is a hard job, but someone has to do it!. Theory Comput. Syst. **54**(4), 595–621 (2014)
26. Wrochna, M.: Reconfiguration in bounded bandwidth and tree-depth. J. Comput. Syst. Sci. **93**, 1–10 (2018)
27. Yato, T., Seta, T.: Complexity and completeness of finding another solution and its application to puzzles. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **86**(5), 1052–1060 (2003)