博士论文

On the Computational Complexity of Portal and Push-Pull Block Puzzles

by

Jayson Lynch

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2015

Author .....
Department of Electrical Engineering and Computer Science
September, 2015

Certified by .....
Erik Demaine
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .....
Prof. Albert R. Meyer
Chairman, Masters of Engineering Theses Committee

On the Computational Complexity of Portal

and Push-Pull Block Puzzles

by

Jayson Lynch

Submitted to the Department of Electrical Engineering and Computer Science
on September, 2015, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

We classify the computational complexity of two types of motion planning problems represented in games. Portal, a popular video game, is shown to be NP-hard or PSPACE-complete depending on the game mechanics allowed. Push-pull block puzzles are games, similar to Sokoban, which involve moving a 'robot' on a square grid with obstacles and blocks that can be pushed or pulled by the robot into adjacent squares. We prove that push-pull block puzzles in 3D and push-pull block puzzles in 2D with thin walls are NP-hard to solve. We also show certain 3D push-pull block puzzles are PSPACE-complete. This work follows in a long line of algorithms and complexity work on similar problems [Wil91, DDO00, Hof00, DHH04, DH01, DO92, DHH02, Cul98, DZ96, Rit10]. The 2D push-pull block puzzle also shows up in a number of video games, thus implying other results, further continuing the work on understanding video games as in [Vig12, ADGV14, For10, Cor04].

Thesis Supervisor: Erik Demaine
Title: Professor of Computer Science and Engineering

Acknowledgments

I'd like to thank my advisor, Erik Demaine, for his support and insight throughout this project. Isaac Grossof made major contributions to the proofs related to block pushing puzzles; those results would not have occurred without his collaboration. The proof of hardness of Portal with turrets, as well as how to extend the results, grew out of ideas from a correspondence with Joshua Lockhart. The terminology use in this thesis would not have been nearly as consistent without Elizabeth Krueger's editing help. I'd like to thank all of my parents who have been essential in allowing me to get to this point.

Finally, I'd like to dedicate this thesis to Andrea Lincoln who has been my best friend through all of the years of this project. Your intellectual collaboration and emotional support have been invaluable.

Contents

1 Introduction **9**

1.1 Our Results 11

1.1.1 Portal 11

1.2 Push-Pull Block Puzzles 12

2 Portal **13**

2.1 Definitions of game elements 13

2.2 Movement is Easy 16

2.3 Generalized Portal with Emancipation Grills is Weakly NP-hard 17

2.4 Portal with Turrets is NP-hard 19

2.4.1 Literal 19

2.4.2 Variable 19

2.4.3 Clause Gadget 20

2.5 Portal with Timed Door Buttons is NP-hard 21

2.6 Portal with High Energy Pellets and Portals is NP-hard 24

2.7 Portal with Cubes, Doors, Long Fall and Buttons is PSPACE-complete 26

2.8 Portal with Lasers, Relays, Long Fall, and Moving Platforms is PSPACE-complete . 26

2.9 Portal with Gravitational Beam Emitters, Cubes, Weighted Door Buttons, Long Fall, and Moving Platforms is PSPACE-complete 28

3 Push-Pull Block Puzzles **31**

3.1 2D Push-Pull with Thin Walls 31

3.1.1 3SAT Construction 31

3.1.2 Set-Verify Gadgets 33

3.1.3 Crossover Gadgets 33

3.2 3D Push-Pull is NP-hard 36

3.3 PSPACE 38

3.3.1 Toggles 38

3.4 Locks 40

3.4.1 Binary Counter 41

3.4.2 Existential Quantifiers 43

3.4.3 Quantifier Chain 44

3.4.4 Clause Gadget 45

3.4.5 Beginning and End Conditions 45

4 Conclusion **47**

4.1 Open Questions 47

Chapter 1

Introduction

This thesis investigates the computational complexity of motion planning problems described by games and puzzles. Games have been and continue to be a source of interesting problems for mathematics and computer science, and game playing continues to be a benchmark in the field of AI. The study of economic games has become a large field with applications in online auctions [dVV03], network analysis [Pap01], and voting [Col13]. Combinatorial games have led to algebraic insights such as Conway's Surreal Numbers [Con01] and the Sprague-Grundy Theorem [Spr35,Gru39]. More recently, games have been studied from an algorithmic perspective and can be seen as models of computation in examples like Constraint Logic [HD09] and Conway's Game of Life [Ren]. For a survey of algorithmic combinatorial game theory, see Demaine and Hearn's paper [DH09].

Understanding the computational and algorithmic aspects of games may lead to better understanding of algorithms and problems that share similar properties. For example, certain asymmetries or algebraic properties may be more intuitively obvious in games, such as the relationship between two-player games and quantified boolean formulas [Sch78]. Proving results about popular media such as video games may also act as a form of public outreach for mathematics and computer science. The redstone computers in Minecraft are perhaps one of the best examples of computational ideas being looked at in video games by the public [ben14]. Also, the tendency of interesting games to be computationally intractable suggests that this field may be able to usefully contribute to future game design. In addition, many games and puzzles can also be seen as simplified models of problems in the real world. Thus, solving these problems gives us partial understanding and tools to attack more difficult questions.

There has been a surge of recent interest in the computational complexity of video games and puzzles. Examples of previous work in this area includes the NP-completeness of Minesweeper [Kay00], Clickomania [BDD+02], and Tetris [DHLN03], as well as hardness for Lemmings [Cor04, Vig15]. Some surveys of the computational complexity of puzzles, including video games, include Demaine and Hearn [DH09], and Kendall, Parkes and Spoerer [KPS08]. Recent work has moved from puzzles to classic arcade games [Vig12], Nintendo games [ADGV14], 2D platform games [For10], and others [Wal14, ACJ+10, Joh12].

A significant amount of research has gone into characterizing the complexity of block sliding puzzles. This includes PSPACE-completeness for well-known puzzles like sliding-block puzzles [HD05], Sokoban [Cul98, DZ96], the 15-puzzle [RW86], and Rush Hour [FB02]. Block pushing puzzles are a type of block sliding puzzle in which the blocks are moved by a small robot within the puzzles. This type of block sliding puzzle has gathered a significant amount of study. Variations include Sokoban [Cul98, DZ96], where blocks must reach specific targets, variations where multiple blocks can be pushed [DDO00, Hof00, DHH02], versions where blocks continue to slide after being pushed [DHH04, Hof00], versions where the robot can pull blocks [Rit10], and versions where the blocks are subject to gravity [Fri]. The problem of motion planning in an environment where blocks may be pushed and pulled is modeled in a general form in Gordon Wilfong's Motion Planning in the Presence of Movable Obstacles [Wil91]. There he shows a polynomial time algorithm for motion planning with one movable object, NP-hardness for the general planning problem, and PSPACE-hardness for the storage problem. Figure 1.2 gives a summary of results on block pushing puzzles.

In this thesis, we prove results about push-pull block puzzles and the popular video game Portal. Push-pull block puzzles can be seen as a simplified model of robotic forklifts operating in a warehouse. Complexity results in these very simplified models help us understand what aspects of problems makes them hard and allows us to start differentiating between the complexity that arises from the combinatorics vs the geometry of path planning problems. Similarly, path planning in dynamic graphs is a complicated problem which occurs every day from traffic to packet routing. Portal gives us a setting where we can explore motion planning problems in an environment with changing geometry and topology.

1.1 Our Results

1.1.1 Portal

Portal and its sequel, Portal 2, are first-person puzzle-platform video games created by Valve Corporation. It is estimated that Portal has sold over 12.5 million copies [YP, stea] and Portal 2 has sold over 10.8 million copies [Cao, steb]. In Portal, the player navigates the main character, Chell, through a series of 'test chambers' filled with physical puzzles constructed by the game's antagonist, GLaDOS. The puzzles often make use of portals that the player can place. Portals connect two physical locations and also preserve objects' momentum as they travel through the portal.

We analyze some of the main mechanics of Portal and give NP-hardness and PSPACE-hardness results. In many cases the mechanics and proof techniques will generalize to a number of other games. Table 1.1 gives a collection of our results for the video games Portal and Portal 2 with various mechanics. We refer to the composition of the mechanics in Portal 1 and 2 as simply Portal. The first column lists the primary mechanics of Portal we are investigating. The second and third column note whether the long fall or portal gun mechanics are needed for the proof. More details on what these models mean can be found in Section 2.1.

Mechanics	**Portals**	**Long Fall**	**Complexity**
None	No	Yes	P (§2.2)
Emacipation Grills and No terminal velocity	Yes	Yes	Weakly NP-hard (§2.3)
Cubes, Weighted Buttons, Doors	No	Yes	PSPACE-comp. (§2.7)
Turrets	No	Yes	NP-hard (§2.4)
HEP Launcher and Catcher	Yes	No	NP-hard (§2.6)
Timed Door Buttons and Doors	No	No	NP-hard (§2.5)
Lasers, Relays, Moving Platforms	Yes	No	PSPACE-comp. (§2.8)
Gravity Beams, cubes, Weighted Buttons, Doors	No	No	PSPACE-comp. (§2.9)

Table 1.1: Summary of New Portal Complexity Results

Although Viglietta [Vig12] argues that most modern games include Turing-complete scripting languages and thus could allow designers to create undecidable puzzles, we feel that many games have consistent sets of mechanics from which puzzles are built and analyzing the complexity of games and puzzles that can be created with those rule sets is interesting and meaningfully captures what we think of as the game they compose.

Name	*Push*	*Pull*	*Blocks*	*Fixed?*	*Path?*	*Sliding*	*Complexity*
Push-k	k	0	Unit	No	Path	min	NP-hard [DDO00]
Push-$*$	$*$	0	Unit	No	Path	min	NP-hard [Hof00]
PushPush-k	k	0	Unit	No	Path	Max	PSPACE-c. [DHH04]
PushPush-$*$	$*$	0	Unit	No	Path	Max	NP-hard [Hof00]
Push-kX	k	0	Unit	No	No-Cross	min	NP-c. [DH01]
Push-$*$X	$*$	0	Unit	No	No-Cross	min	NP-c. [DH01]
Push-1F	1	0	Unit	Yes	Path	min	NP-hard [DO92]
Push-kF	$k \geq 2$	0	Unit	Yes	Path	min	PSPACE-c. [DHH02]
Push-$*$F	$*$	0	Unit	Yes	Path	min	PSPACE-c. [DHH02]
Sokoban	1	0	Unit	Yes	Storage	min	PSPACE-c. [Cul98]
Sokoban$^+$	$k \geq 2$	1	2x1	Yes	Storage	min	PSPACE-c. [DZ96]
Motion Planning	k	1	L	Yes	Storage	min	NP-hard [Wil91]
Sokoban$(k, 1)$	$k \geq 5$	1	Unit	Yes	Storage	min	NP-hard [DZ96]
Pull-1	0	1	Unit	No	Storage	min	NP-hard [Rit10]
Pull-kF	0	k	Unit	Yes	Storage	min	NP-hard [Rit10]
PullPull-kF	0	k	Unit	Yes	Storage	Max	NP-hard [Rit10]
Push-1G[1]	1	0	Unit	Yes	Path	min	NP-hard [Fri]
Push-k Pull-lW	k	l	Unit	Wall	Path	min	**NP-hard** (§3.1)
3D Push-k Pull-lF	k	l	Unit	Yes	Path	min	**NP-hard** (§3.2)
3D Push-1 Pull-1W	1	1	Unit	Wall	Path	min	**PSPACE-c.** (§3.3)
3D Push-k Pull-kF	$k > 1$	$k > 1$	Unit	Yes	Path	min	**PSPACE-c.** (§3.3)

Table 1.2: Summary of Past and New Block Pushing Puzzle Results. $*$ refers to an unlimited number of blocks. F means fixed blocks are included. X means the robot cannot step on the same square more than once. G means the blocks are subject to gravity. W means thin walls are included.

1.2 Push-Pull Block Puzzles

We give new results showing that certain block pushing puzzles which include the ability to push and pull blocks are NP-hard or PSPACE-complete. We introduce *thin walls*, which prevent motion between two adjacent empty squares. We prove that all path planning problems in 2D with thin walls or in 3D, in which the robot can push k blocks and pull l blocks for all $k, l \in \mathbb{Z}^+$ are NP-hard. Our results are shown in the last four lines of Table 1.2. As with many of these problems, closing the gap between NP and PSPACE remains open.

Chapter 2

Portal

In this chapter we prove several different results about the video game Portal. We show generalized Portal with Emancipation Grills is weakly NP-hard2.3; Portal with turrets is NP-hard2.4; Portal with timed door buttons and doors is NP-hard2.5; Portal with High Energy Pellet launchers and catchers is NP-hard2.6; Portal with Cubes, Weighted Buttons, and Doors is PSPACE-complete2.7; Portal with lasers, laser relays, and moving platforms is PSPACE-complete2.8; and Portal with gravity beam emitters, cubes, weighted buttons, and doors is PSPACE-complete without long falls2.9. These results are summarized in Table 1.1. This chapter is joint work with Erik Demaine and Joshua Lockhart.

2.1 Definitions of game elements

A platform game is a single player game in which the player solves a series of levels, which consist of navigating the player's avatar from a start location to an end location. The game-play in Portal involves walking, turning, jumping, crouching, pressing buttons, picking up objects, and creating portals. All positions and movement in the game are discretized. When objects are picked up, they stay in front of Chell unless there is an obstruction in the way. Since we are primarily concerned with objects being placed onto buttons or on the floor allowing Chell to reach higher places, we are not concerned with the details of the motion of objects which have been picked up. The Portal Gun creates a portal on the closest surface in a direct line from the player's avatar if the surface is of the appropriate type. We call surfaces that admit portals, "portalable". There are a variety of

other elements which can be placed in a level of Portal; they are described below.

1. A *long fall* is a fall which is farther than the avatar can jump but still safe for the avatar.

2. A *door* is a game element which can be open or closed, and which can be traversed by the player's avatar if and only if it is open. In Portal, many mechanics can act as doors, such as literal doors, laser fields, and moving platforms. In this paper, on a few occasions we will assume the door being used also blocks other objects in the game, such as High Energy Pellets or lasers, which is not generally true.

3. A *button* is an element which can be interacted with when the avatar is nearby to change the state of the level, usually by opening or closing a door.

4. A *timed button* is a switch which will revert the change made by interacting with it after a set period of time. A timed button cannot be interacted with between the time it is flipped and the time it reverts.

5. A *weighted floor button* is a an element which changes the state of a level when one or more of a set of objects is placed on it. In Portal, the 1500 Megawatt Aperture Science Heavy Duty Super-Colliding Super Button is an example of a weighted floor button which activates when the avatar or a Weighted Storage Cube is placed on top of it. An activated weighted floor button can activate other mechanics such as doors, moving platforms, laser emitters, and gravitational beam emitters.

6. A *block* is an object which can be moved by the avatar and whose location will change whether a path is traversable by the avatar, usually by either blocking a path or allowing the avatar to jump on top of it and then jump to an area higher than they could previously reach. In Portal, the Weighted Storage Cube is an example of a block that can be jumped on or used to activate weighted floor buttons. We will often refer to Weighted Storage Cubes, Companion Cubes, etc. as cubes.

7. A *lethal hazard* is an object which will cause the avatar to die if they come in contact with it. Some examples from Portal are the High Energy Pellets and Goo, a lethal hazard that replaces part of a floor.

8. A *Material Emancipation Grill*, also called an Emancipation Grill or fizzler, destroys some

objects which attempt to pass through it, such as cubes and turrets. When the avatar passes through an Emancipation Grid, all previously placed portals are removed from the map.

9. A *Portal Gun* allows the player to change the state of a section of a portalable surface, which is in line-of-effect to their avatar, to either a blue or orange portal. Whenever a new surface obtains a blue or orange portal the previous blue or orange portal is removed. The blue and orange portals are always co-located, regardless of the previous geometry of the surfaces they were placed upon.

10. A *High Energy Pellet* (HEP) is a spherical object which moves in a straight line until it encounters another object. HEPs move faster than the player avatar and if they collide with the player avatar, the avatar is killed. If a HEP encounters another wall or object, it will bounce off of that object with equal angle of incidence and reflection. In Portal, some HEPs have a finite lifespan, which is reset when the HEP passes through a portal, and others have an unbounded lifespan. These unbounded HEPs are referred to as *Super High Energy Pellets*.

11. A *HEP Launcher* emits a HEP normal to the surface upon which it is placed. These are launched when the HEP launcher is activated or when the previous HEP emitted has been destroyed.

12. A *HEP Catcher* is a device which is activated if it is ever hit by a HEP. In Portal, this device can activate other mechanics, such as doors or moving platforms, when activated.

13. A *Thermal Discouragement Beam*, which we will call a *laser*, is a red cylinder which emanates perpendicularly from a *Laser Emitter* and continues in a straight line until it is stopped by a wall or another object. If the avatar is in sufficiently close proximity to the laser for succinctly long they will die.

14. A *Laser Relay* is an object which can activate other objects while a laser passes through it.

15. A *Laser Catcher* is an object which can activate other objects while a contacts it.

16. A *Moving Platform* is a solid polygon with an inactive and an active position. It begins in the inactive position and will move in a line at a constant velocity to the active position when activated. If it becomes deactivated it will move back to the inactive position with the opposite velocity.

17. A *Turret* is an enemy which cannot move on it's own. If the player's avatar is within the field of view of a turret, the turret will fire on the avatar. If the avatar is shot sufficiently many times within a short period of time, the avatar will die.

18. A *Gravitational Beam Emitter* emits a gravitational beam normal to the surface upon which it is placed. The gravitational beam is directed and will move small objects at a constant velocity in the prescribed direction. Importantly, it will carry Weighted Storage Cubes and the player avatar. Gravitational Beam Emitters can be switched on and off, as well as flipping the direction of the gravitational beam they emit.

A level in Portal is a description of polygonal surfaces in space, and a list of other game elements with their locations and, if applicable, connections to each other. The Portal Problem asks whether there exists a path from a start location in a level to an end location in that level.

There are two main pieces of software for creating levels in Portal 2: The Portal 2: Puzzle Maker and the Valve Hammer Editor with the Portal 2 Authoring Tools. Both of these are available for player to create their own levels. The Puzzle Maker has a smaller set of tools with which to build levels and a different, friendly UI to allow different audiences to create levels for the game. Some notable differences are levels must be polycubes, course discretization of the level space and location objects can be placed, no HEP launchers, and no additional doors. We will often comment concerning which reductions can be constructed with the additional Puzzle Maker restrictions (except, of course, the small level size and item count) but this distinction is not a primary focus of this work.

2.2 Movement is Easy

Theorem 2.1. *A Portal level without portalable surfaces and without any additional mechanics besides traversing the level is in P.*

Proof. We will follow a similar argument to that used in Forisek's Meta-theorem 1 [For10]. The level and allowed movements of the avatar uniquely determine a directed reachability graph. Since the interactions of the geometries of the level and the avatar are relatively simple, this reachability graph can be determined in polynomial time. One then merely needs to perform a Depth First Search for a valid path from the Start to the Exit. □

2.3 Generalized Portal with Emancipation Grills is Weakly NP-hard

We will prove Portal with portals and Emancipation Grills is weakly NP-hard by reduction from Subset-Sum. Subset-Sum asks, given a set of integers $S = \{a_1, a_2, \ldots, a_n\}$, is there a subset of S which sums to a target value T [GJ79]. We plan to encode these numbers in the avatar's velocity. The input will be constructed from long holes the avatar can fall down, and the target will be encoded in a distance the avatar must launch themselves after falling. In the video game, there is a maximum velocity the player avatar can reach. Since this restriction is likely a practical concern, we will imagine Portal without the restriction of a terminal velocity, or one that scales with the map size.

Theorem 2.2. *Portal with portals, long fall, and generalized terminal velocity is weakly NP-hard.*

Proof. The elements of S are represented by a series of wells, each of depth $2 \cdot a_i \cdot n^2 \cdot \epsilon$, where a_i is the number to be encoded, n is the number of elements in S, and ϵ is an expansion factor to be described shortly. An example is shown in Figure 2-1. The bottom of each well is a portalable surface, and the ceiling above each well is also a portalable surface. This will allow the avatar to shoot a portal to a ceiling tile, and to the bottom of the well they are falling into, selecting the next number.

We cannot allow the avatar to select the same element more than once. The Emancipation Grills below each portalable ceiling serve to remove the portal from the ceiling of the well into which they are currently falling, and to prevent sending a portal up to that same ceiling tile. The stair-stepped ceiling will allow the player to see the ceilings of all of the wells with index greater than the one they are currently at, but prevents them from seeing the portalable surface of the wells with the same or lower index. This ensures that the player can only select each element once.

The enforced order of choosing does not matter when solving Subset Sum. The distance between each step is ϵ, thus to ensure the accumulated error from falling these distances does not impact the solution to the subset sum, we scale each a_i by $2 \cdot n^2 \cdot \epsilon$, which is greater than the sum of all of these extra distances. We also wish to choose ϵ to be bigger than the distance the avatar can jump. The verification gadget involves two main pieces: a single portalable surface on a wall, the launching portal, and a target platform for the player to reach. We place the launching portal so it

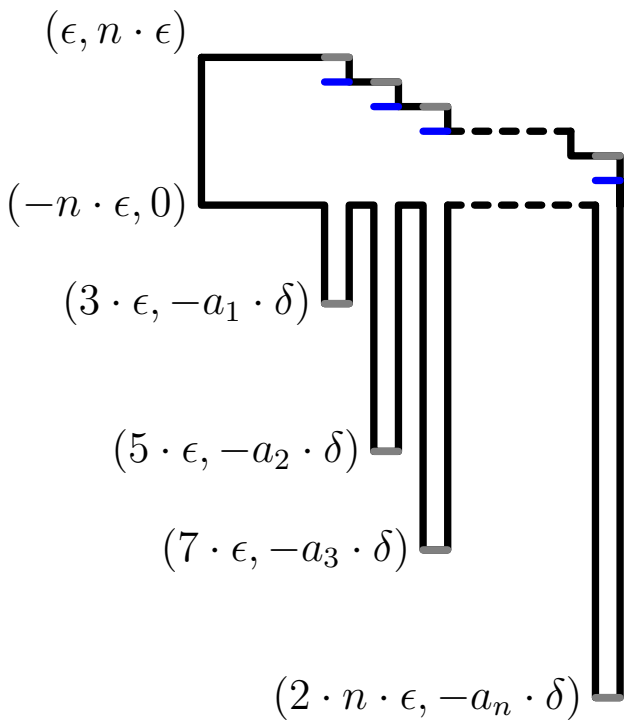


Figure 2-1: A cross-section of the element selection gadget. Grey lines are portalable surfaces and blue lines are Emancipation Grills. The diagram is not to scale, as the elements of S would need to be much larger than depicted.

can always be shot from the region above the wells. The target platform is placed $g/2$ units below the launching portal, where g is the acceleration due to gravity. The target platform is placed a distance of $T \cdot n^2 \cdot \epsilon$ away from the launching portal and in front of the portalable surface such that leaving the portalable surface with the target velocity will cause the player to reach the target platform. Since it takes 1 second to fall the vertical distance to the platform, the avatar will only reach the target if their velocity is equal to $T \cdot n^2 \cdot \epsilon$. We make the target platform $n^2 \cdot \epsilon$ on each side, to account for any errors incurred by the falling region or initial horizontal movement. This size is smaller than the difference if the target value T differed by 1. We now have an encoding of our numbers, a method of selecting them, and verification if they reach the target sum, completing the reduction. □

We note that although all elements needed for this construction can be placed in the Puzzle Creator, maps made in this way will not work for the reduction. Because Subset Sum is only weakly NP-hard, we need the values of the elements of S to be exponential in n. Thus we need to describe the map in terms of coordinates specifying the polygons making up the map, whereas the Puzzle

Creator specifies each voxel in the map.

2.4 Portal with Turrets is NP-hard

In this section we prove that Portal with Turrets is NP-hard and provide a general method for proving many 3D platform games with enemies are hard. The initial idea for this technique and a sketch of a proof for Half-Life was provided by Joshua Lockhart. Although enemies in a game can provide interesting and complex interactions, we can pull out a few simple properties that will allow them to be used as gadgets to reduce solving a video game from 3SAT. This proof follows the architecture laid out in [ADGV14].

1. First, the enemy must be able to prevent the player from traversing a specific region of the map, call this the *blocked region*.

2. Second, player avatar must be able to remove the enemy from an area of the map. Further, it must be possible to make that area path-disconnected from the blocked region and still permit the player avatar to remove the enemy.

If the map also contains one-way gadgets, for example long falls, then the game will be NP-hard. Similarly, a 2D game with these elements and an appropriate cross-over gadget should also be NP-hard. The following is a construction proving Portal with Turrets is NP-hard using this technique.

2.4.1 Literal

Each literal consists of a hallway to be traversed with three Turrets placed on a raised section off of the hallway. If the turrets are active, they will kill the avatar before the avatar can cross the hallway or reach the turrets. The literal is true if the turrets are deactivated or removed, and false if they are active. See Figure 2-2

2.4.2 Variable

The variable consists of a hallway that splits. Each hallway starts and ends with a one-way gadget constructed with a long fall, to ensure the avatar can only traverse one hallway. This is shown in

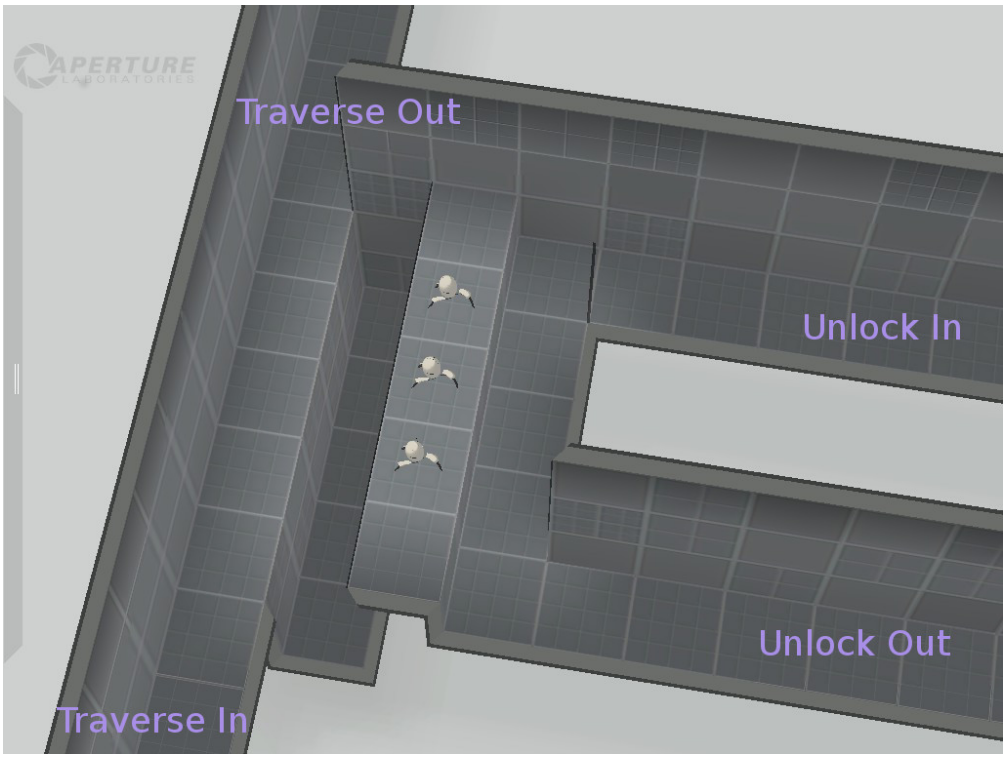


Figure 2-2: An example of a (currently) false literal constructed with Turrets. Labels added over the screenshot.

Figure 2-3. The hallways allow access to the backs of the turrets corresponding to either all the true or all the false literals for that variable. The turrets are located on a platform, such that they can be knocked over by the avatar, but the avatar cannot jump onto the platform or otherwise cross into the hallway representing the literal.

2.4.3 Clause Gadget

Each clause is constructed simply as three hallways in parallel, each containing a literal. The avatar can progress from one end of the clause to the other if any of the literals is true, and thus passable.

Theorem 2.3. *Portal with Turrets is NP-hard.*

Proof. Given an instance of a 3SAT problem, we can translate it into a Portal with Turrets map using the above gadgets. This map is solvable if and only if the corresponding 3SAT problem is solvable. □

It is interesting to note some of the other games this can apply to. For example, games with

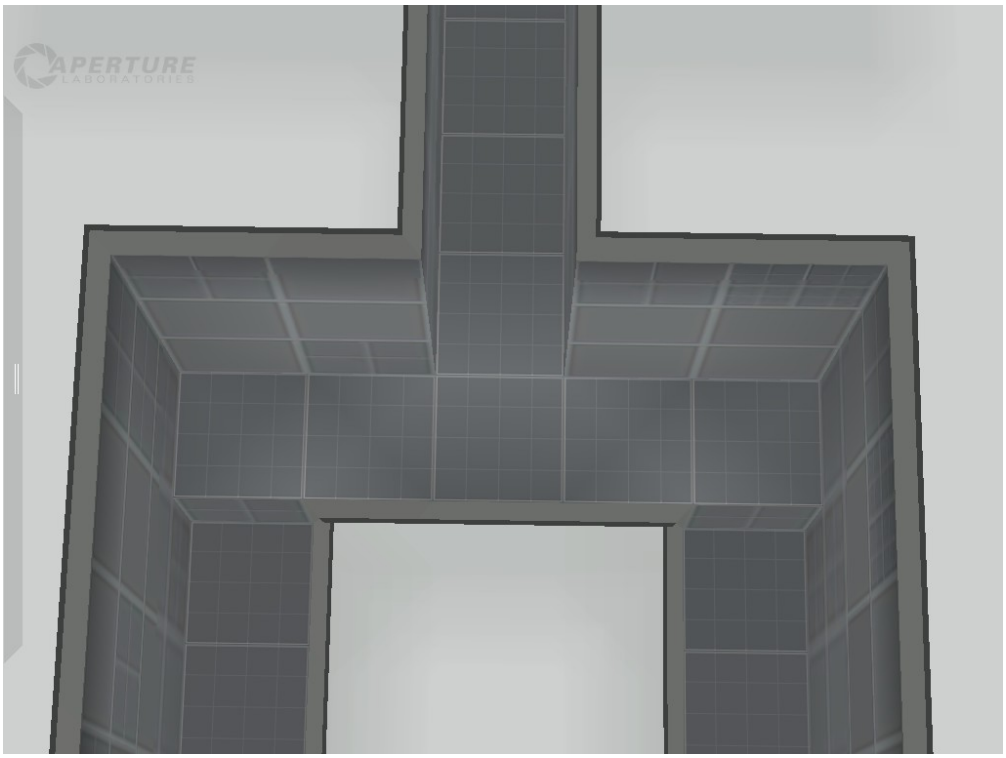


Figure 2-3: An example of the choice gadget used to construct variable gadgets.

stationary turrets, such as the Emplacement Gun in Half-Life 2 or the Type-26 ASG in Half-Life, can force the player avatar to only be able to remove enemies from a specific location. Similarly, games with ranged weapons and strong melee enemies, such as Halo 2, World of Warcraft, Left 4 Dead, The Legend of Zelda, BioShock, Doom, and Diablo, should permit similar constructions. Another general category includes games with stationary enemies that have a fixed field of attack, such the turret in Portal and the sentry guns in Team Fortress and Call of Duty.

2.5 Portal with Timed Door Buttons is NP-hard

We provide a new Meta-Theorem related to Forisek's meta-theorem 2 [For10] and Viglietta's meta-theorem 1 [Vig12].

Theorem 2.4. *A platform game with doors controlled by timed switches is NP-hard.*

Proof. We will prove hardness by reducing from finding Hamiltonian cycles in grid graphs [IPS82]. Every vertex of the graph will be represented by a room with a timed switch in the middle. These rooms will be laid out in a grid with hallways in-between, which will take some minimum time to

traverse. All of the timed switches will be connected to a series of closed doors blocking a hallway off of the start node. The timers will be set, such that the doors will close again after $\alpha(t+1)+\epsilon$ time where α is the time it takes to flip a switch and travel from one switch to an adjacent one in the graph, and ϵ is the time it takes to move from the switch at the start node through the open doors to the exit. The exit is thus only reachable if all of the timed switches are simultaneously active. Since we can make α much larger than ϵ, we can ensure that there is only time to visit every switch exactly once and then pass through before any of the doors revert. □

Corollary 2.5. *A Portal level with only timed door buttons is NP-hard.*

A screenshot of an example map is given in Figure 2-4. Since the Portal 2 Workshop does not allow additional doors, the example uses collapsible stairs, as seen in Figure 2-6 for the verification gadget instead. We note that anything which will prevent the player from passing unless currently activated by a timed button will suffice. Moving platforms and Laser Fields are other examples.

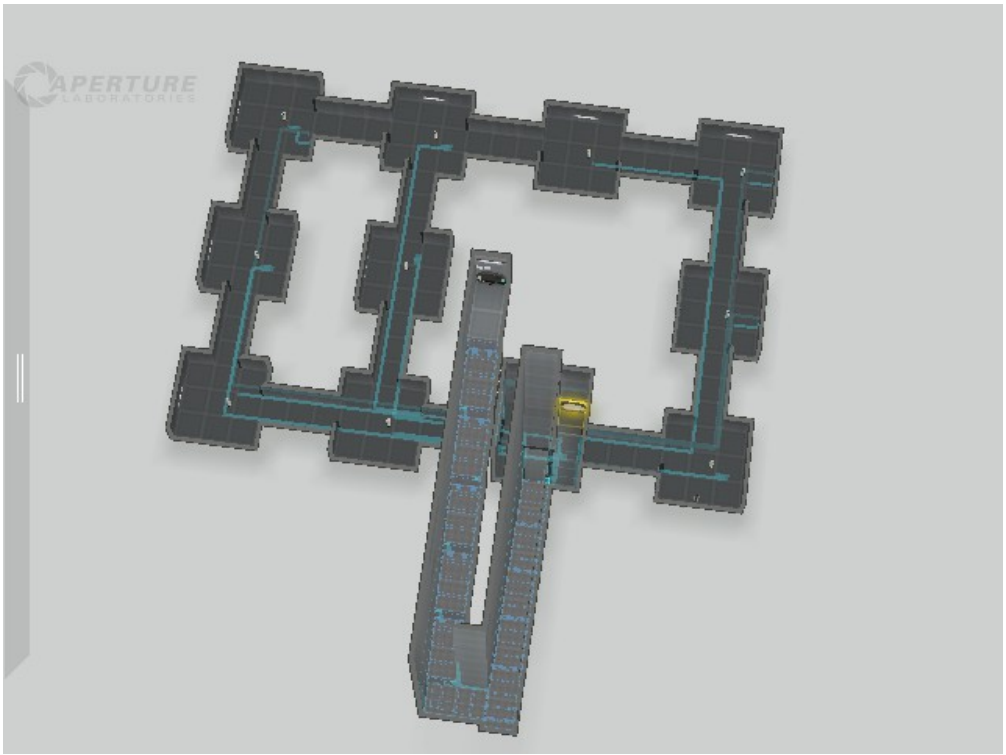


Figure 2-4: An example of a map forcing the player to find a Hamiltonian path in a grid graph.

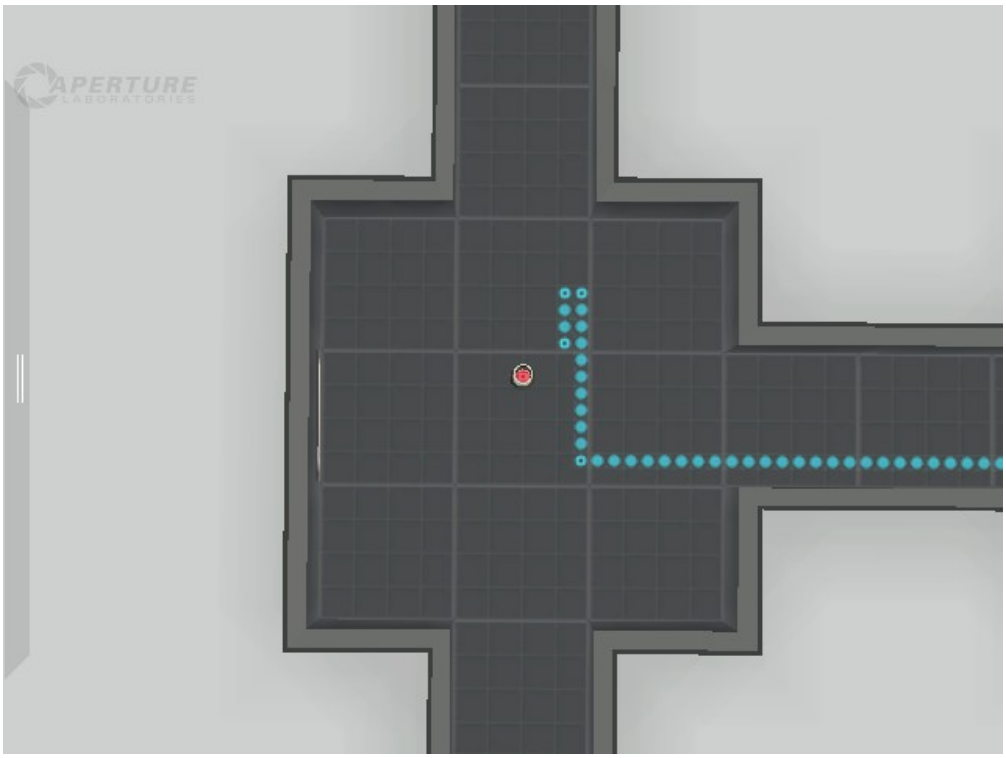


Figure 2-5: Close-up of a node in the grid graph.

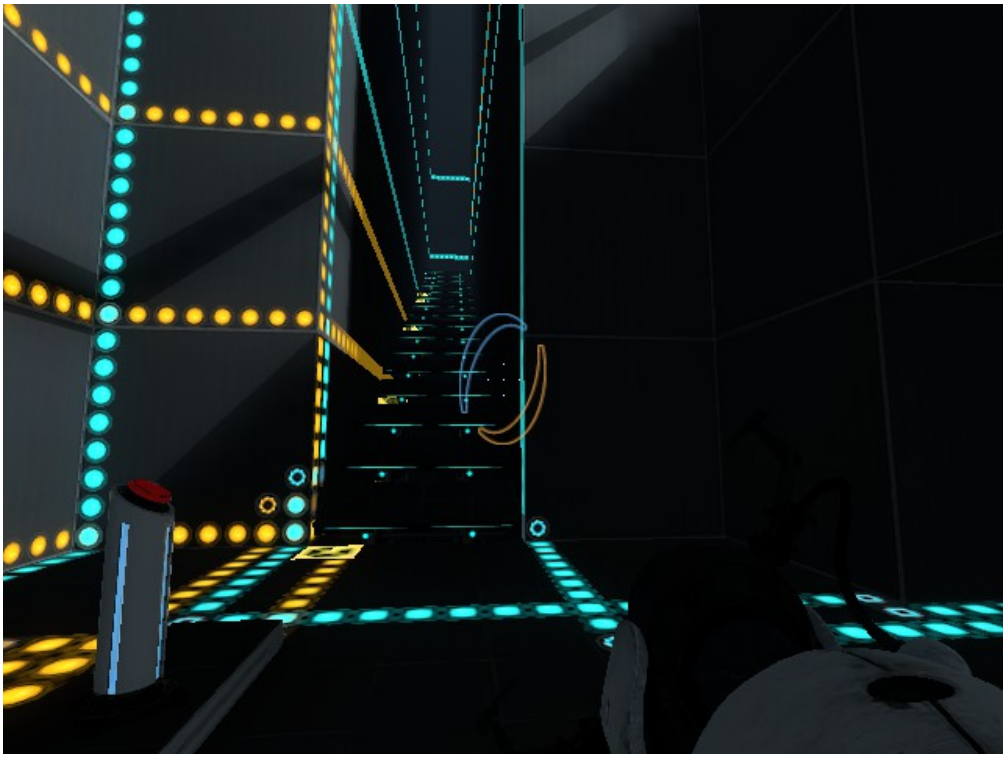


Figure 2-6: A screenshot of the verification gadget, partially satisfied.

2.6 Portal with High Energy Pellets and Portals is NP-hard

In Portal, the High Energy Pellet, HEP, is an object which moves in a straight line until it encounters another object. HEPs move faster than the player avatar and if they collide with the player avatar, the avatar is killed. If a HEP encounters another wall or object, it will bounce off of that object with equal angle of incidence and reflection. In Portal, some HEPs have a finite lifespan, which is reset when the HEP passes through a portal, and others have an unbounded lifespan. These unbounded HEPs are referred to as *Super High Energy Pellets*. A HEP launcher emits a HEP normal to the surface it is placed upon. These are launched when the HEP launcher is activated or when the previous HEP emitted has been destroyed. A HEP catcher is another device that is activated if it is ever hit by a HEP. When activated this device can activate other objects, such as doors or moving platforms.

Theorem 2.6. *Portal with HEP launchers, HEP catchers, and doors controlled by HEP catchers is NP-hard.*

Proof. We will reduce from finding Hamiltonian cycles in grid graphs [IPS82]. For this construction, we will need a gadget to ensure the avatar traverses every represented node, as well as a timing element. Each node in the graph will be represented by a room that contains a HEP launcher and a HEP catcher. They are positioned near the ceiling, each facing a portal-able surface. The HEP catcher is connected to a closed door preventing the avatar from reaching the exit. Proof of theorem 2.4 uses the same idea and has an example of how rooms in Portal can be connected to simulate a grid graph.

The timer will contain two elements. First, we will arrange for a hallway with two exits and a HEP launcher behind a door on one end. The hallway is long enough so it is impossible for the avatar to traverse the hallway when the door is open. Call this the time verifier. In another area, we have a HEP launcher and a HEP catcher on opposite ends of a hallway that is inaccessible to the avatar. The catcher in this section will open the door in the time verifier. This allows us to ensure that the player can only pass through the time verifier if they enter it before a certain point after starting. To complete the proof, we set the timer equal to $\alpha N + \delta$ where δ is the minimum time needed before the door is opened in the time verifier for the avatar to be able to successfully traverse the time verifier, α is the minimum time it takes for the player to move to an adjacent room and change the trajectory of the HEP, and $N + 1$ is the number of HEP catchers in the level. This concludes the reduction to the Grid Hamiltonian Path problem. □

Theorem 2.7. *Portal with Super High Energy Pellets is NP-hard.*

Proof. This reduction uses the same ideas as the previous proof and also reduces from finding Hamiltonian cycles in grid graphs. Once again we need a timing element, and a gadget to ensure the avatar reaches every node. Each node will be a room with access to a Super HEP (shortened to HEP for this section) which must be displaced, otherwise it will impede the players movement in the future. Our timer is a pair of HEPs which will eventually enter a hallway the player must pass. We will actually use two sets of timing elements to complete the reduction.

To construct a node, we have a room with access to a chimney which has portalable walls and a HEP bouncing at a 45 degree angle between two walls and down the chimney. Partway down the chimney the HEP has an escape tunnel which it will bounce into. The room also contains a storage section with a portalable surface out of reach of the avatar. The avatar is thus able to shoot a portal into the storage section and the chimney so the HEP gets trapped in the storage section. If the number of nodes in the simulated graph is N, the HEP will take time T to reach the escape tunnel, where T is N times the time needed to go between adjacent rooms and shoot portals into the chimney and storage area.

The escape tunnels all feed into a long hallway, the Hamiltonian checking hallway (Hch), which will take the avatar $2T$ time to traverse. This hallway is connected to the end node in the Hamiltonian circuit by a long path which takes the avatar $T + D$ time where D is the time it takes an HEP to travel from the escape tunnel to the Hch. Note, the escape tunnels will likely have to wind to make their paths the same length, and we may similarly want to run the avatar back and forth in the hallway before the Hch. If the avatar has not successfully stored all of the HEPs in the graph nodes, they will have to wait for an HEP to travel down the Hch and back, if not longer. With this delay, they will be unable to to the target location in time.

The final timing element involves making the entire ground level of the map except for the target location lethal after a specific time. Except for the chimneys and escape tunnels, all of the ceilings in the hallways and rooms are very high. In addition, there are a large number of HEPs far up heading downward, except at the target location. After $4T + D$ time the HEPs will arrive at ground level and kill the player avatar. Note, some of these will need to be angled to get around the chimneys and escape tunnels; however, it is not difficult to ensure that every avatar sized bounding box will contain an HEP at the $4T + D$ time limit. □

2.7 Portal with Cubes, Doors, Long Fall and Buttons is PSPACE-complete

In this section we show that Portal with Weighted Storage Cubes, doors, long fall, and Heavy Duty Super Buttons is PSAPCE-complete. Companion Cubes or Discouragement Redirection Cubes in combination with the Weighted Storage Receptacle can also replace the button. Edgeless Safety Cubes and Edgeless Safety Cube Receptacles are also adequate replacements.

For this proof we will be following Forisek's Meta-theorem 4: "Any 2D platform game that exhibits the features **long fall**, **opening doors** and **closing doors** is PSPACE-hard." [For10] Here our definition of long fall is the same, however, we need to discuss what is meant by opening and closing doors. Opening and closing doors refers to a mechanic in a game which, when interacted with, can cause multiple doors to open or close. Forisek's paper seems to explicitly mean game elements that look like doors; however, we note our more general notion of door is sufficient. In addition, being a 3D platform game does not cause any of Forisek's constructions to fail as we can create thin levels wherever needed. The added flexibility will only make the game harder.

Theorem 2.8. *Portal with Weighted Storage Cubes, doors, long fall, and Heavy Duty Super Buttons is PSPACE-complete.*

Proof. From here, the result follows almost immediately. We have long falls and doors. The buttons can be connected to open doors when depressed. Placing a cube on a button causes the corresponding doors to open, and removing it causes them to close. This concludes the proof that Portal with cubes, buttons, doors, and long fall is PSPACE-complete. We also note that all of these elements can be found in the Portal Puzzle Maker if one uses stairs or moving platforms as doors. □

2.8 Portal with Lasers, Relays, Long Fall, and Moving Platforms is PSPACE-complete

Here we will prove Portal with lasers, laser relays, moving platforms, long fall and portals is PSPACE-complete. Laser catchers can also be used, although it is interesting that the laser relays need only be connected to a single platform each. This proof also uses Forisek's Meta-theorem 4 which states that games with long fall, and the ability to open and close doors is PSPACE-

complete [For10]. See Section 2.7 for more details. Here doors can be implemented with moving platforms which move into place to allow the avatar to walk over a long fall. Moving platforms are also of interest because they can block lasers. Laser relays can open doors, and we have long fall so it seems we'd be done; however, opening and closing doors must be continued change. If we use our portals to redirect the lasers, then we could only affect a single set of doors at a time. To solve this, we introduce a memory latch gadget, shown in Figures 2-7 and 2-8. When the relay in this gadget is activated for sufficiently long, the platform will move out of the way and the laser will keep the relay active. If the relay has been blocked for enough time, the platform moves back and blocks the laser. Thus, the gadget will remember which of two states it was set to.

Figure 2-7: A memory latch in the off state.

Theorem 2.9. *Portal with lasers, relays, long fall, portals, and moving platforms is PSPACE-complete.*

Proof. Moving platforms can function as doors and can be controlled by laser relays. Memory latches can be augmented with additional laser relays that connect to other doors, allowing them to be opened. Moving platforms can block lasers, allowing the activation of a moving platform to stop a laser and close doors. The avatar can redirect lasers to activate memory latches which provides the opening and closing doors component of Forisek's Meta-theorem 4. Allowing long fall completes the proof. □

Figure 2-8: A memory latch in the on state.

Corollary 2.10. *Portal with lasers, relays, long fall, Discouragement Redirection Cubes, and moving platforms is PSPACE-complete.*

Proof. Discouragement Redirection Cubes can be used to redirect a laser's pathway in the same way portals are used in Theorem 2.11. All other mechanics necessary to fulfill Forisek's Meta-theorem 4 remain. □

2.9 Portal with Gravitational Beam Emitters, Cubes, Weighted Door Buttons, Long Fall, and Moving Platforms is PSPACE-complete

We prove some additional results about gravitational beams, weighted cubes, and weighted cube buttons by showing they can emulate the same gadgets as lasers and laser catchers. When active, a gravity beam causes objects which fit inside its diameter to be pushed or pulled in line with the gravity beam emitter. Objects in the gravity beam ignore the normal pull of gravity, and thus float along their course. If a weighted cube button is placed in the path of a gravity beam, a weighted cube caught in the beam can depress the button as in Figure 2-9. A cube on the floor near a gravity beam, as in Figure 2-10 will be picked up by the beam. Weighted cube buttons can activate and deactivate the same mechanics as laser catchers, including gravity beam emitters. Figures 2-9 and

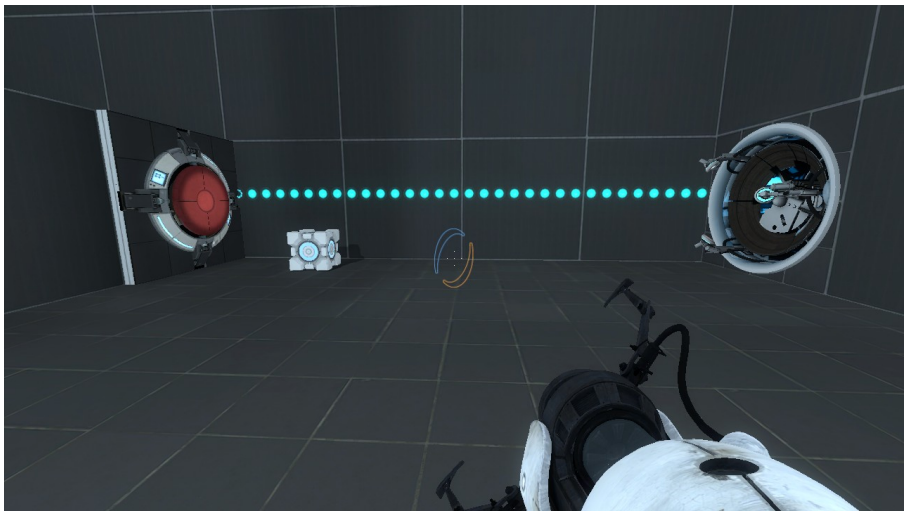


Figure 2-9: A memory latch in the off state.

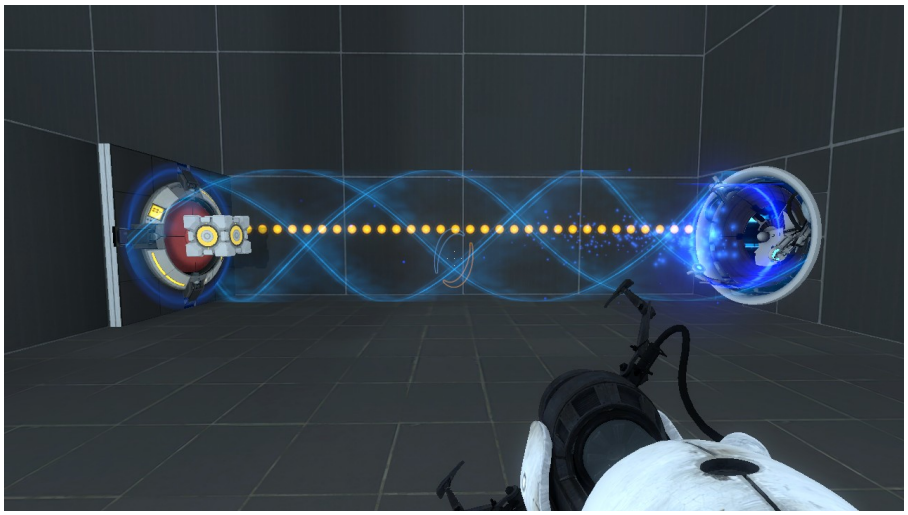


Figure 2-10: A memory latch in the on state.

2-10 demonstrate a memory latch in the off and on positions, respectively. We also note that gravity beams are blocked by moving platforms, just like lasers. At this point we now have the properties we needed from the laser, laser catcher, and moving platform. We also note that the player can pick up and remove cubes from the beam, thus portals are not needed.

Corollary 2.11. *Portal with gravity beams, cubes, weighted cube buttons, long fall, and moving platforms is PSPACE-complete.*

Proof. We emulate the proof from Section 2.8 using the method described above. □

Chapter 3

Push-Pull Block Puzzles

In this chapter we prove several results about push-pull block puzzles. We show Push-1 Pull-1 in 3D with thin walls is PSPACE-complete3.3; Push-i Pull-j in 3D is PSPACE-complete for all positive integers $i, j \geq 2$3.3; Push-k Pull-l in 3D is NP-hard for all positive integers k, l3.2; and Push-q Pull-r in 2D with thin walls is NP-hard for all positive integers q, r3.1. A summary of these results can be seen in Table 1.2. This chapter is joint work with Erik Demaine and Isaac Grosof.

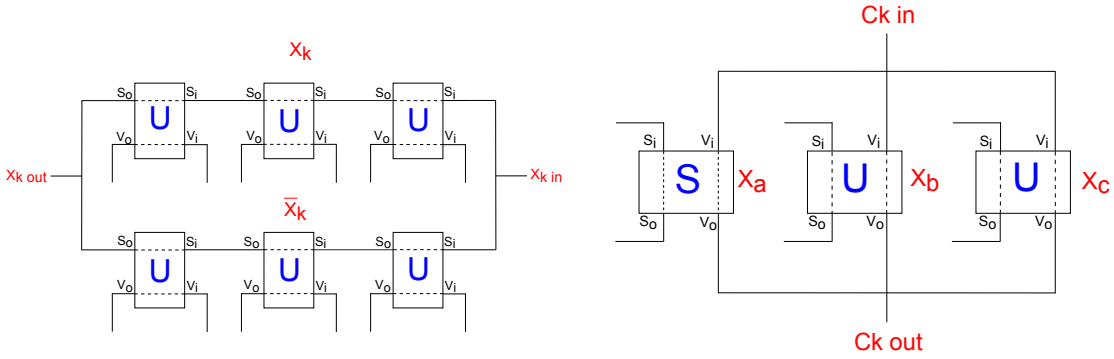
3.1 2D Push-Pull with Thin Walls

In this section we prove that Push-k Pull-l in 2D with fixed blocks is NP-hard, for all positive k and l, if we include *thin walls*. Thin walls are a new, but natural, notion for block pushing puzzles, which prevent blocks or the robot from passing between two adjacent, empty squares, as though there were a thin wall blocking the path. These were needed because many of the gadgets depended greatly on having very tight corridors to ensure limited behavior. We will prove hardness by a reduction from Planar 3SAT.

Theorem 3.1. *Push-k Pull-l in 2D with thin walls is NP-hard.*

3.1.1 3SAT Construction

We reduce from 3SAT and will be making use of the Set-Verify gadget to produce our literals. One significant difficulty with this model is the complete reversibility of all actions. Thus we need to



(a) A variable gadget representing X_k occurring in three clauses.

(b) A clause gadget with variables x_a, x_b, x_c.

Figure 3-1

take care to ensure that going backward at any point does not allow the robot to cheat in solving our 3SAT instance. The directional properties of the Set-Verify allow us to create sections where we know if the robot exits, it must have either reset everything to the initial configuration or put everything in another known state.

Our literals will be represented by Set-Verify gadgets, described in Section 3.1.2. They are considered true when the V_i to V_o traversal is possible, and false otherwise. Thus we can set literals to true by allowing the robot to run through the S_i to S_o passage of the gadget. This allows a simple clause gadget, shown in Figure 3-1b, consisting of splitting the path into three hallways, each with the corresponding verify side of our literal. We can then pass through if any of the literals is set to true and cannot pass otherwise.

The variables will be encoded by a series of passages which split to allow either the true or negated literals to be set, shown in Figure 3-1a. Once variables split into two hallways, we have to show that when you enter or exit a side of a hallway, all the literals are set to true at one end and false at the other. Also, we cannot go back through the other hallway, so if we go backward, everything gets reset. Thus we cannot exploit the reversibility to set anything extra true.

The variables and clauses must be joined together to allow the robot to move between the gadgets in the correct order. We can connect these with simple, empty hallways, except in the cases where these hallways must cross. Although we reduce from Planar 3SAT, it is not known whether the problem remains hard if we further connect all of the variable nodes and then the edge nodes in a single path, as is required in our construction, while still ensuring that the graph is planar.

Therefore, in order to traverse all of the clauses and reach the goal location, the robot must traverse a set of variable hallways that encode an accepting assignment to the 3SAT problem, thus reducing 3SAT to 2D Push-Pull block puzzles.

3.1.2 Set-Verify Gadgets

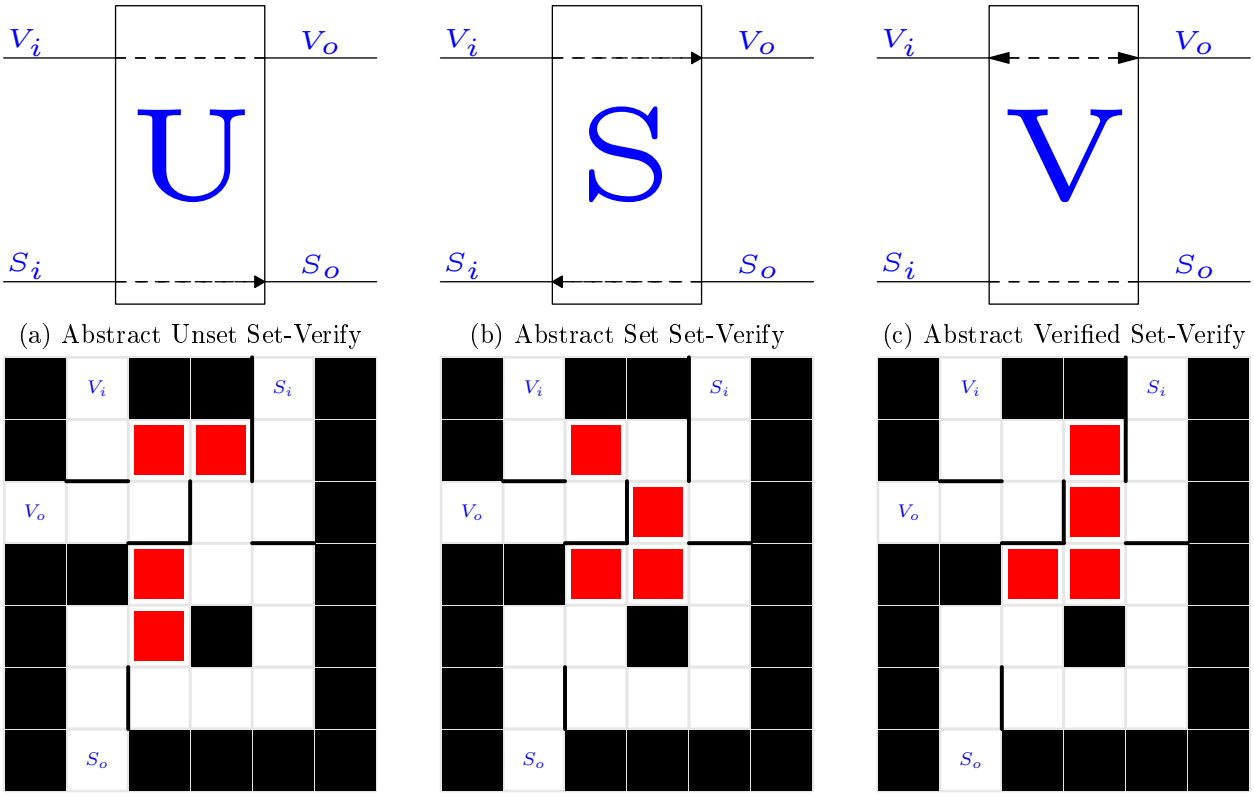
There are four possible states of the Set-Verify gadget: Broken, Unset, Set, and Verified. These are depicted in Figure 3-2. In the Broken state, the only possible transition is $S_o \to S_i$, changing the state to Unset. This state can only be reached from Unset, and allows strictly less future transitions than Set, which can also be reached from Unset, so we will disregard it. In the Unset state, the $S_i \to S_o$ transition is the only possibility, changing the state to Set. In the Set state, the $S_o \to S_i$ transition is possible, changing the state back to Unset, as well as the $V_i \to V_o$ transition, which changes the state to Verified. Finally, from the Verified state, the only transitions possible are $V_o \to V_i$, changing the state back to Set, and $V_i \to V_o$, leaving the state as Verified.

For the Set-Verify gadget in the Unset state, the S_i entrance is the only one which allows the robot to move any blocks. From the S_i entrance they can traverse to S_o, and they can also pull the middle block down behind them. Doing so will allow a traversal from V_i to V_o. To traverse back from S_o to S_i, the robot must first traverse back from V_o to V_i. Then, when the robot travels back from S_o to S_i, they must push the middle block back, ensuring the V_i to V_o traversal is impossible. Further, access to any sequence of entrances will not allow the robot to alter the system to allow traversals between the V_i and S_i entrances.

Since the Set-Verify gadget has no hallways with length greater than 3, any capabilities the robot may have of pushing or pulling more than one block at a time are irrelevant. Thus, the following proof will apply for all positive values of j and k in Push-j Pull-k.

3.1.3 Crossover Gadgets

In this section we build up the needed one use crossover gadget from a series of weaker types of crossover gadgets.



(a) Abstract Unset Set-Verify (b) Abstract Set Set-Verify (c) Abstract Verified Set-Verify

(d) Set-Verify, unset state (e) Set-Verify, set state (f) Set-Verify, verified state

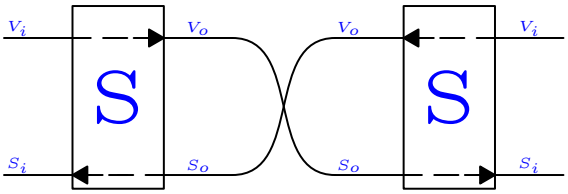
Figure 3-2: Set-Verify Gadgets

Directed Destructive Crossover

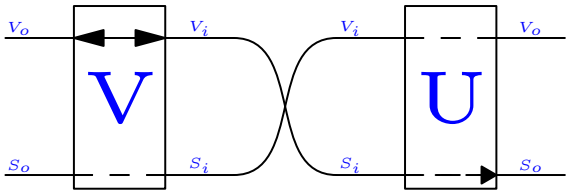
This gadget, depicted in Figure 3-3a, allows either a traversal from a to a' or b to b'. Once a traversal has occurred, that path may be traversed in reverse, but the other is impassable unless the original traversal is undone.

First, observe that transitions are initially only possible via the a and b entrances, since the transitions possible through a Set-Verify in the Set state can be entered through V_i and S_o, not S_i.

Assume without loss of generality that the gadget is entered at a. This changes the state of the left Set-Verify to Unset. At this point, only the right V_i and left S_i transitions are passable. Taking the S_i transition reverts all changes to the original state. Taking the V_i transition changes the right Set-Verify to Verified, and completes the crossover. At this point, the only possible transition is to undo the transition just made, from a' back to a, restoring the original state. Thus, the only transition possibilities are as stated above.

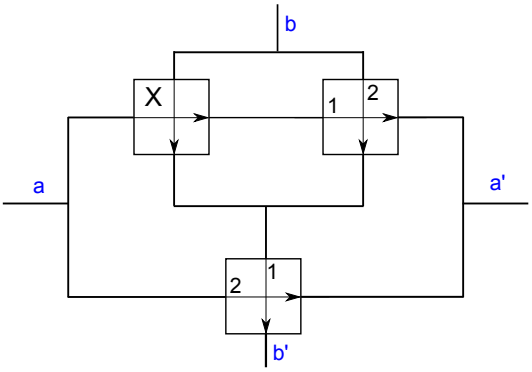


(a) The one-way destructive crossover constructed from two connected Set-Verify gadgets initialized in the set position.

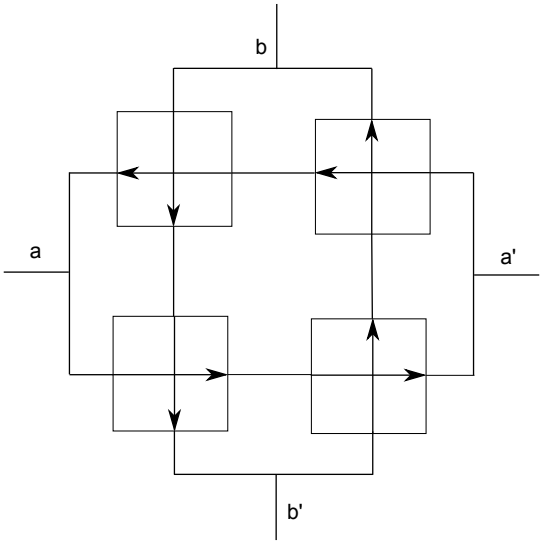


(b) The in-order one-way crossover constructed from two connected Set-Verify gadgets initialized in the verified and unset positions.

Figure 3-3: Two types of crossover gadgets



(a) The one use one-way crossover is constructed from a one-way destructive crossover and two in-order one-way crossovers.



(b) A full one use crossover constructed from four directed one use crossovers.

Figure 3-4: Composite crossover gadgets

In-order Directed Destructive Crossover

This gadget, depicted in Figure 3-3b allows a traversal from a to a', followed by a traversal from b to b'.

Initially, no entrance is passable except for a, since V_o is passable only in the Verified state, and S_o is passable only in the Set state. Once the left $V_o \to V_i$ transition is made, the robot has 2 options. It can either change the left Set-Verify gadget's state to Set, or leave it as Verified. In either case, the S_i entrance on that toggle is impassable, since a S_i entrance may only be traversed in the Unset state. The only transition possible on the right crossover is $S_i \to S_o$, changing the state from Unset to Set. This completes the first crossing.

Now, there are at most 2 transitions possible: from a' back to a, undoing the whole process, or entering at b. Note that entering at b is only possible if the left Set-Verify is in the Set state, so let us assume that state change occurred. In that case, the left $S_o \to S_i$ transition may be performed, changing the left Set-Verify's state to Unset. At that point, the only possible transitions are back to b, or through the right Set-Verify's $V_i \to V_o$ transition, completing the second crossover.

One Use Directed Crossover

The One Use Directed Crossover, depicted in Figure 3-4a, is the gadget needed for our proof. It allows a traversal from a to a' followed by a traversal from b to b', or from b to b' and then a to a'.

It is constructed out of an In-order Directed Crossover gadget and a Destructive Directed Crossover, as shown in Figure 3-4a. The a to a' traversal is initially passable, and goes through both gadgets, blocking the destructive crossover but leaving the in-order crossover open for the b to b' traversal. If the a to a' traversal does not occur, the b to b' traversal is possible via the destructive crossover.

One Use Crossover

Four Directed Crossovers can be combined, as shown below, to create a crossover that can be traversed in any direction. This is not necessary for our proof but is shown for general interest. Unfortunately, the inability to go through this gadget multiple times in the same direction without first going back through means it likely isn't suitable for a PSPACE-completeness reduction.

3.2 3D Push-Pull is NP-hard

In this section we prove that 3D Push-k Pull-l with fixed blocks is NP-hard, for all positive k and l. All of the hard work was done in the previous section. Here we will simply show how we can use the additional dimension to tweak the previous gadgets to build them without thin walls. We reduce from 3SAT, constructing our variables from chains of 3D Set-Verify gadgets, and our clauses from the verify side of the corresponding 3D Set-Verify gadget.

Theorem 3.2.

Proof. We follow the proof of Theorem 3.1 using a modified Set-Verify gadget, shown in Figure 3-5.

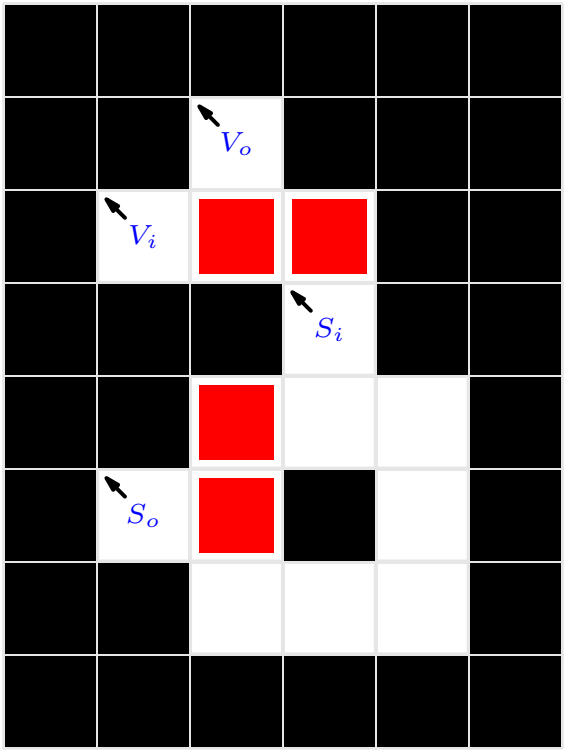


Figure 3-5: A Set-Verify gadget where the entrances and exits extend upward, notated by the diagonal arrows. This gadget is in the unset state.

As before, in the unset state the only possible traversal is S_i to S_0. This traversal allows the top right bock to be pulled down, moving the gadget into the set state. From here the V to V_0 traversal is possible, as well as going back through the S_0 to S pathway. However, the S to S_0 traversal is not possible. For more detail refer to Section 3.1.2. We do note that the cyclic ordering of the entrances in the 3D Set-Verify is different from that of the 2D Set-Verify, however this is not important as we no longer need to construct crossovers.

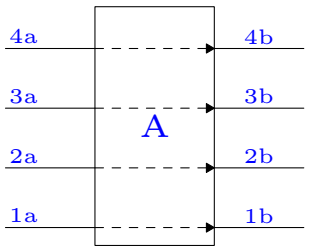
Variables are composed of hallways of 3D Set-Verify gadgets connected S_0 to S, one for each clause in which the variable appears, as in Figure 3-1a. Clauses are composed of three 3D Set-Verify gadgets connected in parallel as in Figure 3-1b. The details of these constructions follow those in Section 3.1.1 This completes the reduction from 3SAT. In addition, we note that all blocks are in hallways of length at most 3, thus the gadgets still function as described for any positive push and pull values. □

3.3 PSPACE

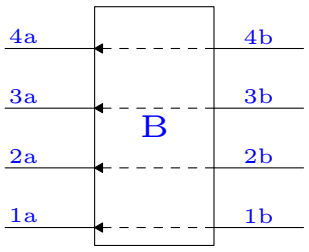
In this section we show the PSPACE-completeness of 3D push-pull puzzles with equal push and pull strength. We introduce a gadget called the 4-toggle and use it to simulate Quantified Boolean Formulas [GJ79]. We construct the 4-toggle gadget in 3D push-pull block puzzles, completing the reduction. In particular we prove 3D Push1-Pull1 with thin walls is PSPACE-complete and 3D Push$i$-Pull$j$, for all positive $i = j$, is PSPACE-complete. A gap between NP and PSPACE still remains for 3D puzzles with different pull and push values, as well as 2D puzzles.

3.3.1 Toggles

We define an n-toggle to be a gadget which has n internal pathways and can be in one of two internal states, A or B. Each pathway has a side labeled A and another labeled B. When the toggle is in the A state, the pathways can only be traversed from A to B and similarly in the B state they can only be traversed from B to A. Whenever a pathway is traversed, the state of the toggle flips.



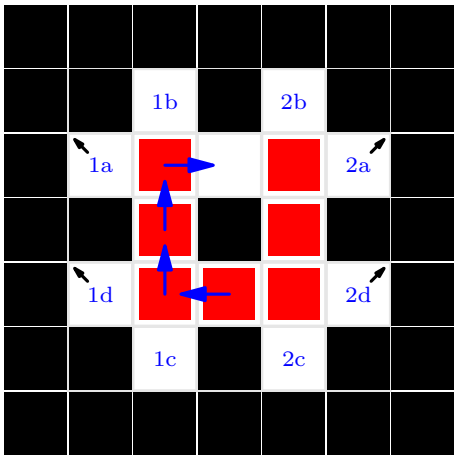
(a) 4-Toggle in state A. Here we can only follow the paths from left to right.



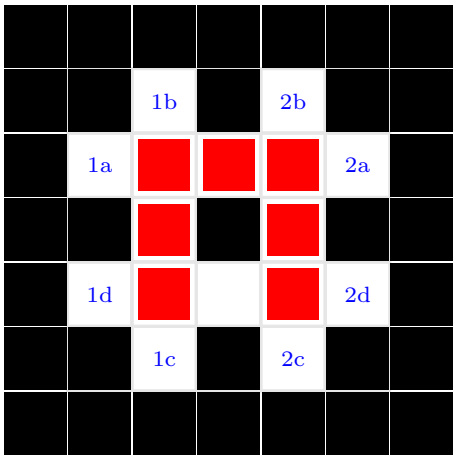
(b) 4-Toggle in state B. The direction of the paths through the toggle are flipped.

Figure 3-6

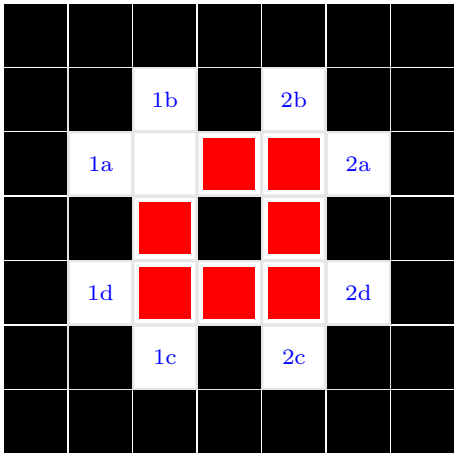
Figure 3-7a acts as a 2-toggle. The locations $1a$, $1d$, $2a$, and $2d$, are all entrances and exits to the 2-toggle, while $1b$ connects directly to $1c$, and $2b$ connects directly to $2c$. Notice that there is a single block missing from the ring of eight blocks. When the missing block is on top, as diagrammed, it will represent state A, and when it is on the opposite side, we call it state B. Notice that in state A, it is impossible to enter through entries $1d$ or $2d$. When we enter in the $1a$ or $2a$ sides, we can follow the moves in the series of diagrams to exit the corresponding $1b$ or $2b$ side, leaving the gadget in the B state. One can easily check that the gadget can only be left in either state A, B, or a broken state as seen in Figure 3-7c. Notice that in the broken state, every pathway except the one just exited is blocked. If we enter through that path, it is in exactly the same state as if it had been



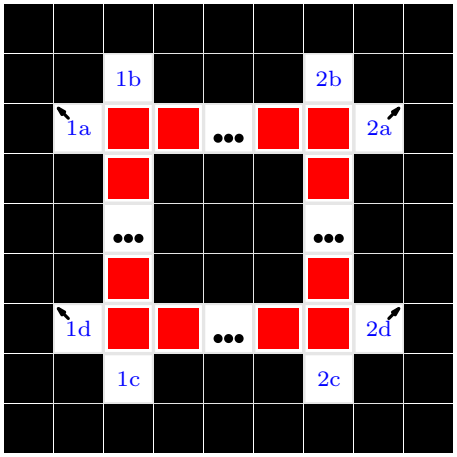
(a) 2-Toggle in state A. The arrows indicate the transition to state B.



(b) 2-Toggle in state B.



(c) 2-Toggle in one of four broken states.

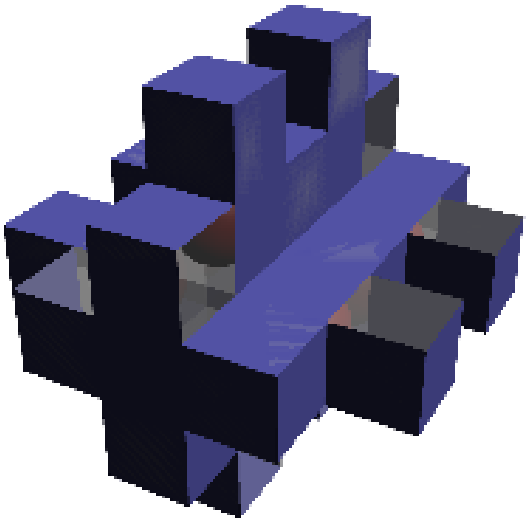


(d) Construction of a 2-toggle when the robot can push or pull multiple blocks.

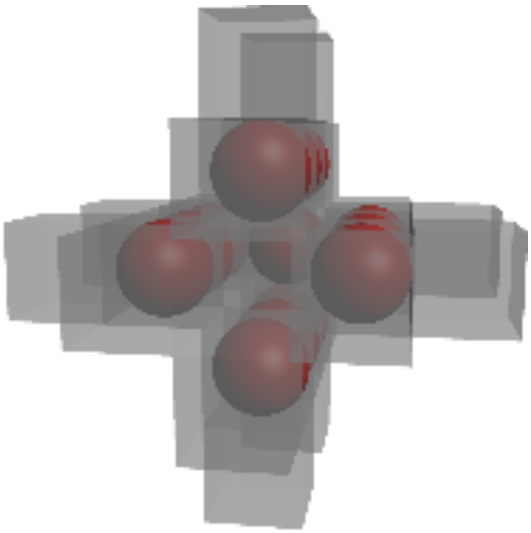
Figure 3-7: 2-Toggles constructed in a push-pull block puzzle.

in an allowed state and entered through the corresponding pathway normally. For example, in the diagram one can only enter through $1b$ and after doing so it is the same as entering in path $1b$ on a 2-toggle in state B. Thus the broken state is never more useful for solving the puzzle and can be safely ignored.

To construct a 4-toggle we essentially take two copies of the 2-toggle, rotate them perpendicular to each other in 3D, and let them overlap on the central axis. See Figure 3-9a. We still interpret the lack of blocks in the same positions as in the 2-toggle as states A or B. Now we have four different paths which function the same as the ones described above. Similar arguments show the broken states of the 4-toggle also don't matter.



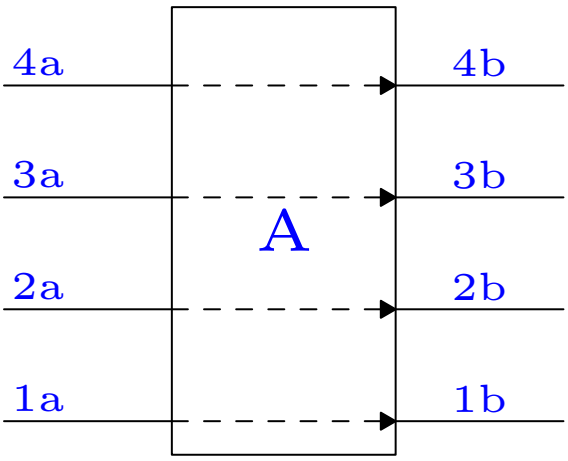
(a) Diagram of a 4-toggle showing impassible surfaces.



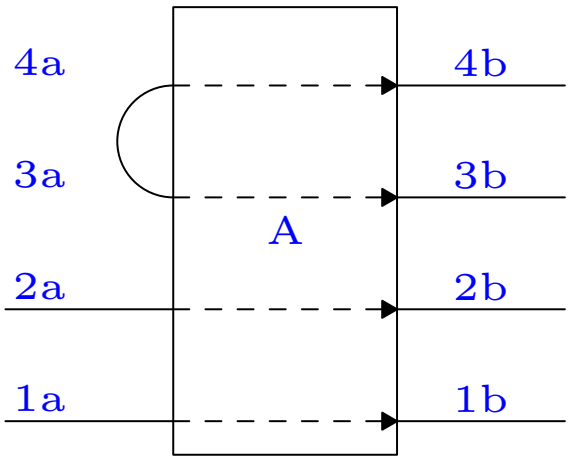
(b) Diagram of the internals of a 4-toggle.

Figure 3-8

3.4 Locks



(a) Representation of a 4-toggle in state A.



(b) Diagram of a lock. The $1a$ to $2a$ traversal is only possible in state A and returns the toggle to state A.

Figure 3-9

A lock is a gadget consisting of a 2-toggle and a separate pathway. Traversing the separate pathway can only be done in a single direction, which is dependent on whether the 2-toggle is in state A or B, and the traversal does not change the internal state of the 2-toggle. The 2-toggle functions exactly as described above. This gadget can be implemented using a 4-toggle by connecting the $3B$ and

$4B$ entrances of the 4-toggle with an additional corridor, as shown in 3-9b. Traversing the resultant full pathway, from $3A$ to $3B$ to $4B$ to $4A$, is possible only if the initial state of the 4-toggle is A, and will leave the 4-toggle in state A. In addition, a partial traversal, such as from $3A$ to $3B$ and back to $3A$, does not change the internal state. The two unaffected pathways of the toggle, 1 and 2, continue to function as a 2-toggle.

A synchronized lock block is a gadget consisting of a 2-toggle and any number of separate pathways. As in the lock, the 2-toggle functions as described above. Each pathway may be set up to be passable in either direction if the toggle is in state A and impassable in either direction if the toggle is in state B (type A), or passable in either direction if the toggle is in state B and impassable in either direction if the toggle is in state A (type B). This is implemented using one lock per non-toggling pathway needed. As shown in Figure 3-10, there are 2 pathways of the entire synchronized lock block system: the 1 pathway, and the 2 pathway. A lock whose pathway will create a type A synchronized lock pathway is placed so that its pathways are in the same direction as the synchronized lock block's pathways, A entrance in the A direction and B entrance in the B direction. Its state matches the synchronized lock block's state. A type B lock is placed in the opposite direction, with its B entrance in the synchronized lock block's A direction and A entrance in the B direction. Its state is also opposite the synchronized lock block's state.

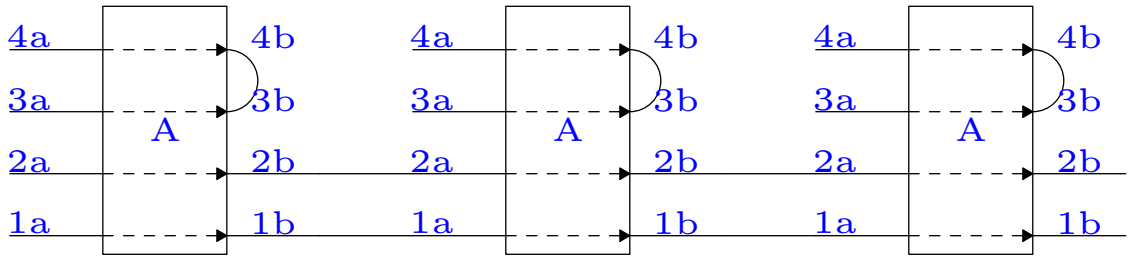


Figure 3-10: A synchronized lock block chain.

When the synchronized lock block is traversed, all of the internal locks' states flip, rendering the synchronized lock block passable in the opposite direction, and switching the passability and impassibility of all of the external pathways.

3.4.1 Binary Counter

Universal quantifiers must iterate through all possible combinations of values that they can take. In this section we construct a gadget that runs though all the states of its subcomponents as the robot

progresses through the gadget. This construction will serve as the base for our universal quantifiers.

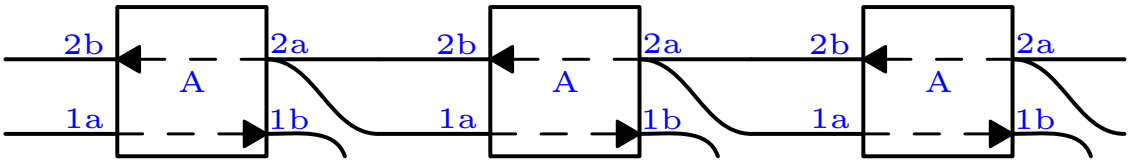


Figure 3-11: The central portion of a three bit binary counter made from 2-toggles.

We now define a binary counter. The binary counter has a fixed number of internal bits. Whenever the binary counter is traversed in the forwards direction, the binary number formed by the internal bits increases by one and the robot leaves via one of the exits. If the binary counter is traversed in the reverse direction, the internal value is reduced by one. If the binary counter is partial traversed, but then the robot leaves via its initial entrance, the internal value does not change.

The binary counter is implemented as a series of 2-toggles, as shown in Figure 3-11. The entrance pathway is connected to the 2-toggle's $1A$ and $2B$ entrances. The $1B$ exit from the 2-toggle will exit from the entire binary counter. The $2A$ exit will continue on to the next 2-toggle, attaching to that toggle's $1A$ and $2B$ entrances. This will continue for every toggle down the line, except that the last toggle's $2A$ exit signals an overflow and exits from the counter.

To see that this produces the desired effect, identify a toggle in state A as a 0 bit, and a toggle in state B as a 1 bit. Let the entrance toggle's bit be the least significant bit, and the final toggle be the most significant. When the robot enters the binary counter in the forwards direction, it will flip the state of every toggle it passes through. When it enters a toggle that is initially in state B, and thus whose bit is 1, it will flip the state/bit and proceed to the next toggle, via the $2B - 2A$ pathway. When it encounters a toggle that is initial in state A / bit 0, it will flip the state/bit and exit via the $1A - 1B$ pathway. Thus, the overall effect on the bits of the binary counter is to change a sequence of bits ending at the least significant bit from 01..11 to 10..00. This has the effect of increasing the value of the binary counter by one.

If the robot approaches this apparatus from the exit side, there are three possibilities. If the robot attempts to enter via a $1B$ pathway whose toggle is in state A / 0, the toggle is impassable and the robot makes no progress. If the robot enters via a $1B$ pathway whose toggle is in state B / 1, after traversal the robot will have 2 options: To return in the direction it came, or to continue through the next toggle's $2A - 2B$ pathway. The latter is only possible if the next toggle (the one just less significant than the entrance) is in state A / 0. Upon traversing that toggle, the robot will again

be able to return from where it came, or progress in the same fashion if the next toggle is in state A / 0. This will continue until the robot either turns back, or exits out the main entrance.

Thus, if and only if the robot enters via the least significant toggle which is set to 1, the robot will be able to leave out the main entrance. If the robot enters any other way, it will be forced to return via the path it entered by, and undo all changes it has made.

The transformation on the bits caused by the only possible successful reverse traversal is to change 10..00 to 01..11, resulting in a decrement operation, as desired.

3.4.2 Existential Quantifiers

We now define an existential gadget. An existential gadget is like a synchronized lock block, except that instead of a 2-toggle, it has a single pathway which is always passable in both directions, and upon traversing the pathway the robot may or may not change the internal state of the synchronized lock block, as it chooses. The variable is considered true if the 4-toggles in the lock block are in state A and false if they are in state B.

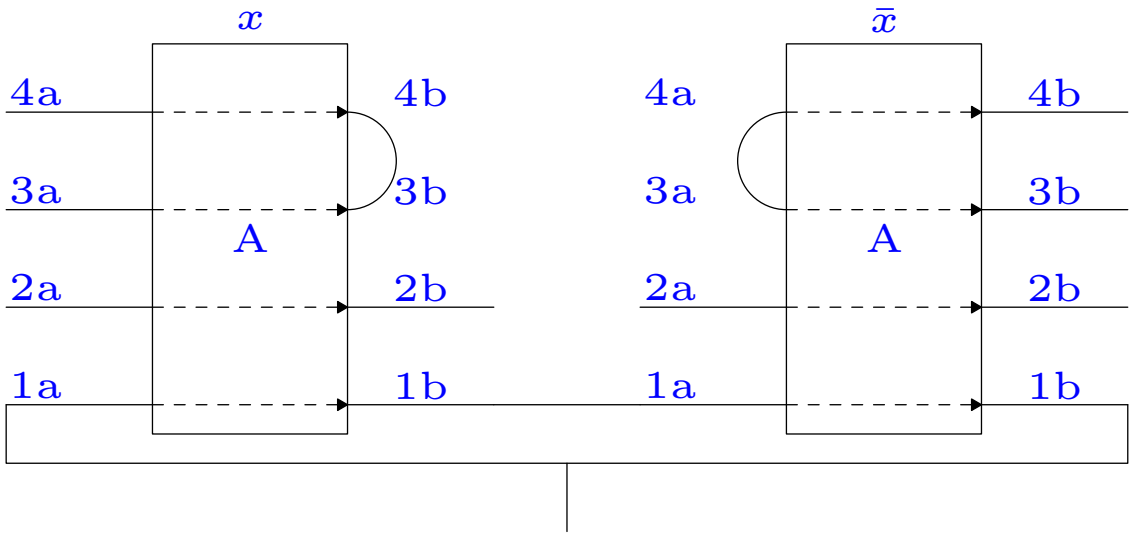


Figure 3-12: An existential gadget is simply a lock block with a number of copies equal to the number of times the corresponding variable occurs in the formula.

As shown in Figure 3-12, an existential gadget consists of a synchronized lock block and a pathway with access to all four pathways of the 2-toggle component of the synchronized lock block. Upon traversing the main pathway, the robot may choose to traverse the 2-toggle any number of times, leaving it in either state, as desired.

3.4.3 Quantifier Chain

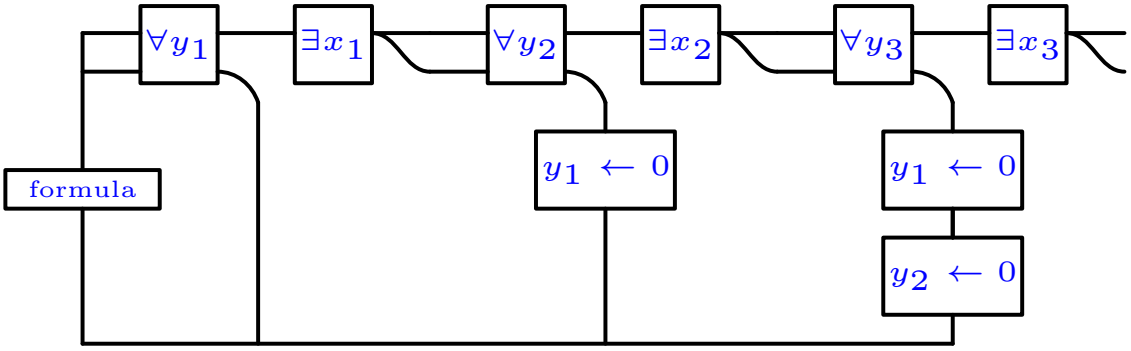


Figure 3-13: A segment of the quantifier chain. Each lock is actually a synchronized lock chain with length equal to the number of times the corresponding variable appears in the formula.

We now define a quantifier chain, as shown in Figure 3-13. A quantifier chain implements a series of alternating existential and universal variables, as well as external literal pathways, which may be traversed if and only if their corresponding variables are set to a pre-specified value.

Traversing the quantifier chain repeatedly in the primary direction will cycle the universal variables through all 2^n possible settings. Upon each traversal, an initial sequence of the universal variables will have their values flipped. During the traversal, the robot will have the option to set a series of corresponding existential variables to whatever value they wish. These comprise the existentials nested within the universal variables whose values were flipped.

Traversing the quantifier chain in the reverse direction is only possible if the robot enters via the lowest order universal toggle whose setting is 1. The traversal will go back one setting in the sequence of possible settings of the universal variables, and allow the robot to set all existential variables corresponding to altered universal variables arbitrarily. No other existential variables can be changed.

There is also a special exit, the overflow exit, which can only be reached after all of the universal variable settings have been traversed. This is the target location for the robot.

A quantifier chain is implemented much like a binary counter, with some additions. Every universal variable will be represented by a synchronized lock block, where each individual lock will serve as a literal. The 2-toggles which are governing the progression through the synchronized lock blocks are hooked up in the same manner as the 2-toggles in a binary counter gadget. This forces the synchronized lock blocks to be set to the corresponding values in the simulated binary counter.

The next addition is the existential variables, which consist of existential gadgets placed just after the $2A$ exits of each universal variable, and just before the $1A$ and $2B$ entrances of the the next universal variable, as shown in Figure 3-13.

One potential flaw in the apparatus as described so far is that a robot could enter via an exit corresponding to a highly significant universal variable set to 1, alter its paired existential variable, and then leave via the original entrance. This must not happen for the existential counter to work properly, so a series of lock-chain pathways are added to each of the exits from the quantifier. Recall from the description of the binary counter that the only possible reverse traversal of the counter should be via entering at the lowest significance variable set to 1. To prevent entry at higher-significance variables set to 1, we will add a series of locks to that exit which are only passable if all lower-significance universal variables are set to 0.

This prevents the undesired high-significance existential alteration problem mentioned above, as it is now impossible to enter the gadget via the higher-significance universal variables which are set to 1, since that entryway will have at least 1 closed lock on it and thus be impassable. However, this addition does not affect the desired forward or backward transition, because the entrance/exit in question will have all of its synchronized lock block external pathways in the passable state.

3.4.4 Clause Gadget

We construct a clause gadget by putting the lock pathways of the three 4-toggles in the lock chains representing the corresponding variables in parallel, as we did with Set-Verify gadgets in Figure 3-1b. Each of these paths can be traversed only if the corresponding variable has been set. Since they are in parallel, only one needs to be passable for the robot to be able to continue on to the next clause.

3.4.5 Beginning and End Conditions

The overall progression of the robot through the puzzle starts with the quantifier chain. The robot increments the universal variables and sets the appropriate existential variables arbitrarily, then traverses the formula gadget to verify that the formula is true under that setting. The process then repeats.

At the beginning of this procedure, the robot must be allowed to set all of the existential variables arbitrarily. To ensure this, we will set up the quantifier gadget in the state 01..11, with all variables set to 1 except the highest order one. The highest order variable will be special, and will not be used in the $3CNF$ formula. The initial position of the robot will be at the entrance to the quantifier gadget. This will allow the robot to flip every universal in the quantifier gadget, from 01..11 to 10..00, and accordingly set every existential variable arbitrarily. To force the robot to go forward through the quantifier gadget instead of going backwards through the clause chain, we will add a literal onto the end of the formula gadget which is passable if and only if the highest order variable is set to 1.

After this set up, the robot will progress through the loop consisting of the quantifier gadget and the formula gadget, demonstrating the appropriate existential settings for each assignment of the universal quantifiers.

After progressing through every possible state of the universal quantifiers, the universals will be in the state 11..11. At this point, the robot may progress through the quantifier gadget and exit via its special pathway, the carry pathway of the highest order bit. This special pathway will lead to the goal location of the puzzle. Thus, only by traversing the quantifier - formula loop repeatedly, and demonstrating the solution to the TQBF problem, will the robot be able to reach the goal. The robot may reach the goal if and only if the corresponding quantified boolean formula is true.

Chapter 4

Conclusion

In this thesis we proved a number of hardness results about single-agent games. The results themselves are obviously of interest to game and puzzle enthusiasts. We also hope the study of motion planning in environments with dynamic topologies leads to new insights. What we consider most interesting are the techniques developed to solve these problems. Looking back, many of the Portal proofs could have been phrased in terms of the style of agent-gadget diagrams seen in the Push-Pull blocks section. Adding new, simple gadgets to this collection of abstractions gives us powerful new tools with which to attack future problems. We also believe the decomposition of games into individual mechanics will be an important tactic for understanding games of increasing complexity.

4.1 Open Questions

This work leads to many open questions to pursue in future research. Many are directly related to the specific games studied and others more broadly applicable. For Push-Pull block puzzles, we leave a number of NP to PSPACE gaps, as is common for many other variations of the problem. One would hope to directly improve upon the results here to show tight hardness results for 2D and 3D push-pull block puzzles. One might also wonder if the gadgets used, or the introduction of thin walls might lead to stronger results for other block pushing puzzles. We also leave open the question of push-pull block puzzles without fixed walls. A single 3×3 block of clear space allows the robot to reach any point, making gadget creation challenging.

There are also interesting questions with regard to the abstract gadgets used in the proof. Are

2-toggles or 3-toggles sufficient to prove NP-hardness or PSPACE-hardness? Can one construct crossovers out of toggles? Are Set-Verify gadgets sufficient for PSPACE-hardness? Is this model of abstraction useful for capturing more agent-based games and puzzles?

In Portal, we leave many hardness gaps and a number of mechanics unexplored. We are particularly curious about Portal with only portals, and Portal with only cubes. The removal of Emancipation Fields from our proofs would be very satisfying. We had hoped push-pull block puzzles would lead to a resolution of Portal with only cubes, however the models proved to be too different. The other major introduction in Portal 2 that was ignored was co-op mode. If the players are free to communicate and have perfect information of the map, this likely won't add much to the complexity of the game. However, a cooperative game with imperfect information could be 2EXPTIME-complete [PRA01].

Finally, one would hope to use these techniques to show hardness for other problems. The toggle and Set-Verify abstractions add new, simple gadgets for proving hardness. Many other games use movable blocks, timed door buttons, and stationary turrets and may have hardness results that immediately follow. Some techniques like encoding numbers in velocities might be transferable. It would be good to generalize some of these into meta-theorems which cover a larger variety of games.

Bibliography

[ACJ+10] David Arthur, Raphaël Clifford, Markus Jalsenius, Ashley Montanaro, and Benjamin Sach. The complexity of flood filling games. In Paolo Boldi and Luisa Gargano, editors, *Fun with Algorithms*, volume 6099 of *Lecture Notes in Computer Science*, pages 307–318. Springer Berlin Heidelberg, 2010.

[ADGV14] Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (NP-)hard. In *Proceedings of the 7th International Conference on Fun with Algorithms (FUN 2014)*, Lipari Island, Italy, July 1–3 2014.

[BDD+02] Therese C. Biedl, Erik D. Demaine, Martin L. Demaine, Rudolf Fleischer, Lars Jacobsen, and J. Ian Munro. The complexity of Clickomania. In R. J. Nowakowski, editor, *More Games of No Chance*, pages 389–404. Cambridge University Press, 2002. Collection of papers from the MSRI Combinatorial Game Theory Research Workshop, Berkeley, California, July 24–28, 2000.

[ben14] bennyscube. YouTube channel. `https://www.youtube.com/user/bennyscube/featured`, May 2014.

[Cao] Eric Caoili. Portal 2 has sold over 4m copies. `http://www.gamasutra.com/view/news/169967/Portal_2_has_sold_over_4M_copies.php`. Accessed: 2015-08-21.

[Col13] Andrew M Colman. *Game theory and its applications: in the social and biological sciences*. Psychology Press, 2013.

[Con01] John H. Conway. *On numbers and games (2nd ed.)*. A K Peters, 2001.

[Cor04] G. Cormode. The hardness of the Lemmings game, or oh no, more NP-completeness proofs. In *Proceedings of Third International Conference on Fun with Algorithms*, pages 65–76, 2004.

[Cul98] J. C. Culberson. Sokoban is PSPACE-complete. In *Proceedings International Conference on Fun with Algorithms (FUN98)*, pages 65–76, Waterloo, Ontario, Canada, June 1998. Carleton Scientific.

[DDO00] Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *Proceedings of the 12th Annual Canadian Conference on Computational Geometry (CCCG 2000)*, pages 211–219, Fredericton, New Brunswick, Canada, August 16–18 2000.

[DH01] Erik D. Demaine and Michael Hoffmann. Pushing blocks is NP-complete for noncrossing solution paths. In *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG 2001)*, pages 65–68, Waterloo, Ontario, Canada, August 13–15 2001.

[DH09] Erik D. Demaine and Robert A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In Michael H. Albert and Richard J. Nowakowski, editors, *Games of No Chance 3*, volume 56 of *Mathematical Sciences Research Institute Publications*, pages 3–56. Cambridge University Press, 2009.

[DHH02] Erik D. Demaine, Robert A. Hearn, and Michael Hoffmann. Push-2-F is PSPACE-complete'. In *Proceedings of the 14th Canadian Conference on Computational Geometry (CCCG 2002)*, pages 31–35, Lethbridge, Alberta, Canada, August 12–14 2002.

[DHH04] Erik D. Demaine, Michael Hoffmann, and Markus Holzer. PushPush-k is PSPACE-complete. In *Proceedings of the 3rd International Conference on Fun with Algorithms (FUN 2004)*, pages 159–170, Isola d'Elba, Italy, May 26–28 2004.

[DHLN03] Erik D. Demaine, Susan Hohenberger, and David Liben-Nowell. Tetris is hard, even to approximate. In *Proceedings of the 9th International Computing and Combinatorics Conference (COCOON 2003)*, pages 351–363, Big Sky, Montana, July 25–28 2003.

[DO92] A. Dhagat and J. O'Rourke. Motion planning amidst movable square blocks. In *Proceedings of the 4th Canadian Conference on Computational Geometry (CCCG 1992)*, 1992.

[dVV03] Sven de Vries and Rakesh V. Vohra. Combinatorial auctions: A survey. *INFORMS J. on Computing*, 15(3):284–309, July 2003.

[DZ96] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4), 1996.

[FB02] Gary William Flake and Eric B. Baum. Rush Hour is PSPACE-complete, or why you should generously tip parking lot attendants. *Theoretical Computer Science*, 270(1-2):895 – 911, 2002.

[For10] Michal Forisek. Computational complexity of two-dimensional platform games. In *Proceedings International Conference on Fun with Algorithms (FUN 2010)*, pages 214–227, 2010.

[Fri] Erich Friedman. Pushing blocks in gravity is NP-hard. `http://www2.stetson.edu/~efriedma/papers/gravity.pdf`.

[GJ79] 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, 1979.

[Gru39] Patrick M Grundy. Mathematics and games. *Eureka*, 2(6-8):21, 1939.

[HD05] Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1):72–96, 2005.

[HD09] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A. K. Peters, Ltd., Natick, MA, USA, 2009.

[Hof00] M. Hoffman. Push-* is NP-hard. In *Proceedings of the 12th Canadian Conference on Computational Geometry (CCCG 2000)*, Lethbridge, Alberta, Canada, 2000.

[IPS82] Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[Joh12] Nathaniel Johnston. The complexity of the puzzles of Final Fantasy XIII-2. arXiv:1203.1633, 2012.

[Kay00] Richard Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.

[KPS08] Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. A survey of NP-complete puzzles. *ICGA Journal*, 31(1):13–34, 2008.

[Pap01] Christos H. Papadimitriou. Algorithms, games, and the internet. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 749–753, New York, NY, USA, 2001.

[PRA01] Gary Peterson, John Reif, and Salman Azhar. Lower bounds for multiplayer noncooperative games of incomplete information. *Computers & Mathematics with Applications*, 41(7):957–992, 2001.

[Ren] Paul Rendell. A Turing machine in Conway's game life. `http://www.cs.unibo.it/~babaoglu/courses/cas00-01/papers/Cellular_Automata/Turing-Machine-Life.pdf`.

[Rit10] Marcus Ritt. Motion planning with pull moves. arXiv:1008.2952, 2010.

[RW86] Daniel Ratner and Manfred K Warmuth. Finding a shortest solution for the n× n extension of the 15-PUZZLE is intractable. In *AAAI*, pages 168–172, 1986.

[Sch78] Thomas J. Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978.

[Spr35] Richard Sprague. Über mathematische Kampfspiele. *Tôhoku Math. J*, 41:438–444, 1935.

[stea] steamspy. App data Portal. `http://steamspy.com/app/400`. Accessed: 2015-08-22.

[steb] steamspy. App data Portal 2. `http://steamspy.com/app/620`. Accessed: 2015-08-22.

[Vig12] Giovanni Viglietta. Gaming is a hard job, but someone has to do it! arXiv:1201.4995, 2012.

[Vig15] Giovanni Viglietta. Lemmings is PSPACE-complete. *Theoretical Computer Science*, 586:120–134, 2015.

[Wal14] Toby Walsh. Candy Crush is NP-hard. arXiv:1403.1911, 2014.

[Wil91] Gordon Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 3(1):131–150, 1991.

[YP] Wesley Yin-Poole. Portal sells nearly four million. `http://www.eurogamer.net/articles/2011-04-20-portal-sells-nearly-four-million`. Accessed: 2015-08-21.