On the Computational Complexity of Doors

Bachelor's Thesis of

Oğuz Mutlu

At the Department of Informatics
Institute of Theoretical Informatics (ITI)

Reviewer:	T.T.-Prof. Dr. Thomas Bläsius
Second reviewer:	PD Dr. Torsten Ueckerdt
Advisor:	Jean-Pierre von der Heydt

01.12.2023 – 31.03.2024

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, **31.03.2024**

. . . .

(Oğuz Mutlu)

Abstract

Puzzles and games are widely studied in theoretical computer science in terms of their computational complexity. In a similar vein, this thesis is devoted to studying the complexity of some puzzle-like problems involving doors. Although rare in everyday life, doors can cause problems by getting in the way of each other if they are placed in the wrong way. We build on this idea by formalizing the problem of determining whether doors can be placed at designated locations on a given two-dimensional floor plan under certain constraints. We show that door placement problems are in **P** or **NP**-complete, depending on the number of allowed ways to place each of the given doors. Similarly, we consider door replacement problems where, given a placement of doors, the problem consists in determining whether one can change the orientation of a door by a sequence of allowed moves. We show that door replacement problems are in **P** or **PSPACE**-complete, depending on the same dichotomy. Furthermore, we show **NP**-completeness for some versions of motion planning on a floor with doors and walls. Finally, we prove that a variant of the door placement problem, where one can continuously shift and rotate doors, is $\exists\mathbb{R}$-complete.

Zusammenfassung

Puzzles und Spiele werden in der theoretischen Informatik im Hinblick auf ihre Komplexität häufig untersucht. In ähnlicher Weise widmet sich diese Arbeit der Untersuchung der Komplexität einiger puzzle-ähnlichen Probleme mit Türen. Obwohl es im Alltag selten vorkommt, können Türen Probleme verursachen, indem sie sich gegenseitig blockieren, wenn sie falsch platziert sind. Wir bauen auf dieser Idee auf, indem wir das Entscheidungsproblem formalisieren, ob Türen unter bestimmten Einschränkungen an bestimmten Stellen auf einem gegebenen zweidimensionalen Grundriss platziert werden können. Wir zeigen, dass Türplatzierungsprobleme in **P** oder **NP**-vollständig sind, abhängig von der Anzahl der zulässigen Möglichkeiten, die einzelnen Türen zu platzieren. In ähnlicher Weise betrachten wir Türumplatzierungsprobleme, bei denen das Problem bei gegebener Türanordnung darin besteht, festzustellen, ob die Ausrichtung einer Tür durch eine Folge zulässiger Züge geändert werden kann. Wir beweisen, dass die Türumplatzierungsprobleme in Abhängigkeit von derselben Dichotomie in **P** oder **PSPACE**-vollständig sind. Darüber hinaus zeigen wir die **NP**-Vollständigkeit für einige Versionen der Bewegungsplanung auf einem Grundriss mit Türen und Wänden. Schließlich beweisen wir, dass eine Variante des Türplatzierungsproblems, bei der man Türen stetig verschieben und drehen kann, $\exists\mathbb{R}$-vollständig ist.

Contents

1 Introduction

People have always been interested in puzzles and games. Games have also been the subject of much research in theoretical computer science. In recent decades, many classic games have been analyzed in the literature in terms of their computational complexity. There are some conferences such as the *International Conference on FUN with Algorithms* where, among others, results on the computational complexity of puzzles, games and interesting problems from everyday life have been published.

In this thesis, we want to consider the computational complexity of *doors*. Doors are ubiquitous in everyday life. As geometric objects, they are simple enough to formalize and lead to many interesting problems. For example, consider the scenario where you are in a building and you have the floor plan. You want to get to a certain place, but there are a lot of doors to open along the way. How hard is the problem of determining whether you can get to the place you want? Or suppose you are an architect and you want to place doors on a floor in such a way that they do not block each other. Is such a placement possible under certain constraints?

As our contribution, we formalize and answer such questions throughout this thesis.

In defining when doors interfere with each other, we are interested in the area that a door can possibly occupy, so we formalize doors as quarter circles on a floor plan and say that they interfere with each other when these quarter circle areas intersect. With the help of this formalization, we define the door placement problem which asks, given threshold locations for doors, whether a door can be placed on every threshold without interference. We show that this problem is **NP**-complete. In addition, we realize that the same problem becomes tractable when we consider doors with square opening ranges, as opposed to the quarter-circle opening ranges of conventional doors. In another variant, this problem becomes $\exists\mathbb{R}$-complete if we consider doors with triangle opening ranges and we allow to shift and rotate them by real number amounts in their respective ranges.

Another problem we consider is what we call the door replacement problem where we are given a floor plan with doors already placed and our goal is to reverse the opening direction of a single door by a sequence of moves changing door orientations during which we have to maintain an interference-free arrangement at each step. We prove that this problem is **PSPACE**-complete for conventional doors, but can be solved in polynomial time if we consider doors that open like a square.

Lastly, we consider motion planning problems and prove **NP**-completeness for deciding whether two designated locations on a given floor are reachable from each other by an agent that is allowed to open (but not close) doors, both in the case where the agent is only allowed to push doors and in the case where both pushing and pulling are allowed.

1.1 Related Work

There are many puzzles and games whose computational complexity has been analyzed in the literature. Two survey papers on this topic are [DH08] and [UEH23]. The lectures by Erik D. Demaine at MIT [Dem23] also present many hardness proofs about games and puzzles.

Some examples are the **PSPACE**-completeness results for the Sokoban puzzle [Cul97], where the player (a warehouse keeper) pushes boxes around trying to store them at designated locations as well as the generalized version of the children's puzzle Rush Hour [FB02], where the player slides rectangular cars trying to move a given car to the exit.

Several motion planning puzzles resembling Sokoban have been considered in the literature. These include pushing blocks to reach a target location [DDHO03], one type of which we also present in Chapter 5, and pulling blocks [PRB16]. The paper [Ani+20] proposes a so-called walking through doors framework for motion planning problems to determine which types of gadgets suffice to prove hardness of problems. However, the term *door* is used for gadgets with multiple tunnels and are different from the doors we consider in this thesis. In [Gre+21], motion planning through turnstiles is considered.

In [HD05] a model of computation called Nondeterministic Constraint Logic is proposed, which is useful for proving hardness of puzzles which contain some kind of reconfiguration element. We present this model of computation in Chapter 4, as we reduce it to our door replacement problem.

An **NP**-hard geometric problem is drawing different-sized discs at certain points on a map such that the visible boundary of all objects are maximized [CHKS10]. Our door placement problems in Chapter 3 are somewhat similar to this problem.

In [AMS22], packing problems with different types of polygons are considered and proven to be $\exists\mathbb{R}$-complete. In our placement problem in Chapter 6, we are allowed to shift and rotate triangle shaped doors in their respective ranges to achieve a pairwise interior-disjoint arrangement. This is somewhat similar to the problem of packing triangles inside a rectangular region.

1.2 Outline

In Chapter 2, we formalize different types of doors and problems. In addition, we give an overview of the complexity classes we deal with.

In Chapter 3, we deal with door placement problems. These are problems where we are given a floor plan and a designated location (door threshold) for each door that the door should occupy when closed. The question is whether we can determine an orientation for each door such that no doors get potentially in the way of each other. We distinguish between squared and circular doors. The opening range of the former is a square, while the latter has a quarter-circle opening range. If the doors are squared, there are two ways of placing each door. One can only choose the opening side. If the doors are circular (i.e. doors we see in real life) there are four ways to place each door. One can choose the opening side and the side of the hinge.

In Chapter 4, we deal with door replacement problems. In these problems, we are given a floor plan where every door is already placed and we try to reverse the opening direction of a single door by a sequence of moves, where each move consists in changing the orientation of a single door resulting in a legal placement. The following table gives an overview of our results for placement and replacement problems.

	Placement	Replacement
Squared	$\in$ **P**	$\in$ **P**
Circular	**NP**-complete	**PSPACE**-complete

In Chapter 5, we consider motion planning problems involving doors. We define two degrees of freedom when considering different variants. In a given problem, the agent is either only allowed to open (and not close) doors, or the agent is allowed to open and close doors arbitrarily many times. Furthermore, the agent is allowed to do one of the following three things: only push, only pull, or both push and pull. The following table summarizes our results, where we do not have any hardness results for the problem Pull-Open-Doors as well as all the Open/Close problems. In particular, the containment of Open/Close problems in **PSPACE** is probably not tight, in the sense that any of those problems can turn out to be in **NP** or even **P**.

	Push	Pull	Push/Pull
Open	**NP**-complete	$\in$ **NP**	**NP**-complete
Open/Close	$\in$ **PSPACE**	$\in$ **PSPACE**	$\in$ **PSPACE**

In Chapter 6, we consider a variant of the placement problem where doors can be shifted or rotated by real number amounts under given constraints. Our results are summarized as below, where it is open whether the problems with squared and circular doors are also hard for $\exists\mathbb{R}$.

	Real Placement
Squared	$\in \exists\mathbb{R}$
Circular	$\in \exists\mathbb{R}$
Triangular	$\exists\mathbb{R}$-complete

2 Preliminaries

In this thesis, we consider *doors* from a top-down perceptive as they are drawn on a two-dimensional floor plan and we are interested in the area a given door can possibly occupy, with the assumption that a door can open at most 90 degrees. So, we define a **(circular) door** as a quarter circle which consists of two perpendicular line segments, an arc, and the interior area enclosed by them (see Figure 2.1 first row). One of the line segments is designated as the threshold of the door and is drawn as a straight black line. The dashed red lines denote the opening range of the door. We may call such a circular door just a *door*.

We generalize the concept of a door in the way that we allow shapes other than quarter circle, in particular we have squared and triangular doors defined analogously. A **squared door** is a square with one edge specified as the threshold, while a **triangular door** is an isosceles right triangle with one cathetus specified as the threshold.

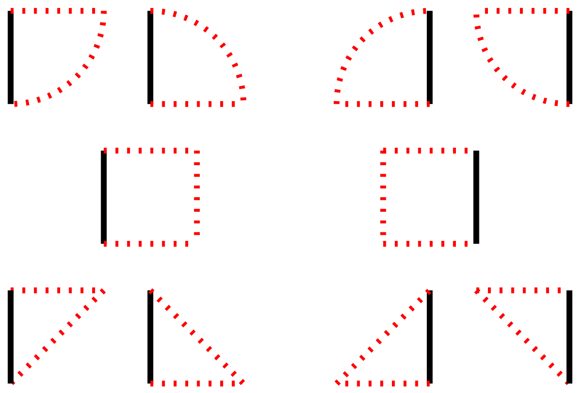
Figure 2.1: Circular, squared and triangular doors

Except for Chapter 6, we assume that doors are drawn on a grid, where the downmost and leftmost corner is denoted by the point $(0, 0)$. Therefore every point of the grid is contained in $\mathbb{N}_0^2$. Door thresholds are line segments on the grid and we denote them by their two endpoints. So every threshold is contained in $\mathbb{N}_0^2 \times \mathbb{N}_0^2$.

We usually talk about a door opening or being fixed upwards/downwards/leftwards/rightwards. The opening side is the side of the opening range with respect to the threshold and the fixed side is the side of the “hinge”. When the door threshold is horizontal or vertical, then these orientations are clear. For example, the leftmost door in the first row of Figure 2.1 is opening rightwards and fixed upwards.

Otherwise assume that we have a door threshold $(a_1, a_2) = ((a_{1,x}, a_{1,y}), (a_{2,x}, a_{2,y}))$ which is neither vertical nor horizontal. Let us draw a perpendicular line segment of the same length $(b_1, b_2) = ((b_{1,x}, b_{1,y}), (b_{2,x}, b_{2,y}))$ such that these two cross with each other at their middle points. Let us assume that the door is opening towards b_1. We say the door is opening rightwards if and only if $b_{1,x} > b_{2,x}$, leftwards if and only if $b_{1,x} < b_{2,x}$. Furthermore, let us assume that the door is fixed towards a_1. We say the door is fixed rightwards if and only if $a_{1,x} > a_{2,x}$, leftwards if and only if $a_{1,x} < a_{2,x}$. (Compare Figure 2.2)

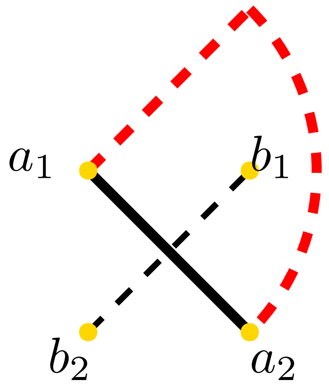

Figure 2.2: Door is opening rightwards and fixed leftwards

We call an arrangement of doors on the grid a **placement**. Formally, we denote the set of orientations with the set $O = \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$, and a placement is a function $p : D \rightarrow O \times O$ for a set of door thresholds D. For a threshold $d \in D$, $p(d) = (o, f)$ denotes the opening (o) and fixed side (f). We call such a tuple (o, f) a **door orientation**. For squared doors we omit the fixed side. We say that two doors **interfere** with each other if they are not pairwise disjoint, i.e. if they intersect with each other. A **legal** placement is one in which there are no interfering doors.

Furthermore we deal with the following door problems throughout the thesis.

A **placement problem** asks for a given set of thresholds D if there is a legal placement for D. A **replacement problem** asks for a given set of thresholds D, a threshold $d \in D$, a legal placement p for D, and a door orientation (o, f) whether there is a sequence of moves from p to another legal placement p' such that $p'(d) = (o, f)$. A **move** always consist in changing the orientation of a single door which results in a new legal placement. A **reconfiguration problem** asks for a given set of thresholds D and two legal placements p and p' whether there is sequence of moves which transforms p into p'.

Another category of problems concerning doors is motion planing. A **motion planning** problem asks for a given set of thresholds D, a placement p for D, a set of line segments W (walls), and two grid cells s and t whether an agent can reach t starting from s. The agent may be allowed to open and close doors by pushing or pulling depending on the problem.

2.1 Computational Complexity

Computational complexity deals with, among others, how much time and space resources are required in the worst case to solve a given decision problem.

A decision problem Π is a YES-NO question on a possibly infinite set of instances. An instance I of Π is said to be a YES-instance of Π if the answer to the question posed by Π is YES in the case of I, otherwise I is a NO-instance of Π. We use the words *problem* and *decision problem* interchangeably in this thesis.

A decision problem Π is said to be hard for a complexity class **CLASS**, if every problem in **CLASS** is polynomial-time many-one reducible to Π. A problem is **CLASS**-complete, if it is contained in **CLASS** and **CLASS**-hard. Note that we do not apply this definition to the complexity class **P**.

A polynomial-time many-one reduction of a problem Π to another problem Φ is a polynomial-time algorithm which transforms every instance I of Π to an instance I' of Φ such that I is a YES-instance of Π if and only if I' is a YES-instance of Φ. All of our reductions in this thesis are polynomial-time many-one reductions.

We use reductions to prove hardness of a problem Π for a complexity class by reducing a known hard problem in this class to Π. We also use reductions to prove containment of a problem Π in a complexity class by reducing Π to a problem known to be contained in this class. We usually use *gadgets* in these reductions which are parts of a reduced problem instance which simulate the behaviours of some parts of the original problem instance.

We briefly define the complexity classes we deal with in the following. We also define some Boolean satisfiability problems contained in these classes. In general, we deal with Boolean formulae in conjunctive normal form (CNF), i.e. formulae which are conjunctions of disjunctions. Thus, we can represent a Boolean formula F as a set of clauses each of which is a set of literals. We denote the set of variables which appear in F as $Var(F)$.

Let us first define three satisfiablity problems, which we are going to refer to in the following chapters:

Definition 2.1: *2-SAT*
Instance: *A Boolean formula F in CNF, in which every clause has at most two literals.*
Question: *Is there an assignment $s : Var(F) \longrightarrow \{true, false\}$, which satisfies F?*

Definition 2.2: *3-SAT*
Instance: *A Boolean formula F in CNF, in which every clause has at most three literals.*
Question: *Is there an assignment $s : Var(F) \longrightarrow \{true, false\}$, which satisfies F?*

Definition 2.3: *True Quantified Boolean Formulae (TQBF)*
Instance: *A Boolean formula of the form $Q_1x_1Q_2x_2...Q_nx_nF'$, where $Q_i \in \{\forall, \exists\}$ for $i \in \{1, ..., n\}$ and F' is a Boolean formula with $Var(F') = \{x_1, ..., x_n\}$.*
Question: *Is there an assignment $s : Var(F') \longrightarrow \{true, false\}$, which satisfies the formula $Q_1x_1Q_2x_2...Q_nx_nF'$?*

Now let us briefly define the complexity classes **P**, **NP** and **PSPACE**.

- A problem is in **P** if and only if it can be solved by a deterministic Turing machine in polynomial time. 2-SAT is in **P**.
- A problem is in **NP** if and only if it can solved by a nondeterministic Turing machine in polynomial time. Intuitively, **NP** is the class of all problems, for which a solution can be verified in polynomial time. The canonical **NP**-complete problem is the general satisfiability (SAT) problem or its restricted version 3-SAT.
- A problem is in **PSPACE** if and only if it can be solved by a deterministic Turing machine using polynomial space. The canonical **PSPACE**-complete problem is TQBF.

In the following chapters, we investigate problems about doors which are contained in the complexity classes **P**, **NP**, $\exists\mathbb{R}$, and **PSPACE**. For a brief definition of $\exists\mathbb{R}$ we refer to Chapter 6.

It is known that

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \exists\mathbb{R} \subseteq \mathbf{PSPACE}$$

and widely believed that all of the inclusions are proper.

3 Door Placement Problems

Suppose that we want to finish the construction of a floor. The walls have already been planned and it is known where the door thresholds should be. We want to know whether it is possible to determine an opening orientation for each door, such that every door can be opened and closed freely without getting in the way of other doors. This is the general placement problem we mentioned and formalized in Chapter 2. In this chapter we deal with placement problems and we are going to show that circular door placement is **NP**-complete, while squared door placement is in **P**.

3.1 DoorPlacement

We begin with the definition of the placement problem for circular doors.

Definition 3.1: *DoorPlacement*
Instance: *A set $D = \{d_1, d_2, \ldots, d_n\} \subseteq \mathbb{N}_0^2 \times \mathbb{N}_0^2$ of n door thresholds, each of which is a line segment on the grid.*
Question: *Is it possible to place a (circular) door on every door threshold in D, such that no two doors interfere with each other?*

More formally, we are asking if there is a legal placement $p : D \longrightarrow O \times O$ which assigns a door orientation to every threshold as defined in Chapter 2.

One can see all four possible placement options for a given door threshold in Figure 3.1 and interfering doors in Figure 3.2.

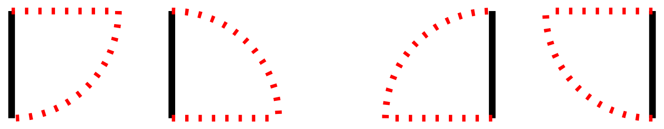

Figure 3.1: Possible placements of a door

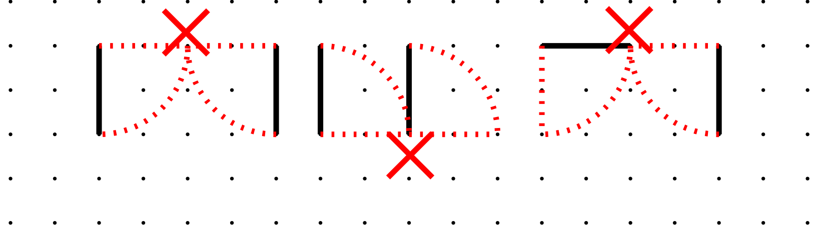

Figure 3.2: Six doors interfering at 3 different points

Lemma 3.2: *DoorPlacement is in* NP.

Proof. To prove the containment in **NP**, we give a polynomial-time verification algorithm. Given a problem instance D and a placement p, we place every door according to p and then pairwise check if any two doors interfere with each other. If this is the case, we return false. Otherwise we conclude that p is a legal placement and D is a YES-instance. This can be done in time $\mathcal{O}(|D|^2)$. Moreover, a placement $p : D \to O \times O$ always takes $\mathcal{O}(|D|)$ space. ∎

To show that DoorPlacement is **NP**-hard, we are going to reduce from the problem Planar Monotone 3-Sat, which is a restricted version of 3-Sat. It was introduced and proven to be **NP**-complete in [BK10].

In Planar Monotone 3-Sat we require that the 3-Sat formula is monotone, that means in every clause either all literals are positive or all literals are negative. Note that 3-Sat remains **NP**-complete when the formula is restricted to be monotone [GJ79]. We also require that the so-called variable-clause incidence graph is planar. This is the bipartite graph which has a vertex for every variable, a vertex for every clause, and an edge between a variable x and a clause C if and only if either the literal x or the literal $\overline{x}$ appears in C. Note that it was already proven in [Lic82] that 3-Sat remains **NP**-complete, when the variable-clause incidence graph is planar. Furthermore, we require that this graph has a plane rectilinear drawing where all variables and clauses are drawn as rectangles and all variables are drawn along a horizontal line, such that the clauses which contain only positive literals are above this line and the clauses which contain only negative literals are below this line. We call such a drawing a monotone rectilinear representation of the formula [BK10]. (See Figure 3.3 for an example)

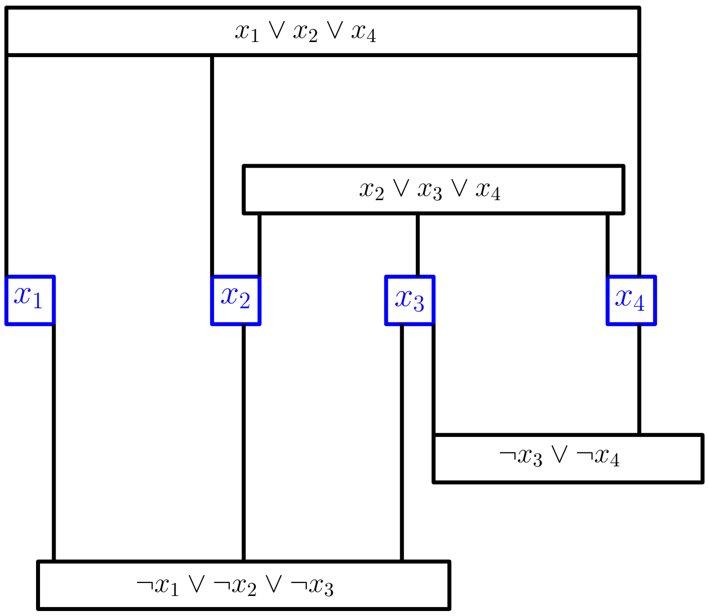

Figure 3.3: A monotone rectilinear representation

Definition 3.3: *Planar Monotone 3-Sat*

Instance: *A 3-Sat formula F which admits a monotone rectilinear representation together with such a representation.*

Question: *Is F satisfiable?*

Theorem 3.4 ([BK10, Theorem 1])**:** *Planar Monotone 3-Sat is* **NP**-*complete.*

Theorem 3.5: *DoorPlacement is* NP*-complete.*

Proof. We showed that the problem is in **NP** in Lemma 3.2.

To show **NP**-hardness, we reduce the problem Planar Monotone 3-sat to DoorPlacement using a variable gadget, a transport gadget, a split gadget, and a clause gadget. Let us introduce these gadgets in the following.

Variable gadget. For every variable in the Planar Monotone 3-sat instance, we draw a door threshold of length 2 units on the grid and we ensure that all variable gadgets are on a horizontal line, e.g. like in Figure 3.4. Setting a variable x_i to *true* (resp. *false*) corresponds to placing a door opening downwards (resp. upwards) on the variable gadget x_i.

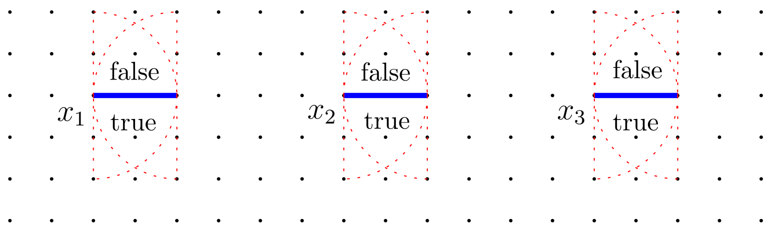

Figure 3.4: Variable gadget

Transport gadget. Placing door thresholds in a row as shown in Figure 3.5 allows to transport the information (the opening side of the door) away from the respective variable gadget. For example, placing an upwards opening door to x_i or x_j forces the other doors above them to open upwards.

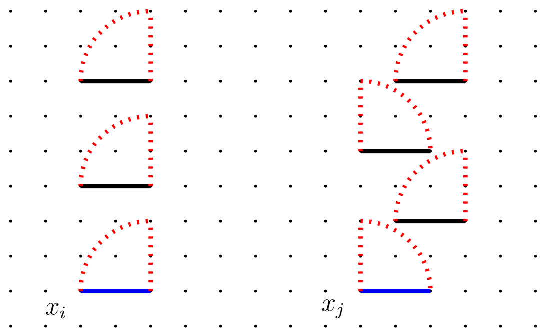

Figure 3.5: Transport gadget

Split gadget. This gadget allows us to make a turn and to make a split while transporting the information of a variable. On the left-hand side of Figure 3.6, the variable x_i is set to $false$, i.e opening upwards and forces a split rightwards and upwards. Meanwhile, on the right-hand side the variable x_i is set to *true*, i.e. the door on x_i is opening downwards and the other doors *can* be placed such that they are opening towards x_i.

Clause gadget. To simulate a 3-sat clause, we place three door thresholds which contain information from three variables (through transport and split gadgets) in such a way that at most two of the doors can be opened *towards the inside* of the clause as shown in Figure 3.7. This simulates the fact that in each 3-sat clause at most two of the literals can be set to $false$. We can also simulate a clause with two literals as shown in Figure 3.8.

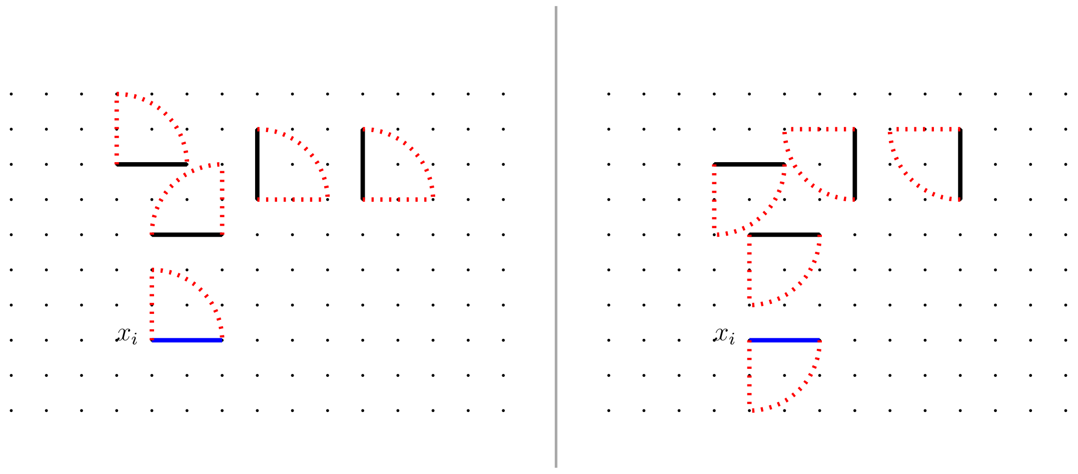

Figure 3.6: Split gadget

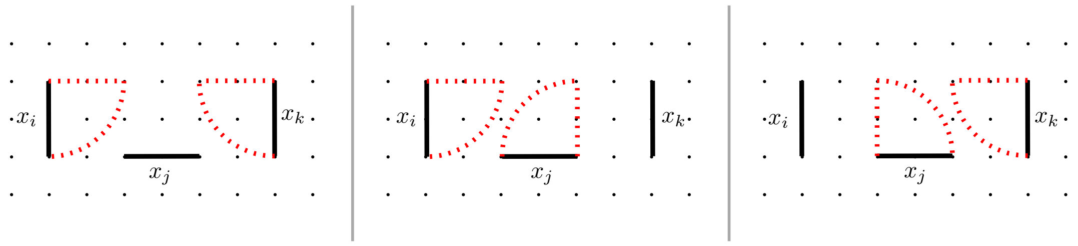

Figure 3.7: Clause gadget

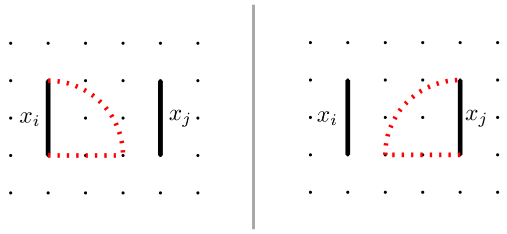

Figure 3.8: Clause gadget with 2 variables

Given a PLANAR MONOTONE 3-SAT instance $I = \langle F, R \rangle$ including the formula F and its monotone rectilinear representation R, we replace clauses in R with our clause gadgets and variables with our variable gadgets as well as edges with our transport and split gadgets and obtain the DOORPLACEMENT instance D. Note that we use the both versions of the transport gadget in Figure 3.5 interchangeably as needed to align the variables and clause gadgets.

We give in Figure 3.9 a sample reduction from a simple PLANAR MONOTONE 3-SAT formula, namely $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_4)$, with doors placed according to a satisfying assignment.

As the transport gadget has polynomial size and all other gadgets have constant size, the reduction can be done in polynomial time.

Correctness of the reduction.

Let $i \in \{1, ..., k\}$. Let $I = \langle F, R \rangle$ be a YES-instance of PLANAR MONOTONE 3-SAT. Furthermore, let $Var(F) = \{x_1, ..., x_k\}$. Consider a satisfying assignment $s : Var(F) \rightarrow \{true, false\}$. Let $\{d_1, ..., d_k\} \subseteq D$ be the corresponding variable gadgets. If $s(x_i) = false$ then we place an

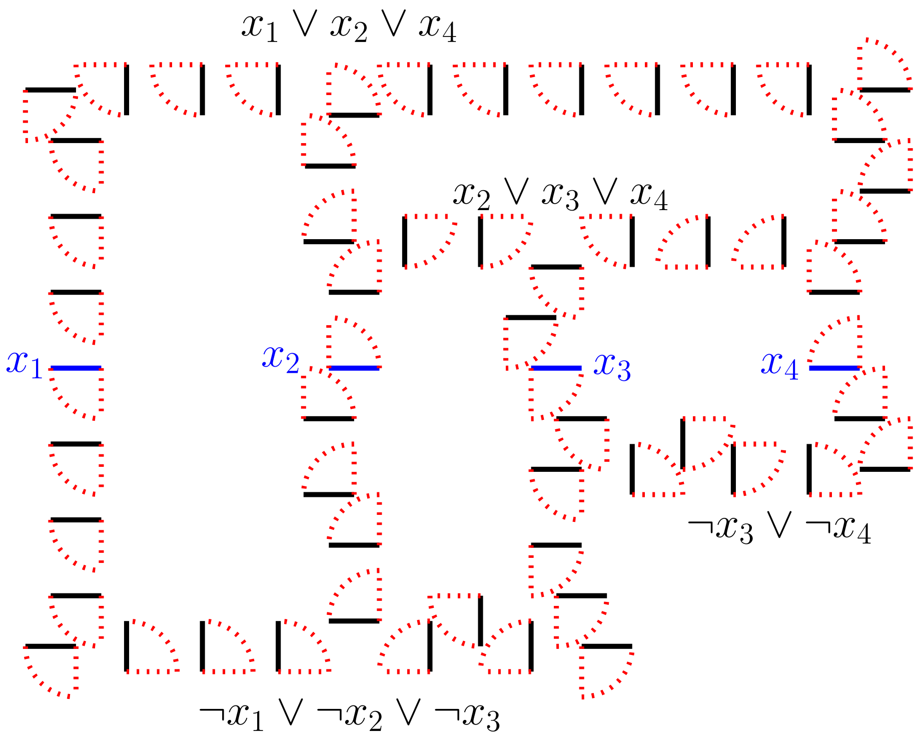

Figure 3.9: Reduction from Figure 3.3 with the assignment $x_1 = true$, $x_2 = false$, $x_3 = true$, and $x_4 = false$

upwards opening door on d_i and otherwise a downwards opening door. If the door on d_i is opening upwards (resp. downwards), the transport and split gadgets connected to the variable gadget d_i from above (resp. below) are forced open *away from* d_i, thus occupying a place in their respective clause gadgets. We make the transport and split gadgets on the remaining side of d_i open *towards* d_i, thus not occupying a place in their clauses. Consequently, if $s(x_i) = false$ (resp. $true$), the opening side of d_i causes, via transport and split gadgets, one place to be occupied in each of the positive (resp. negative) clauses in which x_i appears. Since s is satisfying, the resulting placement is legal.

Conversely, let a reduced instance D have a legal placement p. We claim that the opening sides of the doors which are placed in variable gadgets according to p induce a satisfying assignment for F, where a downwards (resp. upwards) opening door on d_i means that $x_i = false$ (resp. $true$) in that assignment. Assume that the induced variable assignment is not satisfying. Then w.l.o.g. there is a clause in F containing positive literals in which all the variables are set to $false$. That means, doors at the corresponding variable gadgets are all opening upwards. Then, in the corresponding clause gadget all the doors are forced to open towards the inside of the clause creating an interference, so the door placement p is not legal, a contradiction. ■

Note that in the reduction we only used horizontal and vertical door thresholds of size 2 units. So, we can say the following:

Remark 3.6: *DoorPlacement remains* **NP**-*complete, even when the problem is restricted to only horizontal and vertical door thresholds with size 2 units.*

Next, we look at the placement problem for squared doors and show that it is in **P**. The difference between DoorPlacement and SquaredDoorPlacement is that in the former there are four ways to place a door at a given threshold while the latter allows only two options because of the symmetricity of the square.

3.2 SquaredDoorPlacement

We start with the definition of the placement problem for squared doors.

Definition 3.7: *SquaredDoorPlacement*
Instance: *A set $D = \{d_1, d_2, ..., d_n\} \subseteq \mathbb{N}_0^2 \times \mathbb{N}_0^2$ of n door thresholds, each of which is a line segment on the grid.*
Question: *Is it possible to place a squared door on every door threshold in D, such that no two doors interfere with each other?*

More formally, we are asking if there is a legal placement $p : D \to O$ which assigns an opening side to every door threshold as defined in Chapter 2.

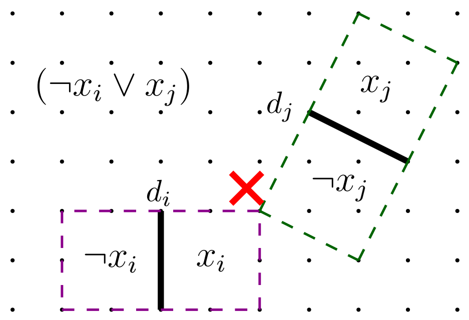

Figure 3.10: d_i and d_j cannot both open towards the middle

In Figure 3.10, one can see that for each threshold one has two options for how to place a door.

Theorem 3.8: *SquaredDoorPlacement is in* **P**.

Proof. We reduce SquaredDoorPlacement to 2-sat. Given is a problem instance D. First, we compute all the possible interferences by pairwise comparing possible orientations on a threshold with the other thresholds which can be done in $\mathcal{O}(n^2)$ time, because the number of possible interferences between n doors is at most $2n \cdot (2n-1)$. Then, for every door threshold $d_i \in D$ $(i = 1, ..., n)$ we introduce a variable x_i. For a threshold d_i, there are two sides to place a door. Let a door opening to one of the sides correspond to x_i being $false$ and a door opening to the other side correspond to x_i being $true$. We fix such a correspondence for every threshold. Then, we define the reduced 2-sat formula F as follows: If a door opening to the side $true$ on d_i interferes with a door opening to the side $true$ in d_j, we add the clause $(\neg x_i \vee x_j)$ to F. This clause reflects the fact that d_i must open to the side $false$ or d_j must open to the side $true$ so that no interference happens (Compare Figure 3.10). For all the other cases, we add clauses to F analogously and we cover all possible interferences. By construction, it is the case that F is satisfiable if and only if D has a legal placement. Furthermore, the reduction takes place in polynomial time and the size of F is at most $2n \cdot (2n-1) \in \mathcal{O}(n^2)$. ∎

Note that in the proof above we only used the fact that there are two ways to place a door on a threshold and not the fact that we have squared doors. So, the construction in Theorem 3.8 with 2-Sat works without the loss of generality for all such door thresholds with two options. So we can say the following.

Remark 3.9: *A door placement problem is contained in* **P**, *when for each door threshold there are only two options for how to place a door on this threshold.*

4 Door Replacement Problems

Imagine that we are given a floor plan where every door is already placed without problems but we are unsatisfied with the placement of a door and we would rather have it opening to the other side, i.e. we want to change its orientation. But we do not want to discard the whole plan and want to change the orientations of doors one by one. In the end, we hope to be able to change the orientation of our target door to our desired orientation. While doing these operations we do not want to mess up the plan and we want to play safe by maintaining a legal placement of doors at all times while we are changing some orientations. This is the general replacement problem we mentioned and formalized in Chapter 2. In this chapter, we are going to show that for squared doors the replacement problem is also in **P**, just like it was the case with the placement problem. After that, we are going to prove that in the case of circular doors the replacement problem is **PSPACE**-complete, so harder than its placement counterpart, which is **NP**-complete as we proved in Chapter 3.

4.1 SquaredDoorReplacement

We begin with the replacement problem for squared doors.

Definition 4.1: *SquaredDoorReplacement*
Instance*: A SquaredDoorPlacement instance D with a legal placement p, which assigns an orientation to each door threshold, and a specified door threshold $d \in D$.*
Question*: Let a move consist in reversing the orientation of a single door such that the placement remains legal. Is there a sequence of moves that eventually reverses the orientation of the door on the threshold d?*

In order to prove that the replacement problem is in **P**, we are going to be concerned with two similar 2-Sat problems: 2-Sat Reconfiguration and VariableFlip 2-Sat, respectively. Let us define 2-Sat Reconfiguration.

Definition 4.2: *2-Sat Reconfiguration*
Instance*: A 2-Sat formula F together with two satisfying assignments $s : Var(F) \rightarrow \{true, false\}$ and $t : Var(F) \rightarrow \{true, false\}$, where $Var(F)$ is the set of variables of F.*
Question*: Let a move consist in flipping the truth value of a single variable, such that the formula remains satisfied. Is there a sequence of moves that transforms s to t?*

Our definition above coincides with the definition of the problem st-connectivity (St-Conn) from the paper [GKMP06], in the case of a 2-Sat formula.

We first prove the following lemmas which will help us prove that 2-Sat Reconfiguration is in **P**.

Lemma 4.3 ([Knu08, p. 72, Theorem S.]): *Satisfying assignments for a 2-Sat formula are closed under the so-called ternary majority operation maj. That means, given three satisfying assignments a, b, and c for a 2-Sat formula F, the assignment $maj(a, b, c)$ constructed by setting each variable the truth value it has in the majority of the assignments a, b, c is also satisfying.*

Proof. Let $x, y \in Var(F)$ and w.l.o.g $(x \vee y)$ a clause in F. Let us observe the satisfying assignments $a, b, c : Var(F) \longrightarrow \{0, 1\}$, where 1 corresponds to *true* and 0 to *false*. We know that we have $(a(x) \vee a(y)) = (b(x) \vee b(y)) = (c(x) \vee c(y)) = 1$. Assume w.l.o.g $a(x) \leq b(x) \leq c(x)$. Observe that $maj(a, b, c)(x) = b(x)$. Then we have

$$maj(a, b, c)(x) \vee maj(a, b, c)(y) = b(x) \vee maj(a, b, c)(y) = 1,$$

because $b(x) = 0$ implies that $a(x) = 0$ and from that $a(y) = b(y) = 1$ follows. Thus we proved that $(x \vee y)$ evaluates to *true* under the assignment $maj(a, b, c)$. Since this clause was arbitrary, it follows that $maj(a, b, c)$ is a satisfying assignment for F. ∎

Lemma 4.4 ([GKMP06, Lemma 4.3]): *Let $I = \langle F, s, t\rangle$ be a 2-Sat Reconfiguration instance which admits an affirmative answer and let D be the set of variables on whose values s and t disagree. Then there is a sequence of moves from s to t flipping only the variables in D.*

Proof. We follow the proof from [GKMP06].

First, note that satisfying assignments for a 2-Sat formula are closed under the majority operation by Lemma 4.3.

Consider any sequence of moves M between s and t, that means $M = \langle s, u_1, ..., u_m, t\rangle$ is a sequence of satisfying assignments where consecutive assignments differ in truth value of only one variable. Let $i \in \{1, ..., m\}$. We construct another sequence M' by replacing every u_i by $v_i = maj(s, u_i, t)$, which is also a satisfying assignment. Now observe that because of the majority operation and the fact that u_i and u_{i+1} differ in only one variable value, either v_i and v_{i+1} are the same or they also differ in only one variable value. Therefore M' is a sequence of moves transforming s to t. Moreover, along M' only the variables in D are flipped because majority operation ensures that every v_i agrees with s and t on the common variable values between s and t.

∎

Theorem 4.5 ([GKMP06, Corollary 4.4]): *2-Sat Reconfiguration is in* **P**.

Proof. We follow the proof from [GKMP06].

The following algorithm decides the problem: Given is an instance $I = \langle F, s, t\rangle$. Let firstly $s' = s$, and at each step find a variable x with $s'(x) \neq t(x)$, flip it if doing so keeps the formula satisfied, and update $s'(x) = t(x)$ until $s' = t$. If at any step there is no such variable to flip then the output is NO.

If the given instance is a NO-instance, then one cannot reach t from s by a sequence of moves and this algorithm must fail. So, assume that I is a YES-instance, i.e. one can reach t from s. Let D be the set of variables, on whose values s and t disagree. We prove that the algorithm reaches t by induction on $|D|$. For $|D| = 1$, there is only one variable to flip and the algorithm flips it. Assume that the algorithm works for $|D| = m$. Let $|D| = m + 1$. Then, by Lemma 4.4 one can reach t from s by only flipping the variables in D, in particular there is a possible move on s, which consist in flipping a single variable from D. Do such a move. Between the resulting assignment s' and t there are m different valued variables. Thus, by induction hypothesis the algorithm reaches t.

So, we proved that the algorithm is correct. Furthermore, observe that if I is a YES-instance then the algorithm reaches t from s in $|D|$ moves. Moreover, the algorithm terminates in $\mathcal{O}(n^2)$ time where $n = |Var(F)|$. ∎

Analogous to this problem, we can define the problem SquaredDoorReconfiguration which, given a SquaredDoorPlacement instance D with two legal placements s and t, asks whether one can transform s into t by changing the orientation of a single door at every step while always preserving a legal placement.

Corollary 4.6: *SquaredDoorReconfiguration is in* **P**.

Proof. Given a SquaredDoorReconfiguration instance $I = \langle D, s, t\rangle$, we first reduce D to a 2-Sat formula F using Theorem 3.8. Furthermore, we convert s and t to assignments s' and t' for F in accordance with our reduction to F. Thereby, we converted our problem to a 2-Sat Reconfiguration instance $\langle F, s', t'\rangle$, which can be solved in polynomial time by Theorem 4.5. ■

But recall that we were interested in proving that the problem SquaredDoorReplacement is in **P**, which is different from SquaredDoorReconfiguration. To prove that, we define the problem VariableFlip 2-Sat, the 2-Sat-analogue of SquaredDoorReplacement and prove the stronger statement that VariableFlip 2-Sat is in **P**.

Definition 4.7: *VariableFlip 2-Sat*

***Instance**: A 2-Sat formula F together with a satisfying assignment $s : Var(F) \longrightarrow \{true, false\}$, where $Var(F)$ is the set of variables of F and a specified variable $x \in Var(F)$.*

***Question**: Let a move consist in flipping the truth value of a single variable, such that the formula remains satisfied. Starting with the assignment s, is there a sequence of moves that eventually flips the value of x?*

$$(\overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_3 \vee x_4) \wedge (\overline{x_4} \vee \overline{x_5}) \wedge (x_4 \vee x_1)$$

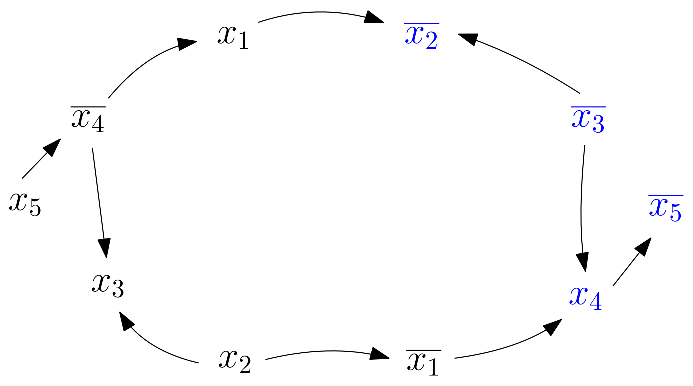

Figure 4.1: Implication graph of a 2-Sat formula. In every satisfying assignment where $\overline{x_3}$ is true, all blue literals $(\overline{x_3}, \overline{x_2}, x_4, \overline{x_5})$ must be true.

Theorem 4.8: *VariableFlip 2-Sat is in* **P**.

Proof. Let $I = \langle F, s, x\rangle$ be a VariableFlip 2-Sat instance. The following algorithm decides whether I admits an affirmative answer:

Consider the directed implication graph G of F defined as follows: the vertices are positive and negative literals of the variables $Var(F)$. There is a directed edge from a literal l to a literal l' if and only if F contains a clause involving l' and the negation of l. Note that if l is a negative literal $\overline{y}$, then the negation of l is y. (See Figure 4.1 for an example.)

Assume w.l.o.g. we want to flip the variable x from $true$ to $false$ by a sequence of moves. Take the literal $\overline{x}$ in G. Consider the subgraph G' constructed by taking all vertices reachable from $\overline{x}$, including $\overline{x}$. (If we had the graph in Figure 4.1 with $x = x_3$, G' would correspond to blue vertices.)

If G' contains y and $\overline{y}$ for some variable y, then there is no satisfying assignment of F in which x is assigned the value $false$. In this case, the algorithm outputs NO. Otherwise there are no contradicting literals in G'. So, consider the assignment t constructed in the following way: For a variable $z \in Var(F)$:

$$t(z) = \begin{cases} true & \text{if } z \text{ appears in } G' \\ false & \text{if } \overline{z} \text{ appears in } G' \\ s(z) & \text{otherwise} \end{cases}$$

The algorithm decides whether the 2-Sat Reconfiguration instance $\langle F, s, t\rangle$ is a YES-instance using the algorithm in Theorem 4.5 and gives the answer as output.

First, we prove that t is satisfying. Assume for the sake of contradiction that t is not satisfying. Take a clause $(l \vee l')$ in F which evaluates to $false$ under t. Then it follows that neither the literal l nor the literal l' is in G'. That means $(l \vee l')$ also evaluates to $false$ under s. But we know that s is satisfying, a contradiction. Therefore t is also a satisfying assignment.

Next, we show that the algorithm is correct. If I is a NO-instance, then there is no sequence of moves transforming s into t, because x appears flipped in t and thus the algorithm fails. So, assume that I is a YES-instance. Then, there is an assignment s' so that $s'(x) = false$ which is reachable from s.

By Lemma 4.4, it follows that in order to reach s' from s, one only has to flip variables on which s and s' disagree. Furthermore, note that s' agrees with t on the values of the variables which has a literal in G'. Because $x = false$ implies these values. So assume $s' \neq t$, then they must disagree on a variable value, which has no literal in G'.

Consider a sequence of satisfying assignments M, which transforms s into s':

$$M = \langle s = s_0, s_1, ..., s' = s_n\rangle$$

Let $i \in \{0, ..., n\}$. We construct another sequence M' by replacing every s_i by $v_i = maj(s, s_i, t)$ which is also a satisfying assignment. Now observe that because of the majority operation and the fact that s_i and s_{i+1} differ in only one variable value, either v_i and v_{i+1} are the same or they also differ in only one variable value. Moreover, it holds that $maj(s, s', t) = t$, because t agrees with s on the values of the variables which has no literal in G' and t agrees with s' on the values of the variables which has a literal in G'. Therefore, M' is a sequence of satisfying assignments transforming s into t.

That means, if there is an s' with $s'(x) = false$ and which is reachable from s then t is also reachable from s and the algorithm outputs YES.

∎

Corollary 4.9: *SquaredDoorReplacement is in* P.

Proof. Given a SquaredDoorReplacement instance $I = \langle D, p, d\rangle$, we first reduce D to a 2-Sat formula F using Theorem 3.8. Furthermore, we convert the placement p to an assignment for F in accordance with our reduction to F. Thereby, we converted our problem to a VariableFlip 2-Sat instance $\langle F, s, x\rangle$, where $x \in Var(F)$ is the variable corresponding to d. This can be solved in polynomial time by Theorem 4.8. ∎

As the proofs do not rely on the specific geometry of the doors, the construction in Corollary 4.9 with 2-Sat works without the loss of generality for all such door thresholds with two options. So, we can say the following.

Remark 4.10: *A door replacement problem is contained in* **P***when for each door threshold, there are only two options for how to place a door on this threshold.*

4.2 DoorReplacement

We continue with the replacement problem for circular doors and show that it is complete for **PSPACE**.

Definition 4.11: *DoorReplacement*

Instance: *A DoorPlacement instance D together with a legal placement p, a specified door threshold $d \in D$ with its desired orientation $(o, f) \in O \times O$, where o denotes the opening side and f the fixed side as before.*

Question: *Let a move consist in removing a door from a threshold and placing it to the same threshold with another orientation such that the resulting placement is legal. Is there sequence of moves that eventually result in a legal placement p' such that $p'(d) = (o, f)$?*

Lemma 4.12: *DoorReplacement is in* **PSPACE**.

Proof. The current placement p of an instance at any given time takes only $\mathcal{O}(|D|)$ amount of space. The possible moves at any given time can be computed in polynomial time by iterating over each door and listing possible replacement options by checking possible inteference around that door. Therefore, at any given move we can nondeterministically guess a move and make it while only keeping track of the current placement and not the previous ones. That means, the problem is in **NPSPACE** and thus in **PSPACE**, by Savitch's theorem that **PSPACE** = **NPSPACE** [Sav70]. ■

To prove the **PSPACE**-hardness of the problem, we want to reduce from Nondeterministic Constraint Logic (NCL). NCL was proposed in [HD05] as a model of computation based on making moves by reversing weighted edges in a directed multigraph (self-loops are allowed in our defintion of a multigraph) while maintaining certain minimum in-flow constraints of the vertices. A move from one configuration to another consist in the reversal of a single edge, such that the constraints remain satisfied. This problem has turned out to be helpful in simplifying **PSPACE**-hardness proofs of several motion planning and sliding block puzzles like Generalized-Rush-Hour and Sokoban [HD05]. Two of the decision problems arising from this model of computation are the configuration-to-edge problem and the configuration-to-configuration problem. The former asks whether, given a configuration of the multigraph, a specified edge can be eventually reversed by sequence of moves. The latter asks whether, given two configurations of the multigraph, one can be transformed into the other by a sequence of moves. Both were proven to be **PSPACE**-complete in [HD05].

In NCL every vertex v is assigned a nonnegative integer minimum in-flow constraint c_v and every edge e is assigned a nonnegative integer weight w_e. A configuration (direction of the edges in the NCL instance) is legal if and only if the weights of the edges directed towards a vertex v sum up to c_v, for all vertices v in the NCL instance.

So, the definition of NCL in its configuration-to-edge variant, which we simply call NCL, is as follows.

Definition 4.13 ([HD05]): *Nondeterministic Constraint Logic (NCL)*

Instance: *An undirected multigraph $G = (V, E)$ together with an assignment of nonnegative integer weights to edges and nonnegative integer minimum in-flow constraints to vertices, and a configuration E_C of G which specifies a direction for every edge in E such that this configuration satisfies the minimum in-flow constraints. Furthermore a target edge $e \in E$ is specified.*

Question: *Let a move from one configuration to another consist in the reversal of a single edge, such that the constraints remain satisfied. Is there sequence of moves that eventually reverses the target edge e?*

However, we want to use a restricted version of NCL, namely when the multigraph is restricted to be planar and a so-called AND/OR constraint graph.

An AND/OR constraint graph is a multigraph which consists only of vertices of the kind shown in Figure 4.2 and Figure 4.3. The blue edges have a weight of 2, while the red edges have a weight of 1. For every vertex, the minimum-inflow constraint is 2. AND vertices are incident to 2 red edges and 1 blue edge, whereas OR vertices are incident to 3 blue edges. That means, in an AND vertex the blue edge could face outwards if and only if both of the red edges are facing inwards. In an OR vertex, at least one of the three edges has to face inwards to satisfy the constraint. In that sense, these vertices simulate the logical AND and OR operators. Since we might want to connect e.g. the edge from an OR vertex to an edge from an AND vertex, a conversion gadget is employed which has the effect that one end of an edge can be blue while the other end can be red.

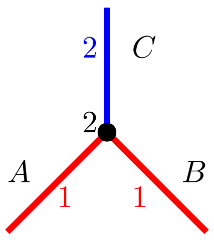

Figure 4.2: AND vertex with edges A (red), B (red) and C (blue): Either C is directed inwards or both A and B are directed inwards.

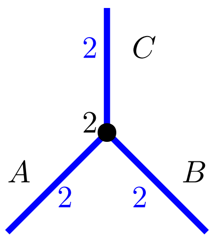

Figure 4.3: OR vertex with edges A (blue), B (blue) and C (blue): At least one of A, B, and C is directed inwards.

Theorem 4.14 ([HD05]): *NCL is* **PSPACE**-*complete, even when the multigraph is planar and restricted to be a so-called AND/OR constraint graph.*

Proof Sketch. NCL is in **NPSPACE** = **PSPACE**, because the configuration takes only linear amount of space at a current state, the possible moves in a state can be computed in polynomial time and one can nondeterministically guess a move to make while only keeping track of the current configuration.

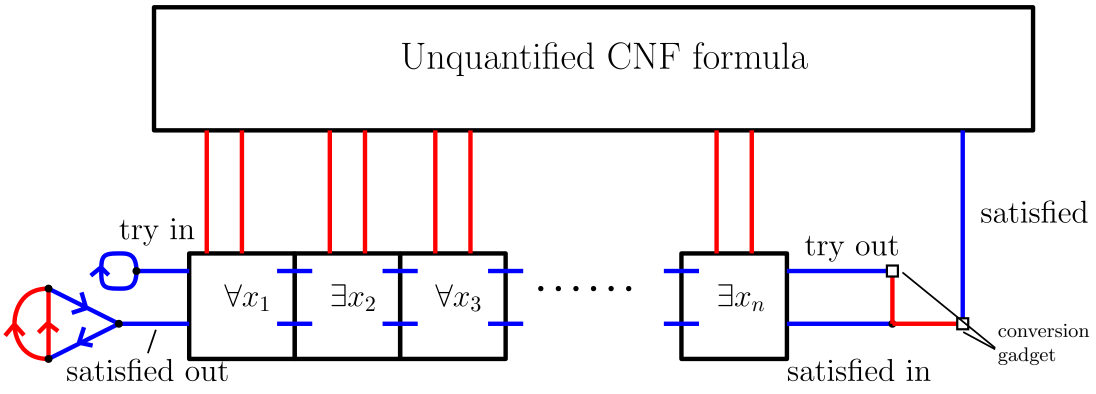

Figure 4.4: Reduction from TQBF to NCL. (taken from [HD05], redrawn)

To show **PSPACE**-hardness, [HD05] gives a reduction from TQBF to an AND/OR constraint graph and an edge in that graph is specified which can be reversed as a sequence of moves if and only if the TQBF formula is satisfiable. Furthermore, planar crossover gadgets are presented to prove that NCL remains **PSPACE**-complete even in its planar version.

The reduction is based on the idea that one can determine if a TQBF formula is satisfiable with the following recursive algorithm: One begins with the first quantifier in the formula and sets its variable to $true$ and then to $false$, recursively checking for both cases if the remaining formula is satisfiable with this (partial) assignment. In the case of an existential quantifier, the algorithm returns true if at least of one assignment of this variable succeeds. In the case of a universal quantifier, the algorithm returns true if both assignments of this variable succeed. The base case of the recursion is checking if the formula is satisfiable when all the variables are assigned.

The reduction is as follows: Given is a TQBF instance I. Firstly, with AND and OR vertices a so-called CNF network is built which corresponds to the unquantified version of I. Furthermore, for each quantifier, a quantifier gadget is used which outputs two edges corresponding to two different assignments of the variable it quantifies, which go into the CNF network. These quantifier gadgets are connected with each other in a string. Every quantifier gadget has a “try in”, a “try out”, a “satisfied in”, and a “satisfied out” edge as shown in Figure 4.4. The graph has a legal configuration when all “try in” edges are directed leftwards and thus I is reduced to such a configuration. Now, the goal is to reverse edges one by one maintaining a legal configuration in the graph to eventually reverse the target edge e (from rightwards to leftwards), which is the “satisified out” edge of the leftmost quantifier. This will be possible if and only if I is satisfiable.

A quantifier gadget becomes active when the quantifier gadgets to its left have fixed their variable assignments and its “try in” edge is thus directed inwards. Loosely speaking, a quantifier gadget can nondeterministically choose a variable assignment by directing one of its two output edges outwards and tries to satisfy the remaining formula under that assignment. This happens in that the quantifier gadget directs its “try out” edge outwards. The output from the CNF network which is the edge labeled “satisfied” is connected to the rightmost quantifier. A quantifier gadget can direct its “satisfied out” edge outwards if and only if the CNF network with quantifiers to the right (including itself) is satisfiable given the fixed variable assignments on the left and its “satisfied in” is directed inwards. The existential (resp. universal) quantifier gadgets are built accordingly to ensure that at least one variable assignment (resp. both variable assignments) are required for it to direct its "satisfied out" edge outwards.

▢

Theorem 4.15: *DoorReplacement is* **PSPACE**-*complete.*

Proof. Lemma 4.12 proves that the problem is in **PSPACE**. We show that the problem is **PSPACE**-hard by a reduction from NCL restricted to be a planar AND/OR constraint graph.

Given is a NCL instance I which contains a planar multigraph $G = (V, E)$ which consists of AND and OR vertices exclusively, the configuration E_C (i.e. direction of the edges) and the target edge $e \in E$. We are going to simulate AND and OR vertices directly in our replacement problem, thus we make no distinction between doors, i.e. there are no red or blue doors. Therefore, we do not need a red-to-blue conversion gadget, as is the case in the reduction from Theorem 4.14.

In the following, we look at AND, OR and the edge gadget we use for our reduction.

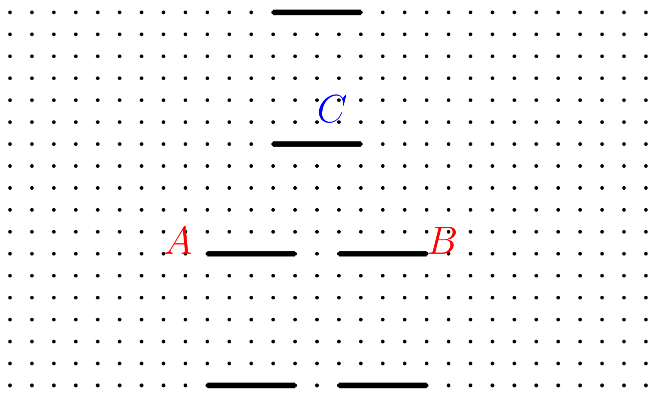

Figure 4.5: AND gadget

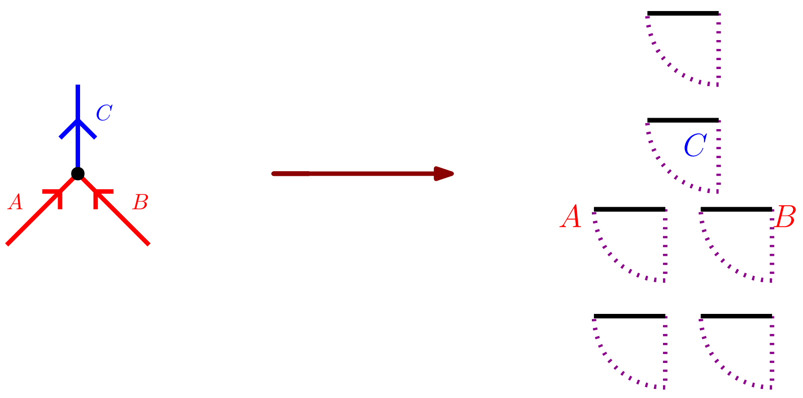

Figure 4.6: Simulating the direction of edges in an AND vertex. Inward edges correspond to outward doors and vice versa

AND gadget. We simulate an AND vertex with door thresholds as shown in Figure 4.5. All the door thresholds have length 4 units. Observe that because of the threshold locations, either the door on threshold C has to face outwards (i.e. upwards in the drawing) or the doors on the thresholds A and B both have to face outwards (i.e. downwards in the drawing) in order that there are no interference. The fixed sides of the doors do not matter here. So, in the reduction, an inward edge in an AND vertex of the NCL instance corresponds to a door opening outwards and vice versa as one can see in Figure 4.6.

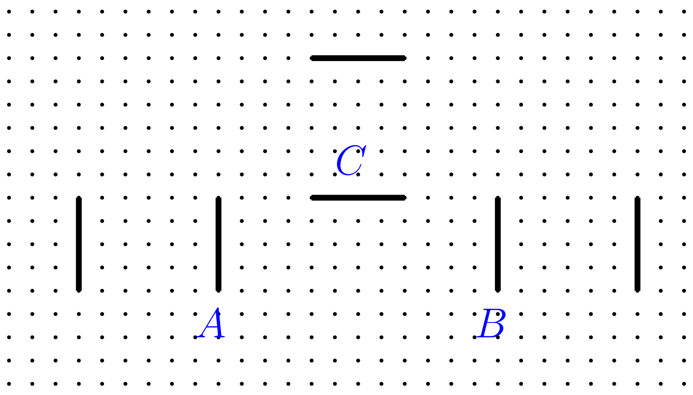

Figure 4.7: OR gadget

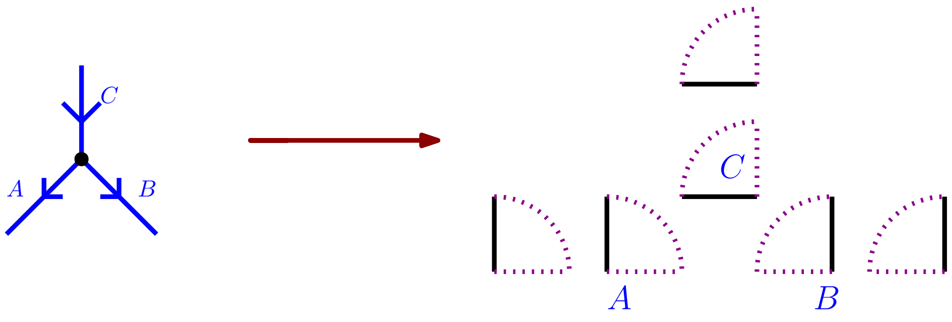

Figure 4.8: Simulating the direction of edges in an OR vertex. Inward edges correspond to outward doors and vice versa

OR gadget. Similarly, we simulate an OR vertex with door thresholds as shown in Figure 4.7. All the door threshold have length 4 units. Observe that because of the threshold locations, at least one of the doors has to face outwards in order that there are no interference. This is essentially the clause gadget we build in the DoorPlacement problem from Theorem 3.5. In Figure 4.8, one can see how an OR vertex is reduced. Again, an inward edge in the NCL instance corresponds to a door opening outwards and vice versa.

Note that a move consists in changing the placement of a single door and not necessarily reversing it. So, it is also a valid move when one changes the fixed side of a door while keeping the opening side same. Such a move might be needed in an OR vertex. For example, assume that in Figure 4.3 the door on threshold C is fixed leftwards and directed downwards whilst A and B are directed leftwards and rightwards respectively. Then it should be possible to reverse the door at A because the OR gadget has one more free “slot”. However, in order to reverse A, one first makes a move changing the fixed of the door at C while keeping the opening side the same and then reverses A.

Edge gadget. To simulate edges, one places thresholds 5 or 6 units apart from each other like in Figure 4.9 and the information gets transported along the edge gadget. One can also make arbitrary turns. Compare Figure 4.11 to see the edge gadgets in action. Furthermore, we can easily simulate self-loops like in Figure 4.10. Note that self-loops only appear in the

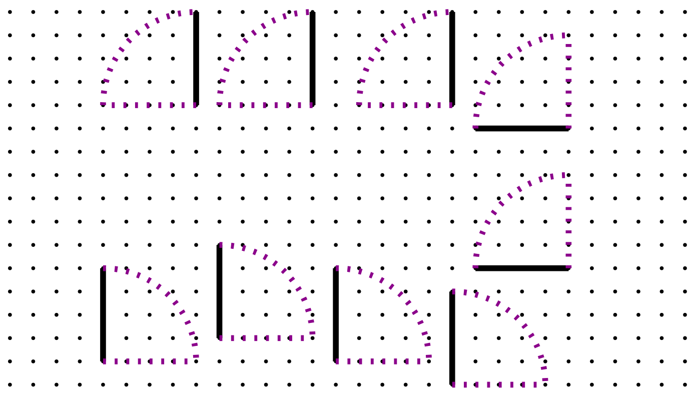

Figure 4.9: Edge gadget

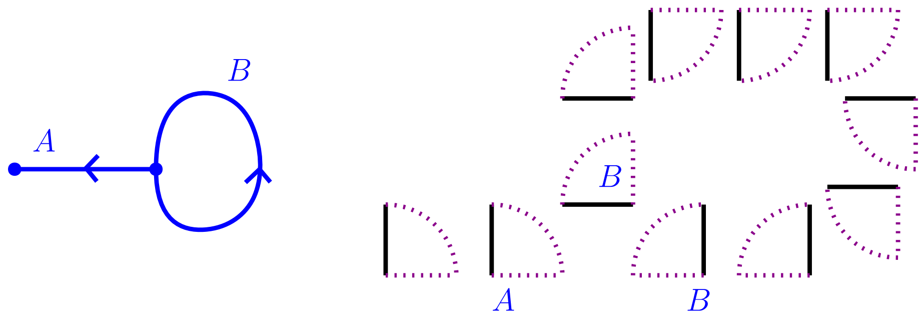

Figure 4.10: Simulating a self-loop

context of OR vertices in the reduction from TQBF to NCL and reversing a self-loop edge does not make any sense, a self-loop edge just indicates that the remaining edge in the OR vertex is free to have either direction.

Let us refer to the thresholds in AND and OR gadgets of the reduced DoorReplacement instance which are denoted with the letters A, B and C in our drawings as vertex gadgets and all the other thresholds as edge gadgets.

Recall that we are promised that the multigraph G is planar and consists of only AND/OR vertices. So, in particular it has maximum degree 3. Thus, we compute a plane rectilinear drawing of it which can be done in polynomial time using algorithms from [NR04]. We replace every vertex with the corresponding gadgets shifting and aligning the thresholds in edge gadgets when necessary. Then, we place the doors on the thresholds according to the orientation of the edges E_C and obtain a placement p together with a set of thresholds D. This is a legal placement because by construction of the AND and OR gadgets they simulate the NCL vertices and we have no interference between doors, since the configuration E_C is legal. As the edge gadget has polynomial size and the other gadgets have constant size, this can be done in polynomial time. We pick as our target door any door threshold d along the edge gadget corresponding to the target edge e in the NCL instance. So, we want reverse the opening side o of d while keeping its fixed side f same. Thus, we obtain the DoorReplacement instance $I' = \langle D, p, (o_{rev}, f)\rangle$, where o_{rev} is the reverse direction of o. In Figure 4.11, one can see a sample reduction.

Correctness of the reduction. Recall that we have an NCL instance I. Let I be a YES-instance. Then there is a sequence of moves M which eventually reverses the target edge e and we can simulate the moves in I' as follows: We start with the first move in M and

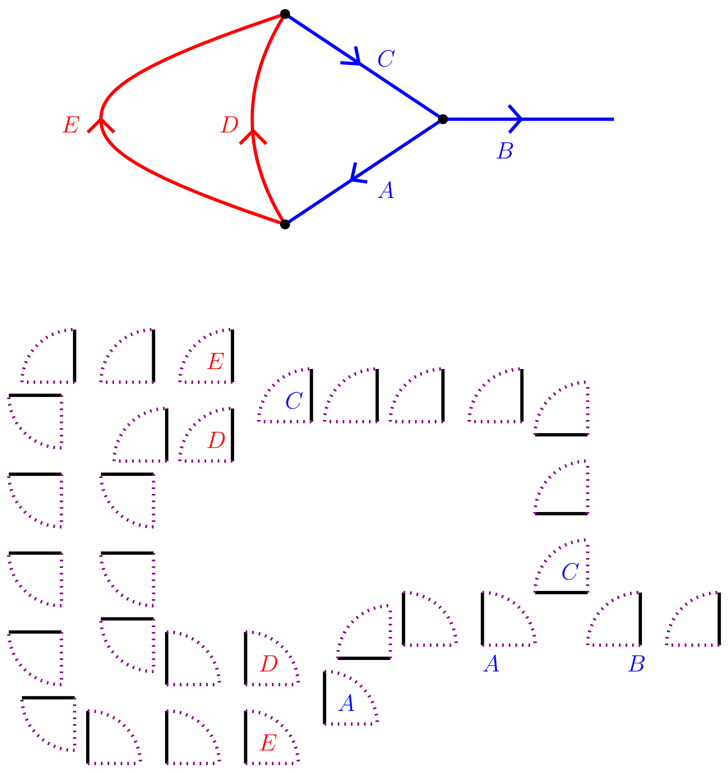

Figure 4.11: Reduction from a small part of an NCL instance

consider the edge gadget of the edge it reverses. At the two ends of this edge gadget we have two thresholds which are part of vertex gadgets. We can assume that exactly one of these thresholds has a door directed outwards from its vertex gadget. We replace this door with a door of reverse opening direction and then one by one replace all the doors of the edge gadget and then the door of the vertex gadget at the other end also with doors of reverse opening direction (Compare Figure 4.12). Thus, we completed the first move and essentially reversed the edge. We continue with the other moves in the same way. At the end, we have a move in I which reverses e. When we simulate this move in I', we replace every door along the edge gadget of e with a door of reverse opening direction, in particular we replace the door on d.

Conversely, assume that the reduced DoorReplacement instance I' admits an affirmative answer. So there is a sequence of moves which replaces the door on d with a door having a reverse opening side but the same fixed side. If we can replace this door right away or just by reversing the doors along the same edge gadget (including the vertex gadget doors at both ends of the edge gadget), then we can also reverse the edge in I right away. Otherwise we have a sequence of moves in I' that will lead to making room in the needed vertex gadget so that we can reverse doors along the edge gadget including d. This sequence of moves can always look like described in the previous paragraph, i.e. at each step reversing the doors at an edge gadget completely corresponding to an edge reversal in I. Because, observe that a single move in I takes more than one move in I' (Compare Figure 4.12). So, we are essentially building an asynchronous version of NCL similar to [Vig13], in the sense that while we are at the process of replacing doors along an edge gadget in I' (i.e. doing the analogue of reversing an edge in I), the doors on the thresholds at the both ends of the edge gadget are both directed inwards to their vertex gadgets. But this temporary situation creates only a disadvantage

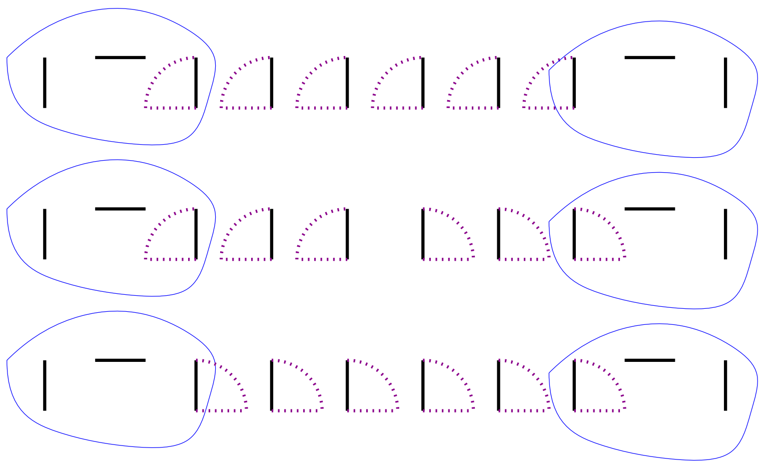

Figure 4.12: Reversing an edge gadget starting with the reversal of the rightmost door and continuing towards left. The OR vertex gadgets marked with blue.

because this edge gadget basically occupies a place in both of the vertex gadgets. So, we may reverse the remaining doors along this edge gadget to create us an advantage for further moves. Thus, the equivalent of reversing an edge in I take more than one move in I' and in the temporary phase it is as if the edge being reversed in I is pointing away from both ends. Therefore, we can translate the sequence of moves that will lead to reversing d in I' to a sequence of moves in I that will lead to the reversal of e. ■

Note that analogous to the configuration-to-configuration variant of NCL, we can also define the problem DoorReconfiguration which asks, given two placements p and p' for a set of door thresholds D, whether one can transform p into p' by replacing a single door at a time and maintaining a legal placement at each step.

Corollary 4.16: *DoorReconfiguration is* **PSPACE***-complete.*

Proof Sketch. By the same argument as in Lemma 4.12, one can show that DoorReconfiguration is in **NPSPACE** = **PSPACE**. For **PSPACE**-hardness, one can reduce from the configuration-to-configuration variant of NCL using the same gadgets as in Theorem 4.15 and by using the asynchronous correspondence between moves and configuration in NCL and those in DoorReplacement or DoorReconfiguration. □

Note that in the reduction we only used horizontal and vertical door thresholds of size 4 units. So, we can say the following:

Remark 4.17: *DoorReplacement remains* **PSPACE***-complete, even when the problem is restricted to only horizontal and vertical door thresholds with size 4 units.*

5 Motion Planning Problems

Suppose that we already have a finished floor plan with doors and walls placed. Now, imagine a scenario where an agent must navigate through this floor filled with doors and walls. The natural question arising from a scenario like that is whether the agent can reach a specific location starting from another specific location. This type of questions are broadly called *motion planning* problems. In this chapter we want to look at motion planning problems involving doors. Depending on how we restrict the capabilities of the agent (doors can only be pushed or pulled etc.), we identify several problems and analyze the complexities of them.

In motion planning problems, we have a grid consisting of points with natural number coordinates as before, where the leftmost and downmost point is $(0, 0)$. Additionally we have 1×1 square cells whose corner points are the grid points. We denote every cell by its down-left corner point. We call the edges of these cells *cell borders*. Each cell border is a line segment and is denoted by its two endpoints.

Moreover, we are given a set of door thresholds D, each of which is a single cell border on the grid, in particular they are 1 unit long and either horizontal or vertical. We are also given a placement $p : D \longrightarrow O \times O$, which assigns every threshold a door orientation (o, f) that denotes the opening and fixed side of the door on this threshold as before.

For a door on a threshold $d \in D$ there are two possible states: open and closed. When closed, it occupies the cell border d. When open, d is no longer occupied and the door occupies the perpendicular cell border in its opening range which is determined by its orientation given in p.

Additionally, we are given a set of walls W, where each element of W is a single cell border on the grid. A wall $w \in W$ occupies a cell border b if $w = b$.

At a given time the agent is on a single cell. The agent can move to an adjacent cell if and only if the common border of two cells is free. A cell border b is free if and only if b is not currently occupied by a wall or a door. Note that a cell border can be occupied by both a door and a wall at the same time but cannot be occupied by two doors at the same time.

So, the current state of a motion planning instance $\langle D, p, W \rangle$ is fully determined by the current cell c of the agent and the current states of doors $z : D \longrightarrow \{open, closed\}$.

5.1 Push-Open-Doors

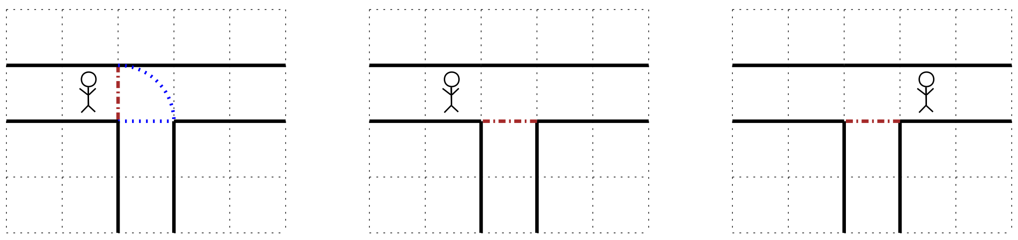

Figure 5.1: An agent opens a door by pushing and then moves two cells to the right.

In the problem Push-Open-Doors, the agent can interact with the doors by pushing them. The agent at a cell c can only push a door, if

- the door is at its closed state,
- the door occupies one of the four borders of c,
- the door opens away from the agent (opening range of the door is not at the cell c),
- and the cell border, which the door will occupy in its open state, is not occupied by any door (but possibly occupied by a wall).

After this interaction, the agent is still at c but the door is at its open state. In particular, an opened door cannot be closed, hence every door can be interacted with at most once.

Let us sum up the definition of the decision problem Push-Open-Doors, which we are going to prove to be **NP**-complete.

Definition 5.1: *Push-Open-Doors*

Instance: *A set $D \subseteq \mathbb{N}_0^2 \times \mathbb{N}_0^2$ of n door thresholds, a set $W \subseteq \mathbb{N}_0^2 \times \mathbb{N}_0^2$ of walls, where each threshold and each wall is a horizontal or vertical line segment on the grid of length 1 unit (hence a cell border); a (not necessarily legal) placement $p : D \longrightarrow O \times O$ and two cells $s, t \in \mathbb{N}_0^2$. Furthermore, we require that there is a rectangular region enclosed by walls such that all the other objects of the problem instance are contained inside this region.*

Question: *Can an agent in the cell s reach the cell t by doing a sequence of actions, where a single action consists in moving to an adjacent cell or opening a door by pushing?*

Lemma 5.2: *Push-Open-Doors is in* **NP**.

Proof. Let $I = \langle D, W, p, s, t\rangle$ be a Push-Open-Doors instance and $g : \{1, ..., k\} \longrightarrow \{1, ..., n\}$ an injective function. Since every door can be opened at most once, if t can be reached from s in I, then there is an ordered sequence $D' = \langle d^{(1)}, ..., d^{(k)}\rangle$ where $d^{(i)} = d_{g(i)} \in D$, such that D' specifies the order in which the agent can visit and open the doors to reach t from s. As $k \leq n$, D' is in particular of linear size. Given an I and a witness D', we can verify whether the agent can reach t from s in I as follows:

Let G_z be the graph where every cell of the problem instance is a vertex and there is an edge between two cells if they are adjacent and their common border is free in state $z : D \longrightarrow \{open, closed\}$. Furthermore let z_0 be the state where all the doors are closed and z_i be the state where $d^{(1)}, ..., d^{(i)}$ are opened and the rest is closed. Let c_i be the cell which is neighbouring $d^{(i)}$ from the opposite of its opening side, i.e. one can possibly open $d^{(i)}$ at c_i.

First, check with a breadth-first-search algorithm if the cell c_1 is reachable from s in G_{z_0}, then if c_i is reachable from c_{i-1} in $G_{z_{i-1}}$ for all $i \in \{1, ..., k\}$ and at last if t is reachable from c_k in G_{z_k}. As everything is inside a rectangular region enclosed by walls, $|G_z|$ is in particular bounded by $|W|^2$. So this can be done in polynomial time as breadth-first-search algorithm is linear in $|G_z|$ and it is used k times. ∎

In order to prove **NP**-hardness, we are going to reduce from the problem Push-1 which was proved to be **NP**-hard in [DDHO03]. It is a pushing-block puzzle similar to the classic Sokoban.

Definition 5.3 ([DDHO03]): *Push-1*

Instance: *A rectangular grid of square cells, where each cell is either occupied by a block/the agent or free. The agent can move horizontally and vertically and push blocks but only at most one block at a time. When pushed, a block changes its location to the adjacent cell in the direction it was pushed, which is only possible if this adjacent cell is free. Furthermore, two cells s and t are marked on the puzzle.*

Question: *Is there a sequence of actions such that by doing them the agent can move from s to t, where an action consists in moving to a free adjacent cell or pushing a block on an adjacent cell?*

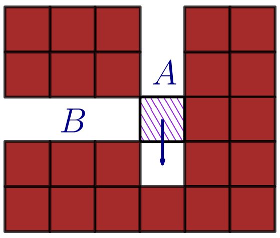

Figure 5.2: A simple Push-1 puzzle

In Figure 5.2, one can see a simple instance. The agent can reach B from A by first pushing the purple-striped block downwards and then moving leftwards. However, if the agent would be located initially at B, then the agent could not reach A because the purple-striped block cannot be pushed. This is in fact the one-way gadget from [DDHO03]. Observe how the blocks which are filled with brown colour cannot be pushed because the agent can push only one block at a time, i.e. the agent is not “powerful” enough to push two blocks at once. In that sense, these blocks effectively act like a wall.

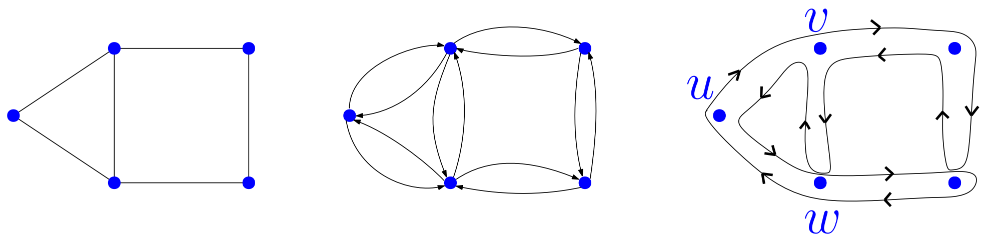

Figure 5.3: Building a planar Eulerian tour

Theorem 5.4 ([DDHO03, Theorem 6]): *Push-1 is* **NP***-hard.*

Proof Sketch. We sketch the proof given in [DDHO03].

The reduction is from Planar-3 Coloring, a well-known **NP**-complete problem [GJ79], which asks whether the vertices of a given planar graph can be colored with 3 colors in total such that no two adjacent vertices have the same color. Let G be a planar graph and $\vec{G}$ the directed multi-graph resulting from G by replacing every undirected edge by two opposite directed edges. By [BKM98], it can be shown that there is a planar Eulerian tour E in $\vec{G}$. Planar means that E does not cross itself at vertices. In Figure 5.3, one can see an undirected graph (on the left) turned into a directed one (in the middle) and a planar Eulerian tour for this graph (on the right). The idea of the proof is to replace every edge and vertex of $\vec{G}$ by sub-puzzles and connect them together to a single Push-1 puzzle, such that the agent is forced to traverse the equivalent of E in the puzzle in order reach from s to t when s and t are selected as two points at the ends of the sub-puzzle representing an arbitrary vertex.

While traversing the equivalent of E in the puzzle, the agent is forced to choose the color of a vertex, whenever it is leaving the vertex, by choosing one of the three corridors labeled 1, 2, and 3. These three corridors represent a single outgoing edge in $\vec{G}$ from that vertex. A so-called fork gadget is employed to force the agent to choose exactly one of the three corridors. Similarly, a so-called one-way gadget is employed when the three corridors are rejoining at the other vertex, to block the agent from going back to another corridor. Furthermore, the two edges of opposite direction between every two adjacent vertices in $\vec{G}$ are connected via the so-called coloring gadget, which ensures that adjacent vertices are colored with different colors. For example, if in one direction the path number 1 is traversed, then it is not possible to traverse the path number 1 in the opposite direction. Another so-called consistency gadget ensures in a similar fashion that in all of the outgoing edges from a vertex the same color is chosen. That means, for example if in one direction the path number 1 is traversed then it is not possible to traverse the paths number 2 and 3 in the opposite direction.

Therefore, the agent can move from s to t in the reduced Push-1 instance if and only if it could traverse the Eulerian path E assigning different colors to adjacent vertices along the way by choosing from a total number of 3 colors. That is the case if and only if the planar graph G is 3-colorable.

In Figure 5.4, one can see how consistency and coloring gadgets are placed. Note that the fork gadget depicted in the figure has three exits. This can be simulated by two fork gadgets with two exits. The fork and one-way gadgets ensure that the edges are always traversed in the intended direction.

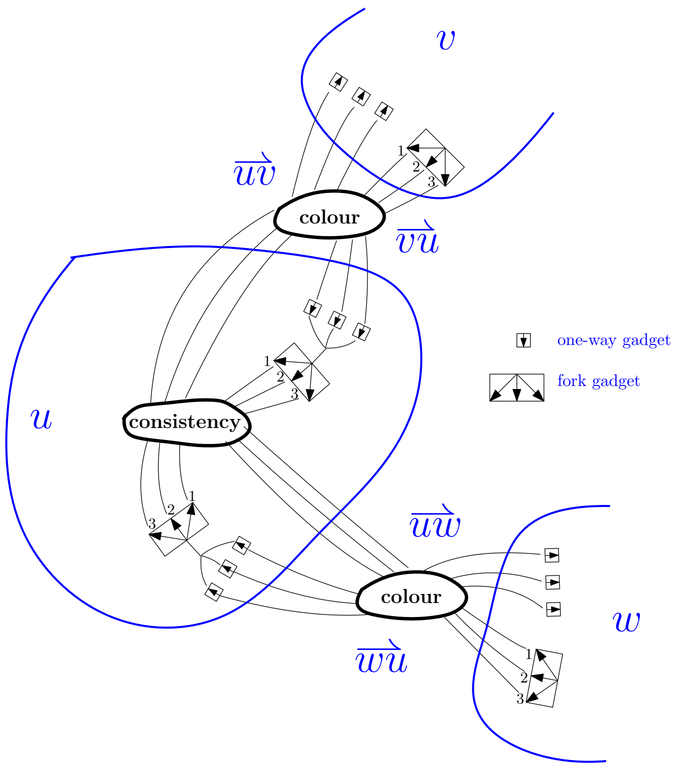

Figure 5.4: Modelling vertices and edges (from [DDHO03], redrawn)

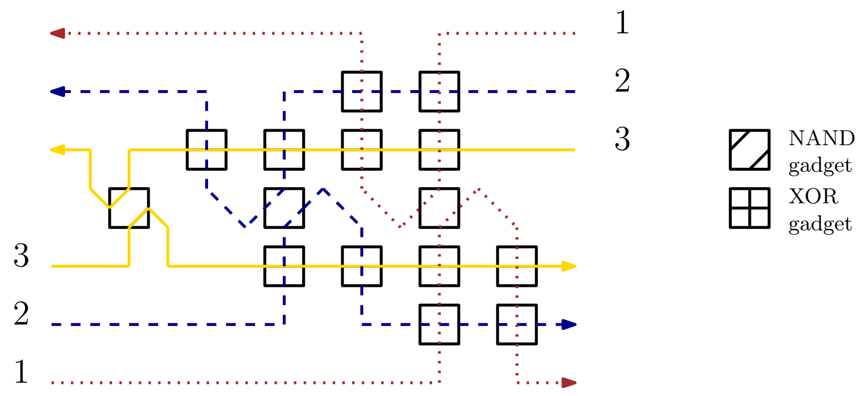

Figure 5.5: Coloring gadget (taken from [DDHO03], redrawn)

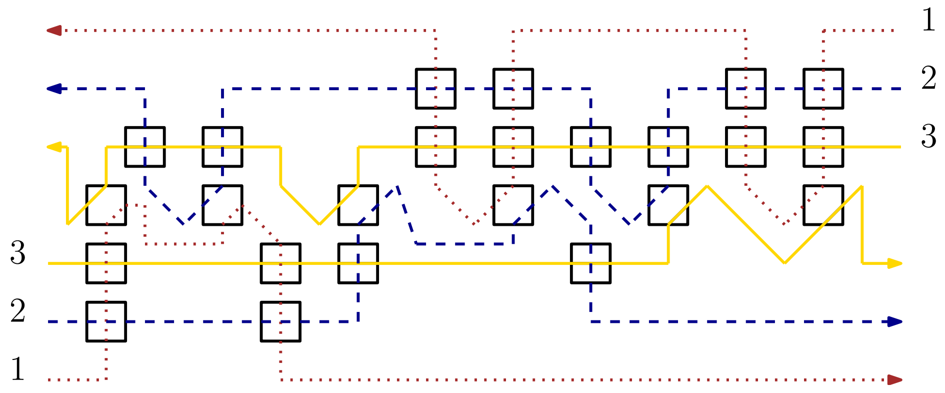

Figure 5.6: Consistency gadget (taken from [DDHO03], redrawn)

To implement the mentioned coloring and consistency gadgets (see Figure 5.5 and Figure 5.6), the authors use three different gadgets as building blocks. So, together with the mentioned one-way and fork gadgets, there are in total five essential gadgets, which we should further analyze. Intuitively, the XOR gaadget allows traversal either in west-to-east or in north-to-south direction, the NAND-1 gadget either in north-to-west or in south-to-east direction, the NAND-2 gadget either in south-to-east or west-to-north direction. We describe the features of the actual gadgets from the paper. In the following, the path notation we use (e.g. A-B is open/closed) means that the agent can/cannot do a full traversal in the given direction, i.e enter the gadget from the former location and exit from the latter.

1 **One-way gadget**
Locations: A, B
Initial state: Only A-B open
After A-B traversal: B-A becomes open

2 **Fork gadget**
Locations: A, B, C
Initial state: All paths open
After A-C traversal: A-B and C-B become closed
After A-B traversal: A-C and B-C become closed

Note that initially a fork gadget cannot be traversed starting from B or C because the ends of the fork gadget are protected by one-way gadgets. (see Figure 5.4)

3 **XOR gadget**
Locations: N (north), S (south), W (west), E (east)
Initial state: Only N-S, W-E, S-E, E-S open
After N-S traversal: S-N, W-N, E-N become open. W-E, S-E, E-S become closed
After W-E traversal: E-W, N-W, S-W become open. N-S, E-S become closed.

Note that the XOR gadget is only used as a part of the coloring and consistency gadgets and since one-way and fork gadgets are placed to the both ends of these gadgets, the XOR gadget cannot initially be traversed from S or E. (see Figure 5.4)

4 **NAND-1 gadget**
Locations: N (north), W (west), S (south), E (east)
Initial state: Only N-W and S-E open
After N-W traversal: W-N becomes open. S-E becomes closed
After S-E traversal: E-S becomes open. N-W becomes closed.

5 **NAND-2 gadget**
Locations: N (north), W (west), S (south), E (east)
Initial state: Only W-N and S-E open
After W-N traversal: N-W becomes open. S-E becomes closed.
After S-E traversal: E-S becomes open. W-N becomes closed.

□

Lemma 5.5: *The functionality of the fork gadget from Theorem 5.4 is also ensured by the following altered definition:*
Locations: A, B, C
Initial state: Only A-B and A-C open
After A-C traversal: C-A becomes open. A-B becomes closed.
After A-B traversal: B-A becomes open. A-C becomes closed.

Proof. The intended functionality of the gadget is that on first traversal, it forces an agent in location A to choose one of B and C. After the agent moves to one of B and C, the other one becomes inaccessible from A. As mentioned in Theorem 5.4, the exits B and C are protected by one-way gadgets, thus initially the agent is forced to enter from A. Therefore, the feature that all paths are open in the initial state in the original definition is not necessary. Thus, our new definition is stricter in the sense that it closes some of the paths which are available but cannot be used in the original definition. ■

Lemma 5.6: *The functionality of the XOR gadget from Theorem 5.4 is also ensured by the following altered definition:*
Locations: N (north), S (south), W (west), E (east)
Initial state: Only N-S and W-E open
After N-S traversal: S-N and W-N becomes open, W-E becomes closed
After W-E traversal: E-W and N-W becomes open, N-S becomes closed

Proof. The intended functionality of the gadget is that it lets the agent move either from N to S or from W to E and choosing one of them blocks the other. As mentioned in Theorem 5.4, the gadget can be initially only entered from one of N or W because coloring and consistency gadgets are protected by fork and one-way gadgets. Thus, again, our new definition is stricter in the sense that it closes some of the paths which are available but cannot be used in the original definition. ■

Lemma 5.7: *The functionality of the NAND-1 gadget from Theorem 5.4 is also ensured by the following altered definition:*
Locations: N (north), W (west), S (south), E (east)
Initial state: Only N-W and S-E open
After N-W traversal: S-E becomes closed. (added: S-N becomes open)
After S-E traversal: N-W becomes closed. (added: N-S becomes open)

Lemma 5.8: *The functionality of the NAND-2 gadget from Theorem 5.4 is also ensured by the following altered definition:*
Locations: N (north), W (west), S (south), E (east)
Initial state: Only W-N and S-E open
After W-N traversal: S-E becomes closed. (added: S-W becomes open)
After S-E traversal: W-N becomes closed. (added: W-S becomes open)

Proof of Lemma 5.7 and Lemma 5.8. Observe that in the consistency and the coloring gadget (see Figure 5.5 and Figure 5.6), NAND gadgets are used between opposite edges to ensure that if one of the two paths going through a NAND gadget is traversed, then the other one is blocked. The new definitions for the gadgets given here are the same as in Theorem 5.4 except for the ones marked as "added". The added features allow that on a second traversal of the gadget from the other entrance (i.e from W if the agent entered from S on the first traversal and vice versa), the agent could exit the gadget through the entrance which was used on the first traversal. However, this does not create any advantage for the agent in trying to find a way from s to t, because this action has the same effect as though the agent goes back in its own route to that point and this is anyways at any time possible because none of the gadgets from Theorem 5.4 employ any structure which would block the agent to go back in its own route. ∎

Theorem 5.9: *Push-Open-Doors is* **NP***-complete.*

Proof. Lemma 5.2 proves that the problem is in **NP**. For the proof of **NP**-hardness, we rely heavily on the proof of Theorem 5.4. Namely, by implementing the functionality of the five essential gadgets mentioned there, we can apply the same reduction to Push-Open-Doors.

Door one-way gadget. (Figure 5.7) This gadget can only be traversed from A to B by pushing the door. Once traversed, it is open from both sides. It replicates the one-way gadget in Theorem 5.4.

Door fork gadget. (Figure 5.8) This gadget can only be traversed starting at A. There are two options: If the right door is pushed, B is permanently blocked and the agent can continue to C. If the left door is pushed, C is permanently blocked and the agent can continue to B. It implements the fork gadget defined in Lemma 5.5.

Door XOR gadget. (Figure 5.9) This gadget can only be traversed starting at N or at W. If traversed starting at N, E gets permanently blocked and the agent can continue to S. If traversed starting at W, S gets permanently blocked and the agent can continue to D. It implements the XOR gadget defined in Lemma 5.6

Door NAND-1 gadget. (Figure 5.10) This gadget can only be traversed starting at N or at W. If traversed starting at N, E gets permanently blocked and the agent can continue to S. If traversed starting at W, S gets permanently blocked and the agent can continue to E. It has the same functionality as the NAND-1 gadget in Theorem 5.4 except for the fact that once the N-S path (resp. W-E path) is traversed an agent coming from W (resp. N) can reach N (resp. W). Thus, it implements the NAND-1 gadget defined in Lemma 5.7

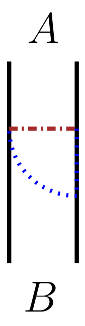

Figure 5.7: Door one-way gadget

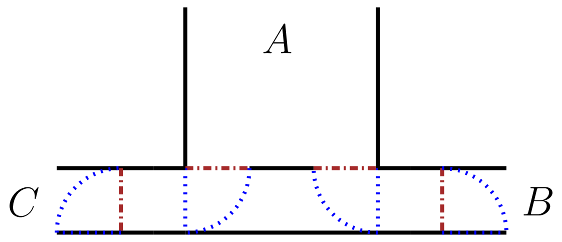

Figure 5.8: Door fork gadget

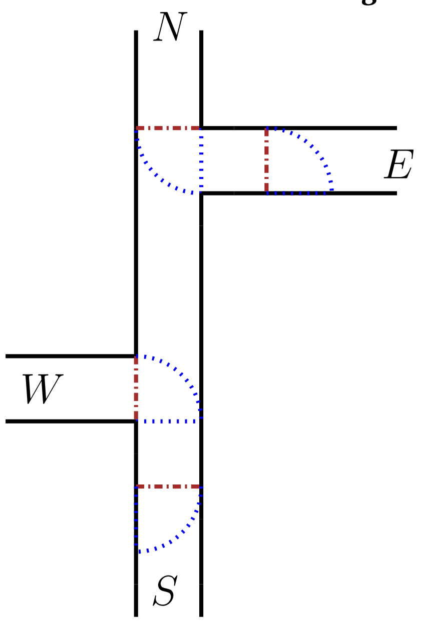

Figure 5.9: Door XOR gadget

Door NAND-2 gadget. (Figure 5.11) This gadget can only be traversed starting at W or at S. If traversed starting at W, E gets permanently blocked and the agent can continue to N. If traversed starting at S, N gets permanently blocked and the agent can continue to E. It has the same functionality as the NAND-2 gadget in Theorem 5.4 except for the fact that once the W-N path (resp. S-E path) is traversed an agent coming from S (resp. W) can reach W (resp. S). Thus, it implements the NAND-2 gadget defined in Lemma 5.8

■

Note that our XOR and NAND gadgets in Theorem 5.9 use the same gadget drawn on the left in Figure 5.12. This gadget is employed to ensure that opening a door blocks access to another door. That means traversing A-B, blocks any future access to C. As shown in Figure 5.12, this gadget can be replicated in the Push-1 setting. That means, the original fork, XOR and NAND gadgets of [DDHO03] mentioned in Theorem 5.4 could be in a sense unified by rebuilding all of them according to our door gadgets using this single gadget.

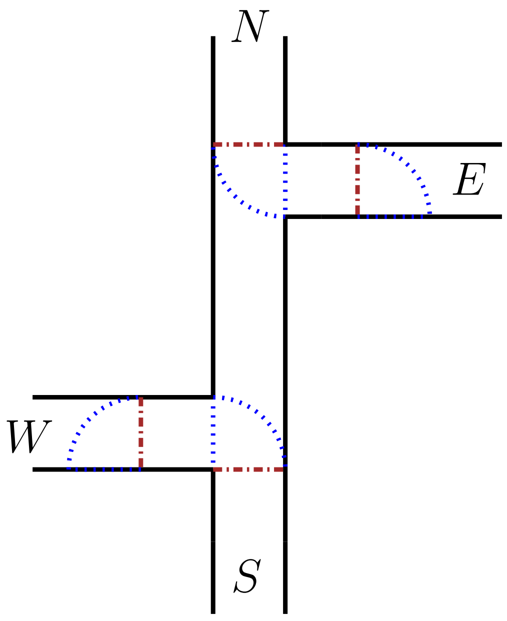

Figure 5.10: Door NAND-1 gadget

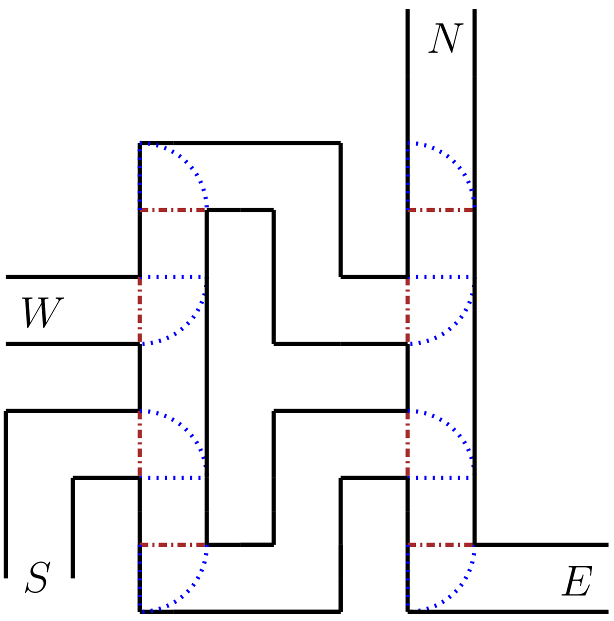

Figure 5.11: Door NAND-2 gadget

Let us call this gadget the block gadget. Then, we can simplify all of our door gadgets as shown in Figure 5.13, by turning and mirroring the block gadget as needed. That means, for a door motion planning problem it is enough to implement a one-way and a block gadget to show that it is **NP**-hard.

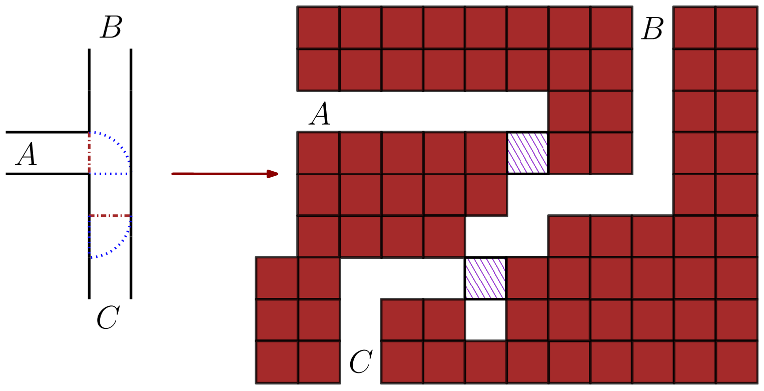

Figure 5.12: Replicating door gadgets in Push-1

Note that all of our gadgets have legal door placements in the sense that none of the doors interfere with each other. That means the hardness of Push-Open-Doors does not rely on the specific geometry of the doors (circular, squared etc.).

Remark 5.10: *Push-Open-Doors remains* **NP**-*complete even when the given placement is restricted to be legal and even when the doors are squared.*

5.2 Push/Pull-Open-Doors

Analogous to pushing, we can also define *pulling* a door.

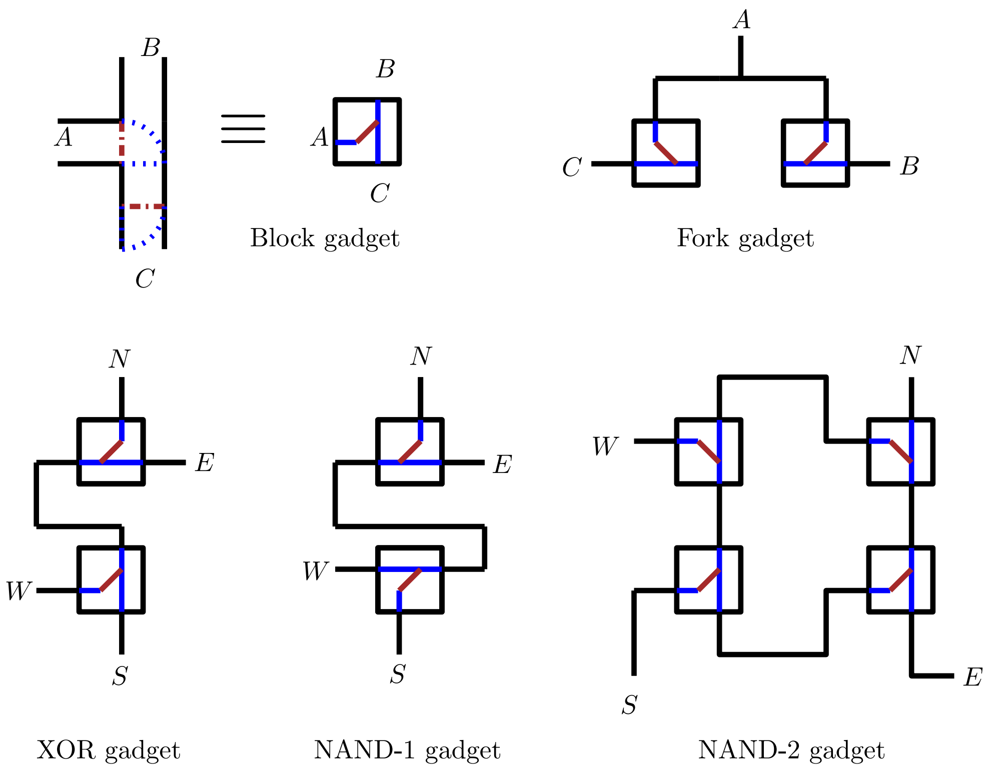

Figure 5.13: Building all gadgets out of the block gadget

The agent at a cell c can only pull a door, if

- the door is at its closed state,
- the door occupies one of the four borders of c,
- the door opens towards the agent (opening range of the door is at the cell c),
- and the cell border, which the door will occupy in its open state, is not occupied by any door (but possibly occupied by a wall).

After this interaction, the agent is still at c but the door is at its open state. In particular, an opened door cannot be closed, hence every door can be interacted with at most once.

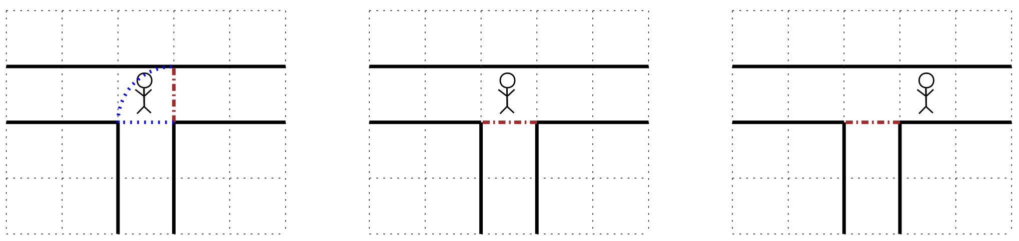

Figure 5.14: An agent opens the door by pulling and moves one cell to the right.

Now, we can define the problem Push/Pull-Open-Doors analogously to Push-Open-Doors. In Push/Pull-Open-Doors, the agent is allowed to open doors both by pushing and pulling.

Theorem 5.11: *Push/Pull-Open-Doors is* **NP***-complete.*

Proof. The problem is in **NP** by a similar argument to Lemma 5.2. The only difference is that the witness is like $D' = \langle (d^{(1)}, op^{(1)}), \ldots, (d^{(k)}, op^{(k)}) \rangle$, where $op^{(i)} \in \{push, pull\}$. That means, when giving a an ordered sequence of doors to open as a witness, one has to specify for each of them if it is to be pushed or pulled.

For **NP**-hardness, we build the one-way and the block gadget to imitate the reduction from Planar 3-Coloring, because all of the gadgets in Theorem 5.9 can be built out of these two gadgets (Figure 5.13).

One-way gadget. (Figure 5.15) This one-way gadget is stricter than the one-way gadget in Theorem 5.9, in the sense that once A-B path is traversed by opening the door by pulling, it is not possible to go back to A from B and a second A-B traversal is also not possible. But this is not a problem as each of these gadgets are only used once to go from A to B.

Block gadget. (Figure 5.16) This block gadget is also stricter than the one in Figure 5.13, in the sense that once A-B or B-C is traversed, the traversals B-A and C-B as well as future A-B and B-C traversals are not possible. This is not a problem, because in the reduction each of these paths are only used once. Apart from that, an A-B traversal blocks all future exits to C as usual.

■

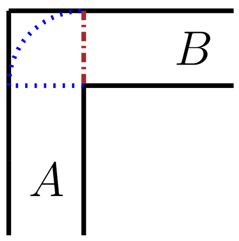

Figure 5.15: One-way gadget

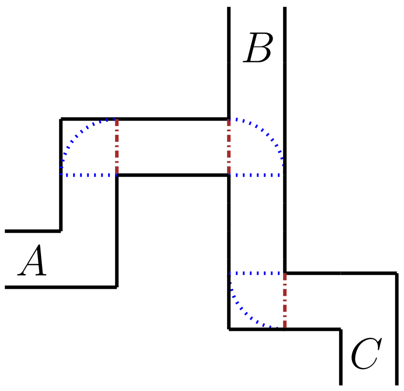

Figure 5.16: Block gadget

Note that we can define the problem Pull-Open-Doors analogously. This problem is in **NP** by the similar argument to Lemma 5.2. It is open, whether it is also hard for **NP**.

Furthermore, we can also define the problems Push-Open/Close-Doors, Pull-Open/Close-Doors, and Push/Pull-Open/Close-Doors, where we also allow the agent to close doors. These problems are clearly in **NPSPACE=PSPACE**, because we can nondeterminitically "guess" a move while only keeping track of the state of the doors and the location of the agent. It is open, whether some of these problems are in **NP** or even **P** and whether some of them are hard for **NP** or **PSPACE**. We suspect that Push-Open/Close-Doors could be **NP**-complete by a reduction to and from Push-Open-Doors. Namely, it could be the case that the ability to close doors does not create an advantage for the agent in that case.

6 An $\exists\mathbb{R}$-complete Door Placement Problem

In this chapter, we change the placement problem to include real number ranges to place doors and we are going to show $\exists\mathbb{R}$-completeness in the case of triangular doors. But we first begin with a brief introduction to the existential theory of the reals and $\exists\mathbb{R}$.

6.1 Existential Theory of the Reals

The existential theory of the reals (ETR) is the set of all true sentences of the from

$$\exists x_1, ..., x_n(\Phi(x_1, ..., x_n)),$$

where Φ is a quantifier-free Boolean formula which may include the symbols $\{0, 1, +, -, \cdot, <, \leq, >, \geq, =, \neq\}$ and connectives $\{\vee, \wedge, \neg, \Leftrightarrow\}$ having their usual meanings and where all variables range over the real numbers [Mat14]. So, we can pose ETR as a decision problem:

Definition 6.1: *Existential Theory of the Reals (ETR)*
Instance: *A quantifier-free formula $\Phi(x_1, x_2, ..., x_n)$ with variables $x_1, x_2, ..., x_n$ over the signature $\{0, 1, +, -, \cdot, <, \leq, >, \geq, =, \neq\}$ and connectives $\{\vee, \wedge, \neg, \Leftrightarrow\}$.*
Question: *Do there exist real numbers $x_1, x_2, ..., x_n$, such that $\Phi(x_1, x_2, ..., x_n)$ is true?*

This leads us to the definition of the complexity class $\exists\mathbb{R}$, which was introduced by [Sch10].

Definition 6.2: *The complexity class $\exists\mathbb{R}$ consists of all decision problems which are reducible to ETR in polynomial time. A decision problem Π is $\exists\mathbb{R}$-hard if ETR is reducible to Π in polynomial time.*

So, we can say that ETR has the same meaning for $\exists\mathbb{R}$, as the satisfiability problem (SAT) for **NP**. Since one can easily write any SAT formula as an ETR formula and ETR can be decided in polynomial space [Can88], it follows that

$$\mathbf{NP} \subseteq \exists\mathbb{R} \subseteq \mathbf{PSPACE}.$$

Intuitively, one can say that an $\exists\mathbb{R}$-hard problem is at least as hard as finding real number solutions to a system of equalities and inequalities of polynomials with integer coefficients.

6.2 RealTriangularDoorPlacement

Imagine that we are given a floor plan and doors (or objects for that matter) have to be placed. Every door has a designated area so that it should be placed somewhere inside that area and we do not want any door to come into the way of each other. We can also think of this problem as if we are given a placement and we are allowed to shift and rotate some doors inside their respective allowed space to resolve the conflicts between them and have them all pairwise interior-disjoint in the end. In the following, we define this problem formally and prove that it is ∃ℝ-complete for triangular doors when we allow real numbers to make shifts and rotations. In the case of circular doors, we could not in particular build the gadgets we have for triangular doors. So, we leave the question for circular doors open.

In the problem RealTriangularDoorPlacement, every door is opening like an isosceles right triangle. So, in input, we are given a set of isosceles right triangles T_s and we are allowed to shift them by a real number amount horizontally and vertically. The maximum shift we can apply to a triangle in the horizontal and the vertical dimension is given separately by the function $s : T_s \longrightarrow \mathbb{N}_0^2$. Let $t_s \in T_s$ and $s(t_s) = (x, y)$. Then t_s can be moved in the positive x-direction (resp. y-direction) by a non-negative real number amount but at most x (resp. y) units. Note that x and y are allowed to be zero. Thus, some doors may not be able move at all, some may be allowed to move only move horizontally or only vertically, while others can move in two dimensions. Furthermore, we are given another set of triangles T_r, which we can rotate freely around their middle point of their hypotenuse. The question is if we can apply shifts and rotations to doors such that in the end the interiors of the triangles (i.e. interior areas of the triangles excluding the edges) are pairwise disjoint.

Definition 6.3: *RealTriangularDoorPlacement*

Instance: *Two sets T_s and T_r of isosceles right triangles, each denoted by three corner points with integer coordinates and a function $s : T_s \longrightarrow \mathbb{N}_0^2$.*

Question: *Let $t_s \in T_s$, $s(t_s) = (x, y)$, and `shifted` be a function that maps every triangle t_s in T_s to a triangle t_s' such that t_s' is the same triangle as t_s shifted along positive x-direction (resp. y-direction) by a non-negative real number amount at most x (resp. y) units. Moreover let $t_r \in T_r$ and `rotated` be function that maps every triangle t_r in T_r to a triangle t_r' such that t_r' is the same triangle as t_r rotated around the middle point of its hypotenuse by some degree. Do there exist such functions(i.e. shift and rotation operations for triangles in T_s and T_r) so that the set of triangles $T = \{\texttt{shifted}(t_s) | t_s \in T_s\} \cup \{\texttt{rotated}(t_r) | t_r \in T_r\}$ is pairwise interior-disjoint?*

Lemma 6.4: *RealTriangularDoorPlacement is in ∃ℝ.*

Proof Sketch. Let $I = \langle T_s, s, T_r \rangle$ be a RealTriangularDoorPlacement instance with $T_s = \{t_{s,1}, ..., t_{s,n}\}$ and $T_r = \{t_{r,1}, ..., t_{r,m}\}$. Let $s(t_{s,i}) = (s_{i,x}, s_{i,y})$ and $t_{s,i} = \langle (a_{s,i}, b_{s,i}), (c_{s,i}, d_{s,i}), (e_{s,i}, f_{s,i}) \rangle$ for all $i \in \{1, ..., n\}$ and finally $t_{r,j} = \langle (a_{r,j}, b_{r,j}), (c_{r,j}, d_{r,j}), (e_{r,j}, f_{r,j}) \rangle$ for all $j \in \{1, ..., m\}$.

We reduce I to the following ETR formula

$$\exists t_{s,1}', ..., t_{s,n}', t_{r,1}', ..., t_{r,m}' \left(\bigwedge_{\substack{i,j \in \{1,...,n\} \\ i \neq j}} \texttt{Shifted}(t_{s,i}', t_{s,i}, s_{i,x}, s_{i,y}) \wedge \texttt{InteriorDisjoint}(t_{s,i}', t_{s,j}') \right) \wedge$$
$$\left(\bigwedge_{\substack{i,j \in \{1,...,m\} \\ i \neq j}} \texttt{Rotated}(t_{r,i}', t_{r,j}) \wedge \texttt{InteriorDisjoint}(t_{r,i}', t_{r,j}') \right) \wedge \left(\bigwedge_{\substack{i \in \{1,...,n\} \\ j \in \{1,...,m\}}} \texttt{InteriorDisjoint}(t_{s,i}', t_{r,j}') \right)$$

where

- `Shifted`$(t_{s,i}', t_{s,i}, s_{i,x}, s_{i,y})$ returns true iff $t_{s,i}'$ is the same triangle as $t_{s,i}$ but shifted along x-axis at most $s_{i,x}$ units and along y-axis at most $s_{i,y}$ units.
- `Rotated`$(t_{r,i}', t_{r,j})$ returns true iff $t_{r,i}'$ is the same triangle as $t_{r,j}$ but rotated around the middle point of its hypotenuse.

- `InteriorDisjoint`$(t'_{r,i}, t'_{r,j})$ returns true iff the edges of $t'_{r,i}$ and $t'_{r,j}$ do not pairwise cross (but may touch) with each other and no corner point of either triangle is in the interior of the other triangle.

$\square$

To prove $\exists\mathbb{R}$-hardness, we reduce from a problem called Planar-ETR-INV*.

Definition 6.5 ([LMM18]): *Planar-ETR-INV**
Instance: *A set of variables $\{x_1, ..., x_n\}$ and a set of equations and inequalities of the form $x = 1, x + y \leq z, x + y \geq z, x \cdot y \leq 1, x \cdot y \geq 1$, for $x, y, z \in \{x_1, ..., x_n\}$. Moreover, the so-called variable-constraint incidence graph, i.e the bipartite graph which has a vertex for every variable and every constraint and an edge between a variable and a constraint if the former appears in the latter, is planar.*
Question: *Do there exist real numbers $x_1, ..., x_n \in [\frac{1}{2}, 4]$ such that they are a solution to the given system of equations and inequalities?*

This problem was introduced and proven to be $\exists\mathbb{R}$-complete by [LMM18] as a planar version of ETR-INV. In [LMM18], Planar-ETR-INV* is then reduced to the graph drawing problem the paper deals with. The problem ETR-INV, which was introduced and proven to be $\exists\mathbb{R}$-complete by [AAM18], allows only equations of the form $x + y = z$, $x = 1$ and $x \cdot y = 1$ for $x, y, z \in \{x_1, ..., x_n\}$ and asks whether a given system of equations has a solution over real numbers, when each variable is restricted to the range $[\frac{1}{2}, 2]$.

Theorem 6.6 ([LMM18, Theorem 3]): *Planar-ETR-INV* is $\exists\mathbb{R}$-complete.*

Theorem 6.7: *RealTriangularDoorPlacement is $\exists\mathbb{R}$-complete.*

Proof. The membership in $\exists\mathbb{R}$ is already shown in Lemma 6.4. To prove $\exists\mathbb{R}$-hardness, we reduce from the problem Planar-ETR-INV*. Given a Planar-ETR-INV* instance I with its planar variable-constraint incidence graph, we create a RealTriangularDoorPlacement instance which admits an affirmative answer if and only if I is a YES-instance. The idea, which is similar to the idea of the reduction from Planar-ETR-INV* given in [LMM18], is to construct gadgets representing variables as well as copy and split gadgets to reuse and transport variables according to the incidence graph and gadgets to enforce addition and inversion inequalities.

In the following we explain these gadgets.

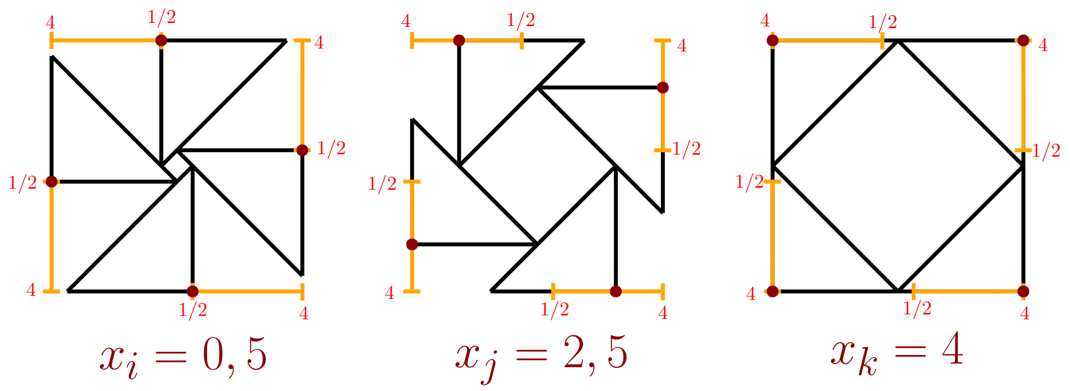

Figure 6.1: Variable gadget

Variable gadget. We represent every variable with four isosceles right triangles of cathesus length 8 units, where all the catheti are parallel to either x or y-axis. As shown in the Figure 6.1, every triangle is movable in one dimension within the drawn range $[\frac{1}{2}, 4]$, which corresponds to the range of the variables in I. The ranges are drawn according to the possible positions of the corner of every triangle where the right angle is formed. $\frac{1}{2}$ units of the drawn range corresponds to 1 unit in our drawing, hence the range $[\frac{1}{2}, 4]$ is 7 units long and every $\frac{1}{2}$ step of a range coincides with an integer coordinate in the drawing. The only possible legal placement of these four triangles is such, that all of them are touching each other creating a windmill-like structure with a square in the middle as in the Figure 6.1. Furthermore, with a thought experiment one can convince himself that if we, for example, shift the upmost triangle to the left, then the triangle on the left is forced to move down, the downmost triangle is forced to move to the right and the triangle on the right is forced to move up, all of them the same amount to maintain a legal position. Because of that and the way the ranges are given, all the four triangles are forced to take the same value in the drawn ranges. Moreover, because of the same reason, in every variable gadget the size of the square in the middle positively correlates with the value of the variable the gadget represents.

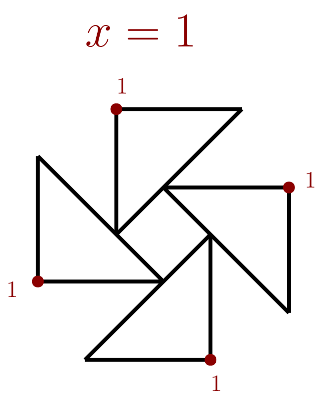

Figure 6.2: Gadget enforcing $x = 1$

$x = 1$ gadget. In order to force a variable to be equal to one, we have simply a normal variable gadget but the triangles are not allowed to move, as shown in Figure 6.2. That means, in input they have a possible maximum shift of 0 units in either direction.

Copy and split gadget. As the variables may come up in more than one constraint in Planar-ETR-INV* instance I, we need a gadget to copy them as well as a so-called split or transport gadget to use them in the constraint gadgets. Say, we want to transport a variable to the right, then we replace the triangle on the right of a variable gadget, which may move up and down, with an isosceles right triangle double its size mirrored along the vertical cathetus of the original triangle, to extend it further to the right as shown in Figure 6.3. In this way, we can draw the mirrored version of the variable gadget to the right and we can transport the variable arbitrarily far, make arbitrarily many copies by splitting up, down, right and left and make turns resembling edges of a plane rectilinear drawing of a graph. Note that the triangle connecting two blocks of variable gadgets (4 triangles each) have cathetus length of $8\sqrt{2}$ units which is equal to the hypotenuse length of one of the small triangles and the ranges for these big triangles are drawn according to the possible positions of the middle point of the triangle's hypotenuse. One can think of these big triangles as doors opening to both sides. One can convince herself that in Figure 6.3, if one shifts an arbitrary triangle, then all of the

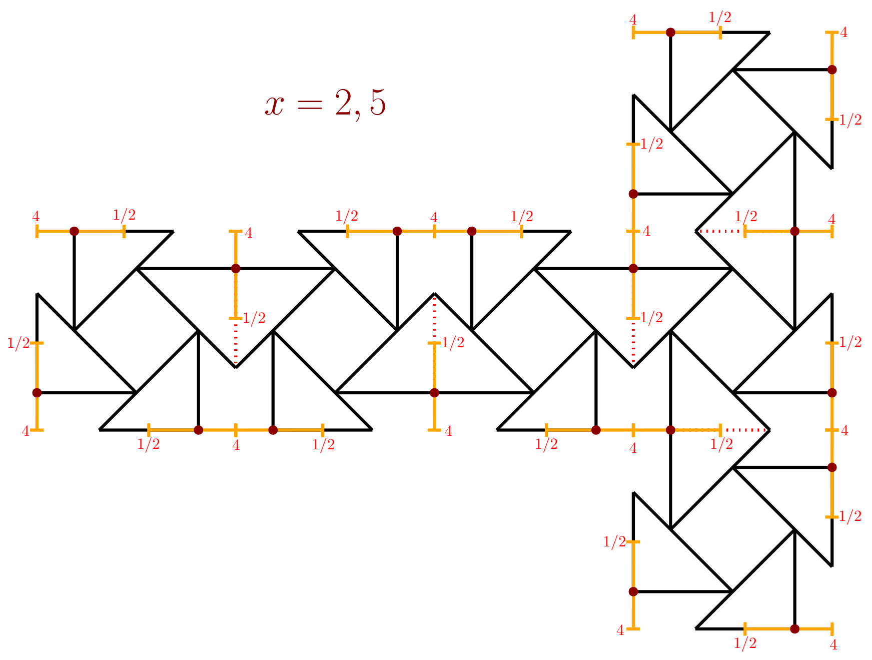

Figure 6.3: Copy and split gadget

remaining ones have to be shifted the same amount to maintain a legal position and the size of the squares in the middle regions of the blocks have always the same size with each other indicating the value of the variable which is being transported.

Addition gadgets. In order to enforce the addition constraints between three variables x, y, z; we need two gadgets, one modelling $x + y \geq z$ and one modelling $z \geq x + y$. To model the former constraint $x + y \geq z$, we use the gadget in Figure 6.4. The variables are transported to come together at the biggest triangle in the middle, which is allowed to shift in two dimensions. We can call this the addition triangle. The triangles in the variable gadgets which may touch the addition triangle are drawn bigger so that there is no conflict between variable gadgets and the addition triangle. Let us call these triangles the pusher triangles of the variable gadgets. Observe that these triangles are just scaled up versions of a normal triangle in that position of a variable gadget. In such, they serve the same functionality, meaning that they shift the same amount if the value of the variable gets bigger or smaller. They can be drawn arbitrarily bigger to resolve alignment issues between transport gadgets. When $x = \frac{1}{2}$ and $y = \frac{1}{2}$ hold, then the addition triangle must be in its downmost and leftmost position and therefore the variable gadget representing z can have the value at most 1. If the variables x and y get bigger, then the pusher triangles of the variable gadgets shift and the addition triangle can move by $y - \frac{1}{2}$ units up and by $x - \frac{1}{2}$ units to the right, which add up to $y - \frac{1}{2} + x - \frac{1}{2} = x + y - 1$ units. Then, the pusher triangle of the gadget representing z is allowed to move right by the same amount. Therefore it holds that $z \leq x + y - 1 + 1$, and our constraint is satisfied.

To model the latter constraint $z \geq x + y$, we use the gadget in Figure 6.5. This is a very similar setup to the other addition gadget, but roles of the variables are changed. Here, if the addition triangle is in its upmost and rightmost position, then the variables x and y are forced to have the value $\frac{1}{2}$ and the variable gadget representing z can have the value at least 1.

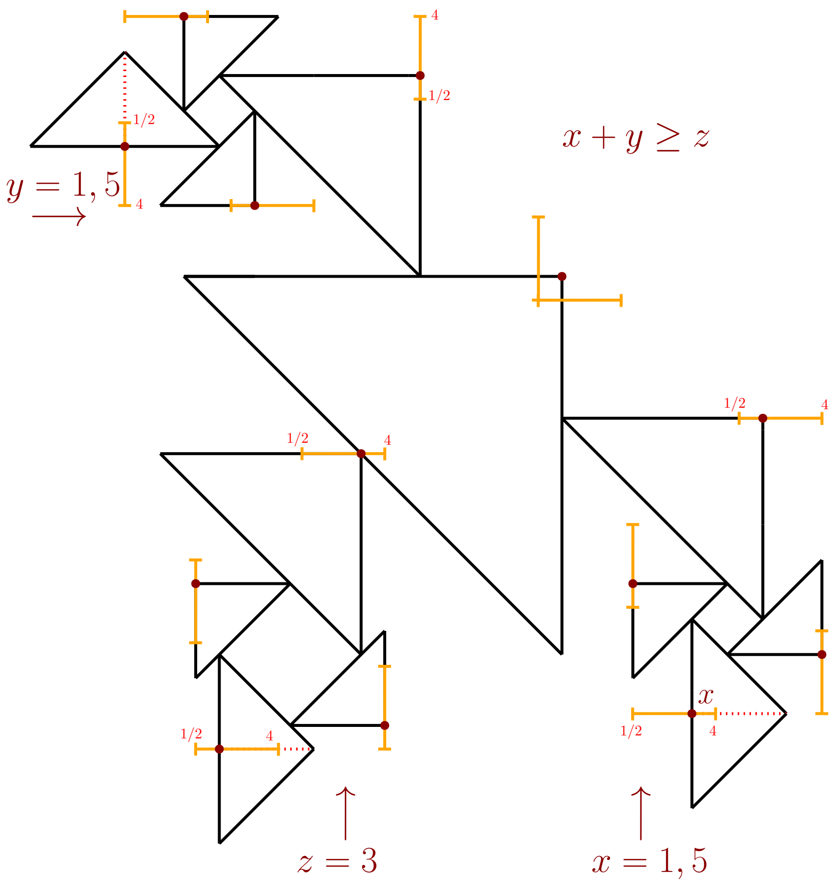

Figure 6.4: Gadget enforcing $x + y \geq z$

If z gets bigger, than the addition triangle is allowed to shift down and to the left by a total amount of $z - 1$. As a result of this, the pusher triangles of x and y can also move and thus the sum $x + y$ can have a value of at most $1 + z - 1$, thereby satisfying our constraint.

Note that we can mirror the additions gadgets vertically so that the orientation of the variables gadgets y-z-x can be both counter-clockwise and clockwise. This is important when we are reducing from the variable-constraint incidence graph.

Inversion gadgets. In order to enforce inversion constraints between two variables x and y, we need two gadgets, one modelling $x \cdot y \geq 1$ and the other modelling $x \cdot y \leq 1$.

The essential idea is the same as in [LMM18]. In both inversion gadgets, we have triangle which can be rotated around the middle point of its hypotenuse. We can call this triangle the inversion triangle. We have the ranges of the triangles which may touch the inversion triangle as shown in Figure 6.6. A line intersecting the blue fixed point and the two ranges create two triangles Δ_1 and Δ_2. Observe that the triangle Δ_1 has always cathetus lenghts x and 1, while the triangle Δ_2 has always cathetus lenghts 1 and y. As, by construction, these two triangles are always similar triangles, we have $\frac{x}{1} = \frac{1}{y}$ and therefore $x \cdot y = 1$.

In the gadget enforcing the constraint $x \cdot y \geq 1$ (Figure 6.7), the variables x and y are always forced to be above the hypotenuse of the inversion triangle, which marks the $x \cdot y = 1$ boundary, so the inequality holds. If, for example, the variable gadget representing x gets smaller, the pusher triangle of x turns the inversion triangle counter-clockwise around the fixed point and y has to get bigger to satisfy the constraint. Note that we have an additional

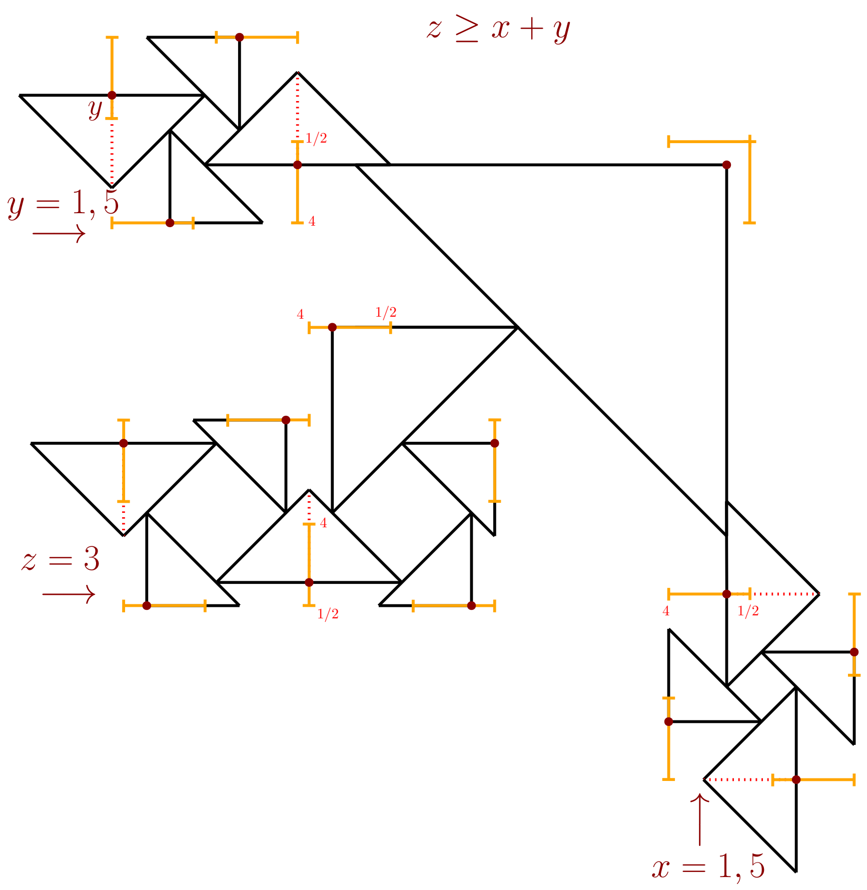

Figure 6.5: Gadget enforcing $z \geq x + y$

triangle below the big triangle in the variable gadget representing y. This has the same value in the range as triangle above it as long as it is forced to touch the inversion triangle. If y gets bigger, it is allowed to move up so as to act like the variable y.

In order to model the constraint $x \cdot y \leq 1$, we use a very similar setup (Figure 6.8). Here, instead of above, the variables x and y are always forced to be under the hypotenuse of the inversion triangle, satisfying the constraint.

Summing up the reduction. Remember that we are given a Planar-ETR-INV* instance I with variables $x_1, \ldots, x_N$, where we are promised that the variable-constraint incidence graph is planar. Note that every constraint vertex has maximum degree 3 in this graph, because every constraint includes at most 3 variables. However, a variable may come up in every constraint at worst case. Therefore, we modify this graph such that a variable vertex of degree more than 3 is splitted into vertices of degree at most 3 (Compare Figure 6.9). Therefore, there is a plane rectilinear drawing of this graph and we compute such a drawing D in polynomial time using algorithms from [NR04]. We replace every variable vertex in D with a variable gadget and replace edges with copy and split gadgets. Note that while splitting we are basically making 90 degree turns (see Figure 6.3) and transport the value of the variable. Finally we replace constraints with the corresponding addition and inversion gadgets and obtain the RealTriangularDoorPlacement instance. Note that in inversion constraints we have two variables. Therefore, the ordering of the edges at the inversion constraint vertices in D does not matter for our reduction, as we can do further turns with our copy and split gadgets and

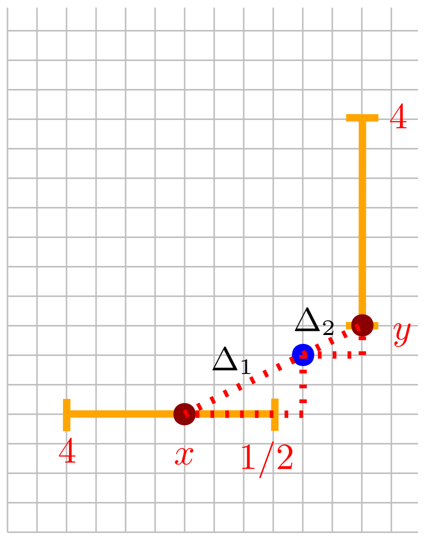

Figure 6.6: Idea of inversion gadgets

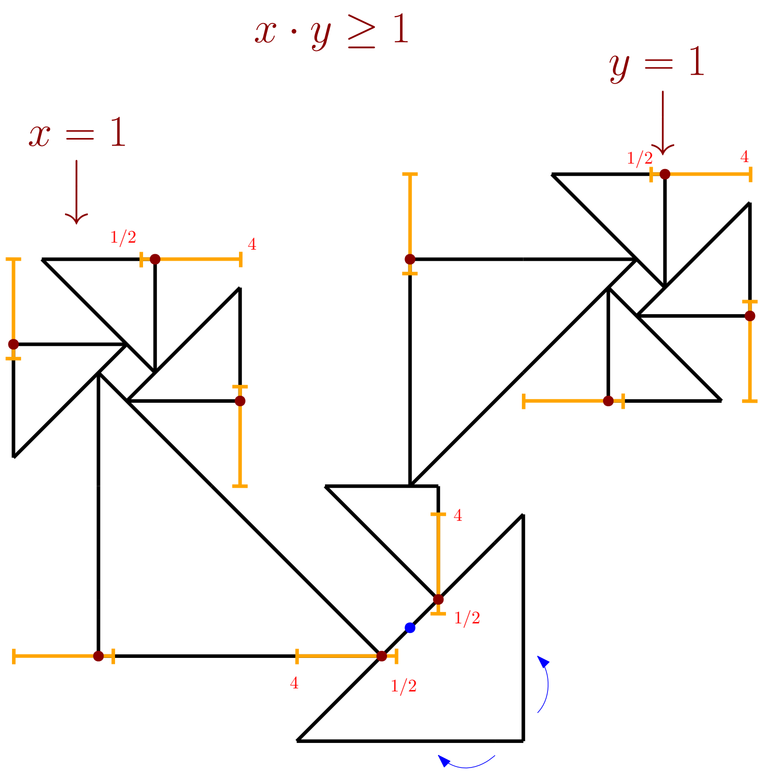

Figure 6.7: Gadget enforcing $x \cdot y \geq 1$

our inversion gadgets are indifferent to the ordering of x and y, because the multiplication operation is commutative. We have to be more careful with addition gadgets, as there the variable z has a different role from the variables x and y in our addition gadgets, whose ordering does not matter because addition is commutative. But further consideration makes it clear that this is not a problem, as we can mirror addition gadgets to have the desired ordering (counter-clockwise or clockwise) of the variable gadgets x, y, z. Also note that we draw every

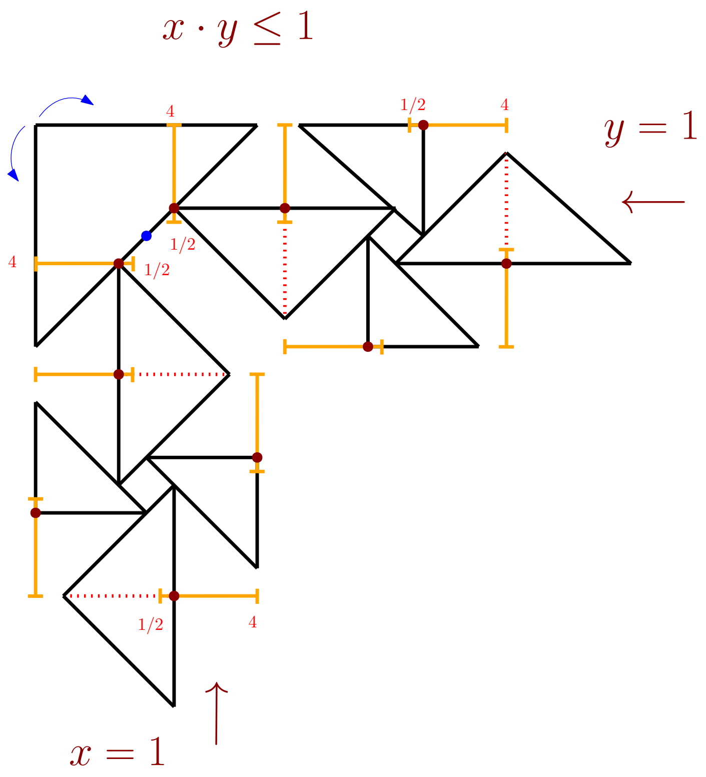

Figure 6.8: Gadget enforcing $x \cdot y \leq 1$

triangle in its downmost and leftmost position in its range to ensure that all coordinates we use in defining the reduced REALTRIANGULARDOORPLACEMENT instance I' are integer and recall that in the problem definition we allowed shifting triangles given in input, only up and to the right for the sake of simplicity of the definition. Moreover, to resolve alignment issues we can always make the pusher triangles in addition and inversion gadgets bigger, which are already drawn bigger than the normal triangles in variable gadgets. As all the gadgets have polynomial size, the reduction can be done in polynomial time.

Furthermore, if I is as YES-instance then there are real numbers $x_1, ..., x_N$ that satisfy the constraints in I. Every variable gadget in I' can be adjusted according to these values and and since the constraint gadgets simulate the constraints in I, I' is a YES-instance. Conversely, if I' is a YES-instance, one can read off the values from the variable gadgets and these are a solution for I.

■

Note that in the definition of REALTRIANGULARDOORPLACEMENT we did not require that the sets of shifting and rotating triangles are disjoint. But in the reduction we did not use any triangle which we both shift and rotate so we proved that REALTRIANGULARDOORPLACEMENT is $\exists\mathbb{R}$-complete, even when T_s and T_r are required to be disjoint.

Also note that one can generalize REALTRIANGULARDOORPLACEMENT to allow arbitrary triangles. Now, consider the case where four triangles are placed such that they enclose a rectangular region between them and these triangles cannot be shifted nor rotated. Let each of the remaining (not necessarily right isosceles) triangles be both in T_s and T_r and initially placed at the bottom left corner of this rectangular region. Furthermore, let their shifting ranges correspond to the rectangular region. So, they can be moved and rotated freely inside

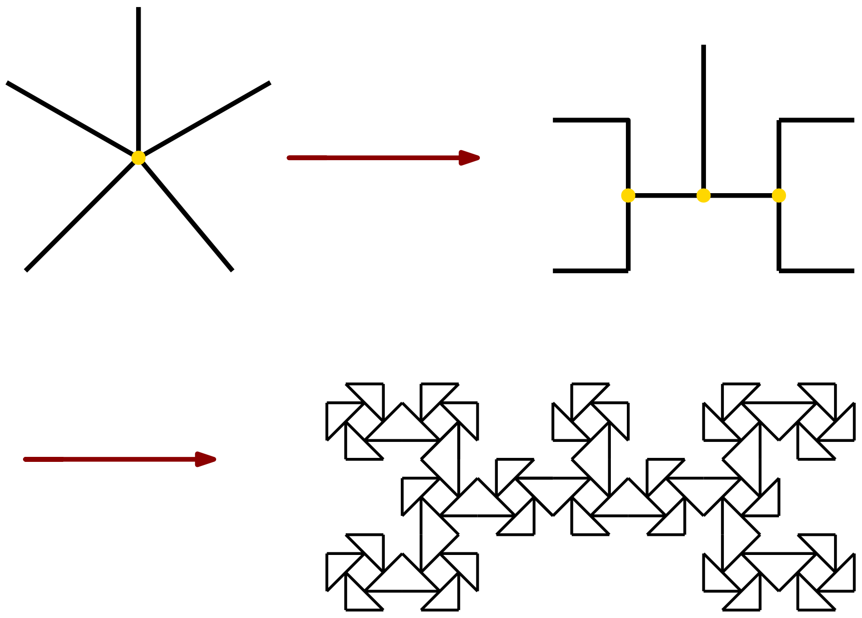

Figure 6.9: Converting variable-constraint incidence graph

the rectangular region. This is a special case of RealTriangularDoorPlacement (when we allow arbitrary triangles) and a triangle packing problem. This problem is clearly in ∃ℝ. Although packing convex polygons inside a square container is proven to be ∃ℝ-complete [AMS22], to the best of our knowledge triangle packing inside a rectangular region is only known to be **NP**-hard [Cho16].

7 Conclusion

In this thesis, we considered several types of problems related to doors. We proved that both the placement and the replacement problem are tractable for squared doors, while these problems are **NP**-complete and **PSPACE**-complete, respectively, for circular doors. In such, we realized that there is a dichotomy between squared and circular doors, caused by the number of placement options for the respective door shape. Namely, one can see a correspondence between 2-Sat and squared doors and between 3-Sat and circular doors.

We then proved that two motion planning problems where an agent can move and open doors (but not close them) in an environment with walls are **NP**-complete. One of these problems only allows pushing doors, while the other allows both pushing and pulling doors. So, in that case the ability to pull doors does not make the problem "easier". Furthermore, we simplified the motion planning gadgets from [DDHO03] and showed that the one-way gadget and the block gadget are sufficient to make a reduction from Planar 3-Coloring and thus to prove **NP**-hardness.

We also showed that a packing-like placement problem involving triangular doors and real number shifts and rotations is complete for $\exists\mathbb{R}$.

7.1 Future work

Placement and Replacement. It might be interesting to know if there are any interesting tractable door problems where we have more than two ways of placing a door. Furthermore, a more general framework could be established for determining exactly in which cases these problems are tractable/intractable.

Motion planning. The complexity of the remaining variants of door motion planning problems (see section 1.2), in particular Pull-Open-Doors and Push-Open/Close-Doors, are open questions. We suspect that Push-Open/Close-Doors could be **NP**-complete by a reduction to and from Push-Open-Doors. Namely, it could be the case that the ability to close doors does not create an advantage for the agent. One variant we did not explicitly mention, but which is interesting, is motion planning through doors that can be pushed more than 90 degrees, for example a simple single door turnstile that can be freely rotated around its hinge, where each push rotates the door 90 degrees. This is also an open problem from [Gre+21]. It might also be interesting to know if there are any **PSPACE**-complete motion planning problems with doors similar to the ones we have considered.

Problems related to $\exists\mathbb{R}$. It remains open, whether the problems RealSquaredDoorPlacement and RealCircularDoorPlacement, defined analogously to RealTriangularDoorPlacement, are also hard for $\exists\mathbb{R}$. They can be shown to be contained in $\exists\mathbb{R}$ by a similar argument to Lemma 6.4. Another open problem is, whether triangle packing inside a rectangular region is hard for $\exists\mathbb{R}$.

Bibliography

[AAM18] Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. “The art gallery problem is $\exists\mathbb{R}$-complete”. In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. Los Angeles, CA, USA: Association for Computing Machinery, 2018, pp. 65–73. ISBN: 9781450355599. DOI: *10.1145/3188745.3188868*.

[AMS22] Mikkel Abrahamsen, Tillmann Miltzow, and Nadja Seiferth. *Framework for $\exists\mathbb{R}$-Completeness of Two-Dimensional Packing Problems*. 2022. arXiv: *2004.07558*.

[Ani+20] Hayashi Ani, Jeffrey Bosboom, Erik D. Demaine, Jenny Diomidova, Dylan Hendrickson, and Jayson Lynch. *Walking through Doors is Hard, even without Staircases: Proving PSPACE-hardness via Planar Assemblies of Door Gadgets*. 2020. arXiv: *2006.01256*.

[BK10] Mark de Berg and Amirali Khosravi. “Optimal Binary Space Partitions in the Plane”. In: *Computing and Combinatorics*. Edited by My T. Thai and Sartaj Sahni. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 216–225. ISBN: 978-3-642-14031-0.

[BKM98] Therese Biedl, Michael Kaufmann, and Petra Mutzel. “Drawing Planar Partitions II: HH-Drawings”. In: *Graph-Theoretic Concepts in Computer Science*. Edited by Juraj Hromkovič and Ondrej Sýkora. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 124–136. ISBN: 978-3-540-49494-2.

[Can88] John Canny. “Some algebraic and geometric computations in PSPACE”. In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 460–467. ISBN: 0897912640. DOI: *10.1145/62212.62257*.

[CHKS10] Sergio Cabello, Herman Haverkort, Marc Kreveld, and Bettina Speckmann. “Algorithmic Aspects of Proportional Symbol Maps”. In: *Algorithmica* Volume 58 (Nov. 2010), pp. 543–565. ISSN: 0178-4617. DOI: *10.1007/s00453-009-9281-8*.

[Cho16] Amy Chou. “NP-Hard Triangle Packing Problems”. In: 2016.

[Cul97] Joseph C. Culberson. “Sokoban is PSPACE-complete”. In: 1997. DOI: *10.7939/R3JM23K33*.

[DDHO03] Erik D. Demaine, Martin L. Demaine, Michael Hoffmann, and Joseph O’Rourke. “Pushing blocks is hard”. In: *Computational Geometry* Volume 26 (2003), pp. 21–36. ISSN: 0925-7721. DOI: *10.1016/S0925-7721(02)00170-0*.

[Dem23] Erik D. Demaine. *Algorithmic Lower Bounds: Fun with Hardness Proofs*. 2023. *http://courses.csail.mit.edu/6.5440/fall23/lectures/* Accessed: 2024-03-23.

[DH08] Erik D. Demaine and Robert A. Hearn. *Playing Games with Algorithms: Algorithmic Combinatorial Game Theory*. 2008. arXiv: *cs/0106019*.

[FB02] Gary William Flake and Eric B. Baum. "Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants"". In: *Theoretical Computer Science* Volume 270 (2002), pp. 895–911. ISSN: 0304-3975. DOI: *10.1016/S0304-3975(01)00173-6*.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1979. ISBN: 0716710447.

[GKMP06] Parikshit Gopalan, Phokion G. Kolaitis, Elitza N. Maneva, and Christos H. Papadimitriou. "The Connectivity of Boolean Satisfiability: Computational and Structural Dichotomies". In: *CoRR* Volume abs/cs/0609072 (2006). arXiv: *cs/0609072*.

[Gre+21] Aster Greenblatt, Oscar Hernandez, Robert A. Hearn, Yichao Hou, Hiro Ito, Minwoo Kang, Aaron Williams, and Andrew Winslow. "Turning Around and Around: Motion Planning through Thick and Thin Turnstiles". In: *Proceedings of the 33rd Canadian Conference on Computational Geometry, CCCG 2021, August 10-12, 2021, Dalhousie University, Halifax, Nova Scotia, Canada*. Edited by Meng He and Don Sheehy. 2021, pp. 377–387.

[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". In: *Theoretical Computer Science* Volume 343 (2005), pp. 72–96. ISSN: 0304-3975. DOI: *10.1016/j.tcs.2005.05.008*.

[Knu08] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 0: Introduction to Combinatorial Algorithms and Boolean Functions (Art of Computer Programming)*. 1st ed. Addison-Wesley Professional, 2008. ISBN: 0321534964.

[Lic82] David Lichtenstein. "Planar Formulae and Their Uses". In: *SIAM Journal on Computing* Volume 11 (1982), pp. 329–343. eprint: *https://doi.org/10.1137/0211025*.

[LMM18] Anna Lubiw, Tillmann Miltzow, and Debajyoti Mondal. "The Complexity of Drawing a Graph in a Polygonal Region". In: *Graph Drawing and Network Visualization*. Edited by Therese Biedl and Andreas Kerren. Cham: Springer International Publishing, 2018, pp. 387–401. ISBN: 978-3-030-04414-5.

[Mat14] Jiri Matousek. *Intersection graphs of segments and* $\exists\mathbb{R}$. 2014. arXiv: *1406.2636*.

[NR04] Takao Nishizeki and Md Saidur Rahman. *Planar Graph Drawing*. WORLD SCIENTIFIC, 2004. eprint: *https://worldscientific.com/doi/pdf/10.1142/5648*.

[PRB16] André G. Pereira, Marcus Ritt, and Luciana S. Buriol. "Pull and PushPull are PSPACE-complete". In: *Theoretical Computer Science* Volume 628 (2016), pp. 50–61. ISSN: 0304-3975. DOI: *10.1016/j.tcs.2016.03.012*.

[Sav70] Walter J. Savitch. "Relationships between nondeterministic and deterministic tape complexities". In: *Journal of Computer and System Sciences* Volume 4 (1970), pp. 177–192. ISSN: 0022-0000. DOI: *10.1016/S0022-0000(70)80006-X*.

[Sch10] Marcus Schaefer. "Complexity of Some Geometric and Topological Problems". In: *Graph Drawing*. Edited by David Eppstein and Emden R. Gansner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 334–344. ISBN: 978-3-642-11805-0.

[UEH23] Ryuhei UEHARA. “Computational Complexity of Puzzles and Related Topics”. In: *Interdisciplinary Information Sciences* Volume 29 (2023), pp. 119–140. DOI: *10.4036/iis.2022.R.06*.

[Vig13] Giovanni Viglietta. *Partial Searchlight Scheduling is Strongly PSPACE-Complete*. 2013. arXiv: *1201.2097*.