University of Basel

Computational Complexity of Classical Planning Domains Based on Grids

Master's thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence research group
ai.dmi.unibas.ch

Examiner: Dr. Gabriele Röger
Supervisor: Augusto Blaas Corrêa

Travis Rivera Petit
travis.riverapetit@unibas.ch
2015-117-427

December 12, 2023

Acknowledgments

I am grateful to my supervisor Augusto Blaas Corrêa for his regular feedback and advice. I also thank Malte Helmert for hinting at the invariant from Lemma 8. I thank Emil Jeřábek and David Eppstein for suggesting bipartite matching as an approach to proving Corollary 3. Finally, I thank my aunt Béatrice Conde-Petit who helped me correct grammatical mistakes.

Abstract

The International Planning Competitions (IPCs) serve as a testing suite for planning systems. These domains are well-motivated as they are derived from, or possess characteristics analogous to real-life applications. In this thesis, we study the computational complexity of the plan existence and bounded plan existence decision problems of the following grid-based IPC domains: VisitAll, TERMES, Tidybot, Floortile, and Nurikabe. In all of these domains, there are one or more agents moving through a rectangular grid (potentially with obstacles) performing actions along the way. In many cases, we engineer instances that can be solved only if the movement of the agent or agents follows a Hamiltonian path or cycle in a grid graph. This gives rise to many **NP**-hardness reductions from Hamiltonian path/cycle problems on grid graphs. In the case of VisitAll and Floortile, we give necessary and sufficient conditions for deciding the plan existence problem in polynomial time. We also show that Tidybot has the game Push -1F as a special case, and its plan existence problem is thus **PSPACE**-complete. The hardness proofs in this thesis highlight hard instances of these domains. Moreover, by assigning a complexity class to each domain, researchers and practitioners can better assess the strengths and limitations of new and existing algorithms in these domains.

Table of Contents

1 Introduction

Automated planning is a broad area of artificial intelligence. Researchers in this field devise computer programs that are able to come up with strategies that solve problems in planning and scheduling. Mathematically, such problems are modeled as planning tasks. Broadly speaking, these define a set of states, an initial state, a set of goal states, and one or more actions that transform a state into a new state. A planning task is solved once a strategy has been found that reaches a goal state from the initial state (such a strategy is called a plan), or when it is shown that no such strategy exists. Once a plan has been found, it can be physically realized or executed by a real-world agent (e.g. a robot).

Classical planning is a branch of automated planning. It studies domain-independent, fully observable, discrete planning tasks. The long-term goal of this area of research is to devise efficient and reliable domain-agnostic planning algorithms. A strategy that solves a classical planning task takes the form of a certificate of unsolvability, or a sequence of actions, that, when applied in succession, transforms a given initial state into a goal state. Despite being a restricted variant of automated planning, classical planning has many applications, for instance in conformance checking, organic synthesis, and logistics [7], [20], [23].

Most classical planning algorithms try to exploit the given task's structure to significantly cut down the run time [14]. However, they sometimes underperform. There are two main reasons for this. First, since they are domain-independent, they may not always correctly capture the inherent structure of the given task, so they could for example fail to identify dead-ends early or fail to take advantage of symmetries. Moreover, many algorithms and heuristics are sensitive to reformulations. Hence, the amount of time spent looking for a plan can vary when using distinct yet equivalent descriptions of the same planning task [12]. Second, while it is true that (propositional) classical planning is known to be **PSPACE**-complete [5], some domains are simpler than others. Unfortunately, the computational complexity of many domains is not well understood, since there are so many of them, and new ones emerge every year. It is often hard to diagnose which of these two is the underlying reason for bad performance. Domains can have strange and novel regularities that most planners fail to identify. A planning task may be hard to solve even when there are only a few actions that can be applied in each state, and even when it consists of sub-problems that are easy to solve in isolation. In particular, hard problems often have dependencies in such a way that local progress cannot be extended to global progress, meaning that it is possible to solve one sub-problem in such a way that the next sub-problem can only be solved by un-doing progress in the previous one. This severely punishes greedy algorithms, whose search is guided by inappropriate simplifications of the state space.

Understanding the complexity of planning domains has many advantages. First, practitioners can use this information to make a more informed decision on which algorithm to use when solving a particular instance. Second, it helps researchers understand the strengths and limitations of their algorithms on hard and easy problems. This is of special importance when the domains in question are used in benchmark planning competitions since they are used to compare and showcase new ideas among researchers in classical planning. Third, it helps in identifying the problem formulation mentioned before. Finally, by assigning a complexity class a number of the domains, we help to expand the list of problems in the complexity classes **P**, **NP**, **NP**-complete, **PSPACE**, and **PSPACE**-complete.

The previous paragraph motivates this thesis. We study five IPC planning domains from the theoretical point of view of computational complexity theory, namely VisitAll, TERMES, Tidybot, Floortile, and Nurikabe. For each domain, we assign a complexity class to the plan existence and bounded plan existence problem. We also do a literature review on the domains Snake, Ricochet Robots, and Labyrinth. The PDDL files of each domain are freely available online[1].

1.1 Background

In this section, we provide the necessary background: computational complexity, and classical planning. Also, since we analyze domains of classical planning based on grids, we go over the types of grids that occur in these domains. Additionally, since many of our **NP** hardness reductions stem from Hamiltonian path and cycle problems, we devote an additional section to these.

1.1.1 Computational Complexity Theory

Computational complexity theory is a branch of theoretical computer science that studies the amount of resources, like time and memory, needed to solve computational problems regardless of the technology used. In the following two paragraphs, we distinguish the terms "decision problem" and "instance". We borrow and expand on the definitions from Garey et. al. [11].

Conceptually, a decision problem is the blueprint or description of a computational task whose output is "yes" or "no". An instance is a concrete realization matching that description. As a running example, we take the decision problem of deciding whether graphs are connected. We call this decision problem `CONNECTED`. An instance of `CONNECTED` fixes a (finite) graph and asks if it consists of a single connected component.

[1] As of the writing of this thesis, the PDDL files for the domains VisitAll, Tidybot, Floorile, and Nurikabe can be found under https://editor.planning.domains/, and the PDDL files for the domains Snake, Ricochet Robots, and Snake can be found under https://github.com/ipc2023-classical/ipc2023-dataset.

Formally, a decision problem $\Pi = (D_\Pi, Y_\Pi)$ is an ordered pair where D_Π is a set of instances and $Y_\Pi \subseteq D_\Pi$ is the set of instances whose output is "yes". Instances are words of an encoding language that describe computational problems. Thus, elements of D_Π are descriptions of computational tasks. For each word $w \in D_\Pi$ we let $\|w\|$ denote its length. Coming back to the previous example of `CONNECTED`, we can encode graphs through their adjacency matrices. To do this we let $D_{\texttt{CONNECTED}} \subset \{0, 1, \texttt{newline}\}^*$ be the formal language that describes all well-defined adjacency matrices, and $Y_{\texttt{CONNECTED}} \subset D_{\texttt{CONNECTED}}$ be such that it corresponds to the adjacency matrices of connected graphs.
Most of the time, instances with long descriptions will require more computational effort. For example, in most cases, an instance of `CONNECTED` that fixes a graph with 10^{10} vertices will spend more computational resources than an instance with 10 vertices. Computational complexity theory gives us a framework with which we can measure how the resources needed to solve an instance I scales relative to $\|I\|$.
Describing a decision problem formally can be tedious for the reader, it is also oftentimes un-insightful. Therefore we adopt the convention of defining decision problems in the canonical **INPUT** + **QUESTION** manner, where the **INPUT** provides one or many generic uninstantiated mathematical objects (a function, set, ordered pair, etc.), and **QUESTION** asks if the object admits some property. In this case, the length of an instance is measured in a way that reflects the size of the object (cardinality, size of function domain, number of vertices and edges in a graph, etc.). The decision problem `CONNECTED` could thus be posed as follows:

`CONNECTED`
INPUT: A finite graph $G = (V, E)$
QUESTION: Is G connected?

We close off this section by explaining a few more notions that are of particular importance in computational complexity: classes, membership to a class, hardness, completeness, and reductions.
Conceptually, a complexity class $\mathcal{C}$ defines an amount of resources r and a model of computation m. A decision problem Π belongs to a class $\mathcal{C}$ if there exists an algorithm that runs on m and can successfully decide any instance I of Π subject to using at most $r(\|I\|)$ resources. Formally, $\mathcal{C}$ is the set of all decision problems that can be decided under the restrictions mentioned above.
The complexity classes we work with in this thesis are **P**, **NP**, **PSPACE**, and **NPSPACE**. The model of computation for **P** and **PSPACE** are deterministic one-tape Turing Machines. **NP** and **NPSPACE** use the non-deterministic Turing Machine model. The resource bound set by **P** and **NP** is that of running polynomial time. On the other hand, **PSPACE** and **NPSPACE** require their problems to be solved with a polynomial amount of space. It holds that $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE}$. It is conjectured that these inclusions are proper. The fact that $\mathbf{PSPACE} = \mathbf{NPSPACE}$ stems from Savitch's Theorem [24]. If $\mathbf{P} = \mathbf{NP}$ it would imply that many currently intractable problems could be solved in polynomial time, and if $\mathbf{NP} = \mathbf{PSPACE}$ then verifying the correctness of candidate solutions for hard intractable problems could be done efficiently.
If $(D_\Pi, Y_\Pi) \in \mathbf{NP}$, then a non-deterministic algorithm can decide if an instance $I \in D_\Pi \in Y_\Pi$

in polynomial time. In particular, when this algorithm runs, each branch of the non-deterministic Turing Machine guesses a candidate certificate that can be verified in polynomial time by a deterministic Turing machine. This certificate acts as proof that the instance I of the decision problem Π has a solution.

Reductions are a key notion in computational complexity. Often an instance I of a problem Π can be manipulated such that it becomes a new instance I' of Π' in a manner where $I \in Y_\Pi$ iff $I' \in Y_{\Pi'}$. If this transformation can be computed efficiently, this tells us that devising an efficient algorithm for Π' is at least as hard as devising one for Π. We say that Π can be reduced to Π', written $\Pi \leq \Pi'$ if there exists a function $f : D_\Pi \rightarrow D_{\Pi'}$ that runs in polynomial time with regards to each $\|I\|$ and satisfies $I \in Y_\Pi \iff f(I) \in Y_{\Pi'}$. If every problem in a class $\mathcal{C}$ can be reduced to Π, we say that Π is $\mathcal{C}$-hard. On the other hand, Π is said to be $\mathcal{C}$-complete if it is both $\mathcal{C}$-hard and $\Pi \in \mathcal{C}$. This implies that a $\mathcal{C}$-complete problem is not only as hard as any problem in the class but is also a representative problem for the class. Solving a $\mathcal{C}$-complete problem efficiently would enable us to solve all problems in the complexity class $\mathcal{C}$ efficiently as well.

1.1.2 Classical Planning

Classical planning, as a field of AI, revolves around the problem of finding a sequence of actions that can lead an agent from an initial state to a goal state within a formal model of the world. The environment can be modeled with a number of formal systems. In this thesis, we deal with propositional planning, i.e. planners whose models are formalized with propositional logic. This model consists of a finite set of states, actions that map states to new states, and descriptions of the initial state and goal states. We start by formally defining planning task models and transition graphs, which then give rise to a formal definition of domains. We borrow the definitions from Helmert [13].

Definition 1 (Planning task state model, transition graph)**.** *A planning task state model is a tuple $\mathcal{M} = (S, s_0, S_G, A)$ comprised of*

- *a finite set of states S,*
- *the initial state $s_0 \in S$,*
- *the set of goal-states $S_G \subseteq S$,*
- *a finite set of actions A. An action is a partial function from S to S.*

A planning task model $\mathcal{M}$ induces a transition graph $T(\mathcal{M})$, an edge-labeled directed finite graph whose vertex set is S with arcs $s \xrightarrow{a} a(s)$ for all pairs (a, s) of actions and states for which $a(s)$ is well defined.

Definition 2 (planning domain, planning task)**.** *A planning domain $\mathcal{D}$ is a map that assigns words in an encoding language $\mathcal{E}$ to planning task state models. A word $T \in \mathcal{E}$ is called a planning task of $\mathcal{D}$.*

Definition 3 (plan, plan length)**.** *Let Π be a planning task of a domain $\mathcal{D}$ and $\mathcal{M} = \mathcal{D}(\Pi) = (S, s_0, s_G, A)$ be the planning task state model of Π. A plan for Π is a finite sequence of* *actions $\pi = \langle a_1, \ldots, a_n \rangle$ where $a_i \in A, i = 1, \ldots n$ and the composition $(a_n \circ \ldots \circ a_1)(s_0)$ is well defined and belongs to S_G.*

The previous definitions show that a task in classical planning can be thought of as a labeled, directed, finite graph, where each node n represents a state, and there exists an arc $n \xrightarrow{a} n'$ if applying action a to n results in the state n'. Hence, at its core, planning algorithms search for directed paths in large graphs. This is often done implicitly, since due to a large number of states, classical planning tasks and domains tend to be encoded using logic-based languages such as STRIPS and PDDL.
In PDDL, domains are defined through custom types, predicates of arbitrary arity, and actions. Each action is defined in terms of its precondition and effect. The precondition of an action a is a logical formula that, when evaluated to true, can be applied in a given state. The effect of an action describes how the current state changes after applying a, thus yielding a new state. All of the domains covered in this thesis are originally defined in PDDL. However, in each chapter, we describe states, preconditions, and effects in mathematical English.
In classical planning, researchers often focus on two distinct classes of planning algorithms: satisficing and optimal planning algorithms. The former aim to find any plan, while the latter search only for plans of minimal length or minimal cost.
There exist natural formulations for satisficing and optimal planning as decision problems. We describe them below:

PE−$\mathcal{D}$
INPUT: A planning task Π of a domain $\mathcal{D}$.
QUESTION: Does there exist a plan for Π?

BPE−$\mathcal{D}$
INPUT: A planning task Π of a domain $\mathcal{D}$ and a positive integer N.
QUESTION: Does there exist a plan for Π of length $\leq N$?

Here PE stands for "plan existence", while BPE stands for "bounded plan existence".A well known result is that if $\mathcal{D}$ is arbitrary, then both PE−$\mathcal{D}$ and BPE−$\mathcal{D}$ are **PSPACE**-complete [5]. However, there exist planning domains for which, PE and BPE are in **NP**-complete or are solvable in polynomial time [13].

1.1.3 Grid graphs

We present an overview of the various grid types that arise in the domains of classical planning whose complexity we study by using graph theory.

Definition 4 (graph, simple graph)**.** *A graph is a tuple (V, E) where V is a finite or countable set and E is a binary symmetric relation on V, i.e. E is a subset of $V \times V$ satisfying $(u, v) \in E \iff (v, u) \in E$ for all $u, v \in V$. If V is finite, nonempty, and E contains no self-loops we say that the graph is simple.*

For grid graphs, rectangular and square graphs we follow the definitions of Itai et. al. [17]. For solid grid graphs, we follow De Biasi et. al. [3], and for subgrid graphs, we follow Buro [4].

Definition 5 (grid graph)**.** *Let $G^\infty = (V, E)$ be the the integer lattice, i.e. $V = \mathbb{Z} \times \mathbb{Z}$ and $(\mathbf{x}, \mathbf{y}) \in E \iff d_{\ell_2}(\mathbf{x}, \mathbf{y}) = 1$. A grid graph is a simple, node-induced, connected subgraph of G^∞.*

Definition 6 (subgrid graph)**.** *A subgrid graph is a connected edge-induced subgraph of a grid graph.*

Note that all grid graphs are also subgrid graphs, yet the converse is not true, since subgrid graphs allow for vertices that are at ℓ_2 distance 1 to be non-adjacent.

Definition 7 (rectangular graph, squared graph)**.** *A rectangular graph $G = R(m, n)$ is a grid graph whose vertex set is $\{1, \ldots, m\} \times \{1, \ldots, n\}$ for some natural numbers m and n. A rectangular graph is squared if $m = n$.*

Definition 8 (solid grid graph)**.** *A grid graph $G = (V, E)$ is solid if each point $p \in \mathbb{Z} \times \mathbb{Z}$ that is not in V lies in the outer face of G.*

Geometrically, solid grid graphs are grid graphs that do not admit any "holes", i.e. all of their inner faces have the same area. Rectangular grid graphs are also solid. Figure 1.1 shows examples of grid graphs. For a vertex u in a (sub) grid graph, we use the notation (u_x, u_y) for its x and y coordinates in the plane. Without loss of generality, we always assume that the vertices at the left-most column of a (sub) grid graph have an x coordinate of one. Similarly, we assume that vertices at the bottom-most row of any grid graph have a y coordinate of one.

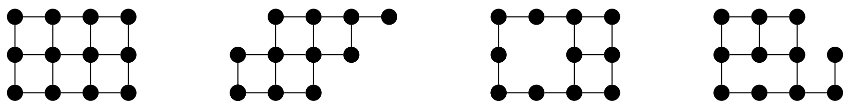


Figure 1.1: From left to right: a rectangular grid graph, a solid grid graph, a grid graph that is neither rectangular nor solid, a subgrid graph that is not a grid graph.

Definition 9 (graph width, graph height)**.** *Let $G = (V, E)$ be a grid graph or a subgrid graph. We define* **width**(G) *to be the value of the x coordinate of a right-most vertex in G minus one,* **height**(G) *is defined analogously. Formally*

$$\mathbf{width}(G) = \max\{n \in \mathbb{N} : \exists u \in V : u_x = n\}$$

$$\mathbf{height}(G) = \max\{n \in \mathbb{N} : \exists u \in V : u_y = n\}$$

1.1.4 Hamiltonian paths and cycles

Hamiltonian paths and cycles are key topics in this thesis, as many of our hardness reductions feature reductions from Hamiltonian paths or cycle problems in grid graphs. Given a finite graph G, a Hamiltonian path in G is a simple path that visits each vertex. An s-t Hamiltonian path is a Hamiltonian path where s and t are the first and last nodes visited respectively. Moreover, if s and t are neighbors, the path is called a Hamiltonian cycle. Deciding whether a graph admits a Hamiltonian cycle is computationally challenging, even if the graphs in question are grid graphs or subgrid graphs. We define the following decision problems

DEG-k-GHC
INPUT: A grid graph $G = (V, E)$ such that $\deg(u) \leq k \; \forall u \in V$.
QUESTION: Is there a Hamiltonian cycle in G?

DEG-k-GHP
INPUT: A grid graph $G = (V, E)$ such that $\deg(u) \leq k \; \forall u \in V$ and $s \neq t \in V$.
QUESTION: Is there an s-t Hamiltonian path in G?

DEG-k-SHC
INPUT: A subgrid graph $G = (V, E)$ such that $\deg(u) \leq k \; \forall u \in V$.
QUESTION: Is there a Hamiltonian cycle in G?

DEG-k-SHP
INPUT: A subgrid graph $G = (V, E)$ such that $\deg(u) \leq k \; \forall u \in V$ and $s \neq t \in V$.
QUESTION: Is there an s-t Hamiltonian path in G?

Here `GHC`, `GHP`, `SHC`, `SHP` stand for grid Hamiltonian cycle, grid Hamiltonian path, subgrid Hamiltonian cycle and subgrid Hamiltonian path. The following results stem from Itai et. al., Papadimitriou, and Buro respectively [17], [22], [4].

Theorem 1. *`DEG−4−GHC` and `DEG−4−GHP` are **NP**-complete.*

Theorem 2. *`DEG−3−GHP` is **NP**-complete.*

Theorem 3. *`DEG−3−SHC` is **NP**-complete.*

The three previous theorems are used in **NP**-hardness reductions in the next chapters.

2
VisitAll

In the VisitAll domain, there is an agent who can move in the four cardinal directions one cell at a time and starts off in the middle of a square grid graph. The goal is achieved when a preselected set of cells is visited. Some planning instances of VisitAll define the set of preselected cells to be the full grid. We call those instances VISITALL, and all other instances VISITSOME. This chapter is devoted to the decision problem counterparts of these two instances, namely PE-VISITALL, PE-VISITSOME, BPE-VISITALL, BPE-VISITSOME. We show that the first three are in **P** and that BPE-VISITSOME is **NP**-complete.

2.1 PE-VISITALL & PE-VISITSOME are trivial

We start by defining the decision problems:

PE-VISITALL
INPUT: A square graph $G = R(n, n)$ of side length n.
QUESTION: Is there a path in G that starts at the cell $(\lfloor n/2 \rfloor, \lfloor n/2 \rfloor)$ and visits every element of V at least once?

PE-VISITSOME
INPUT: A square graph $G = R(n, n)$ of side length n, a set of cells $S \subseteq V$.
QUESTION: Is there a path in G that starts at the cell $(\lfloor n/2 \rfloor, \lfloor n/2 \rfloor)$ and visits every element of S at least once?

We note how since grid graphs are connected by definition, there always exist paths that visit any subset of their vertices. Hence, the answer to any instance of PE-VISITALL and PE-VISITSOME is always "YES". In particular, they are decidable in polynomial time.

2.2 BPE-VISITALL is in **P**

This decision problem is formulated below:

BPE-VISITALL
INPUT: A square graph $G = R(n, n)$ of side length n and a positive integer $K > 0$.
QUESTION: Is there a path in G of length at most K that starts at the cell $(\lfloor n/2 \rfloor, \lfloor n/2 \rfloor)$ and visits every element of V at least once?

If $K = |V|$, `BPE-VISITALL` asks whether an $n \times n$ square graph has a Hamiltonian path that starts in the cell $(\lfloor n/2 \rfloor, \lfloor n/2 \rfloor)$. We show that for $n \geq 2$, this is always the case. The definitions 10, 11, 12 and Theorem 4 are taken from Itai et. al. [17].

Definition 10 (vertex parity)**.** *Let $G = (V, E)$ be a grid graph. The parity of a vertex $u \in V$ is even if $u_x + u_y \equiv 0 \ (mod\ 2)$. The parity of u is odd if it is not even.*

Definition 11 (color compatible)**.** *Let $G = (V, E)$ be a grid graph and $s \neq t \in V$. Write V as the disjoint union $\mathcal{E} \cup \mathcal{O}$ where $\mathcal{E}$ is the set of even vertices and $\mathcal{O}$ is the set of odd vertices. We say that (G, s, t) is color compatible if one of the following conditions hold:*

- *$|\mathcal{O}| = |\mathcal{E}|$ and s, t have different parity.*
- *$|\mathcal{O}| + 1 = |\mathcal{E}|$ and s, t are both even.*
- *$|\mathcal{O}| = |\mathcal{E}| + 1$ and s, t are both odd.*

Definition 12 (forbidden instance)**.** *Let $G = R(m, n)$ be a rectangular grid and $s \neq t$ be vertices in V. The tuple (G, s, t) is forbidden if one of the following conditions holds:*

1. *If $m = 1$ or $n = 1$, then and either s or t is not in a corner of G.*
2. *If $m = 2$ or $n = 2$, and s, t are neighbors, then the edge (s, t) is not on the outermost face of G.*
3. *If $m = 3$ and n is even, or if $n = 3$ and m is even, and if the parity of s differs from the parity of the left corners of G and from t, then $s_x < t_x - 1$ or $s_y = 2 \wedge s_x < t_x$.*

Theorem 4. *A rectangular graph G has an s-t Hamiltonian path iff (G, s, t) is color compatible and not forbidden.*

Using this result, the following Lemma is easy to prove.

Lemma 1. *Let $G = R(n, n), n \geq 2$ be a square graph and s be a vertex in G. Then there exists an s-t Hamiltonian path in G for some vertex t.*

Proof. Let $\mathcal{E}$, $\mathcal{O}$ be the sets of even and odd vertices of G. If $|\mathcal{E}| = |\mathcal{O}|$, then choose t to be any vertex whose parity is different from the parity of s. Otherwise, choose t to be any vertex to be any vertex whose parity equals the parity of s. It is easy to see that (G, s, t) is then color compatible and not forbidden. And thus, by Theorem 4, there is an s-t Hamiltonian path in G. □

Lemma 1 can be used to decide `BPE-VISITALL` in polynomial time by simply checking whether $K \geq |V| - 1$. Indeed, `BPE-VISITALL` is equivalent to asking whether a Hamiltonian path exists in a square grid starting at a specific vertex s.

2.3 BPE-VISITSOME is **NP**-complete

This decision problem is formulated below:

BPE-VISITSOME
INPUT: A square graph $G = R(n, n)$ of side length n, a set of cells $S \subseteq V$ and a positive integer $K > 0$.
QUESTION: Is there a path in G of length at most K that starts at the cell $(\lfloor n/2 \rfloor, \lfloor n/2 \rfloor)$ and passes through every element of S at least once?

We prove **NP** hardness through the following reductions

$$\texttt{DEG-4-GHC} \leq \texttt{DEG-4-GHP}_1 \leq \texttt{BPE-VISITSOME}$$

where

DEG-4-GHP$_i$
INPUT: A grid graph G and two vertices $s \neq t$ in G s.t. $\deg(t) \leq i$.
QUESTION: Does there exist an s-t Hamiltonian path in G?

We recall that by Theorem 1 DEG-4-GHC is **NP**-complete.

Lemma 2. *DEG-4-GHC $\leq$ DEG-4-GHP$_1$.*

Proof. Let $G = (V, E)$ be a grid graph with $|V| \geq 2$ and v be the right-most top vertex of V. The vertex v must exist because V is not empty. We have $\deg(v) \leq 2$ since otherwise, v would not be a corner vertex. If $\deg(v) = 1$ then there cannot be a Hamiltonian cycle in G so we may assume $\deg(v) = 2$. Given this, and that v maximizes the y coordinate of the right-most column of G we get that v's bottom neighbor $s = (v_x, v_{y-1})$ also exists. Let G' be the graph that results when adding a new vertex $t = (v_{x+1}, v_y)$ to the right of v i.e. $G' = \Big(V \cup \{t\}, E \cup \{(v,t),(t,v)\}\Big)$. An example of the construction of G' is shown in Figure 2.1.
A Hamiltonian cycle can be seen as a Hamiltonian path that starts at any vertex and ends at one of its neighbors. Thus, G has a Hamiltonian Cycle iff it has an s-v Hamiltonian Path. Next, because t is a neighbor of v with degree one, G has an s-v Hamiltonian Path iff G' has an s-t Hamiltonian path.

□

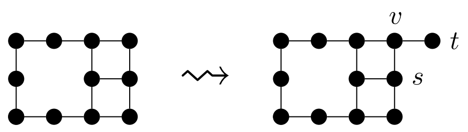


Figure 2.1: Transformation of the original grid graph on the left G to a modified grid graph G' on the right.

Lemma 3. *BPE-VISITALL is **NP**-complete.*

Proof. Membership to **NP** is straightforward since a non-deterministic algorithm can check whether a given path visits each element in S and has length at most K in polynomial time.
We show the reduction

$$\texttt{DEG-4-GHP}_1 \leq \texttt{BPE-VISITSOME}$$

Let (G, s, t) be a $\texttt{GHP}_1$ instance where V, E are the vertex and edge set of G. Let $n = \max\{\texttt{width}(G), \texttt{height}(G)\} \leq |V|$. We may assume that $s = (\lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor)$ since otherwise we may relabel the entire graph by translating each vertex through the extension the linear map $s \mapsto (\lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor)$ to each $v \in V$. Further, let $K = |V| - 1, S = V \setminus \{s\}$ and $G' = R(n, n)$. We claim that G has an s-t Hamiltonian path iff (G', S, K) is a YES-instance of BPE-VISITSOME. This construction clearly runs in polynomial time. We first argue in the direction '$\implies$'. Assume that there exists a Hamiltonian path P in G that starts in s and ends in t. The path P induces a path P' in G' of length $\|P'\| = \|P\| = |V|$, that can be solved by moving the robot $|V| - 1 = K$ times along the path P'. Since P is a Hamiltonian path, P' visits each element of $V \setminus \{s\} = S$. Now we argue in the direction '$\impliedby$'. Assume that P' is a path in G' of length $\leq K$ that visits each of element in S and starts in s. Since $\|P'\| = |S|$, P' must visit a new $p \in S$ on each move, this corresponds to a Hamiltonian path in G that starts in s. Also, P' must end in t because it has degree one. □

2.4 Representation matters

The decision problems we worked on in this chapter could also be parameterized more compactly. Instead of taking a square graph $R(n, n)$ as input, it suffices to provide n. Under this representation, our **NP** completeness proof of BPE-VISITALL would no longer be valid, unless n is encoded in base one. This is because with this parametrization, optimal plans of instances I of VISITSOME and VISITALL can be exponential in length with regards to $\|I\|$. To see why this is true we can fix an $n \in \mathbb{N}$ and let $s_n = (\lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor), t_n = s + (0, \lfloor \frac{n}{2} \rfloor)$, a path from s_n to t_n has length $\lfloor \frac{n}{2} \rfloor$. If n is represented in base $b \neq 1$, then $\left\| \lfloor \frac{n}{2} \rfloor \right\| = b^{\|n\|}$, yet $\lfloor \frac{n}{2} \rfloor = O(\|R(n, n)\|)$.

3 Floortile

In the Floortile domain, a number of robots must paint a pattern in a rectangular grid using white and black paint. Robots can only paint in the cells directly above or below them. They cannot step on cells that are already colored. Floortile is the only domain in this thesis that features non-uniform action costs.

Definition 13 (Floortile state, initial state)**.** *A Floortile state s is a tuple $s = (G, \texttt{Robots}_s, \texttt{robots\_pos}_s, \texttt{painted}_s, \texttt{avail\_color}_s)$ where*

- *$G = R(m, n)$ is a rectangular grid for some $m, n \in \mathbb{N}$.*
- *$\texttt{Robots}_s = \{\texttt{robot}_1, \ldots, \texttt{robot}_N\}$ for some $N \in \mathbb{N}$ is the set of robots.*
- *$\texttt{robots\_pos}_s : \texttt{Robots}_s \to V$ is a map that assigns a vertex to each robot.*
- *$\texttt{painted}_s : V \to \{\texttt{white}, \texttt{black}, \texttt{none}\}$ describes the color of each cell.*
- *$\texttt{avail\_color}_s : \texttt{Robots} \to \{\texttt{white}, \texttt{black}\}$ is used to describe what color each robot can use to paint.*

All initial states s_0 of Floortile are such that $\texttt{painted}_{s_0}(u) = \texttt{none}\ \forall u \in V$. Furthermore, all initial states contain at least one robot.

Definition 14 (colored cell, clear cell)**.** *Let s be a Floortile state and (V, E) be the grid of s. A cell $u \in V$ is clear in s if $\texttt{painted}_s = \texttt{none}$ and if no robot occupies u. A cell $u \in V$ is colored if $\texttt{painted}_s(u) \neq \texttt{none}$.*

The actions, costs, and preconditions for the Floortile are given in Tables 3.1 and 3.2. We note that there are no actions that modify the grid. Thus, for a Tidybot state s, we refer to its grid with G instead of G_s. The decision problems for Floortile are given below:

PE-FLOORTILE
INPUT: An initial Floortile state s_0 such that $\texttt{painted}_{s_0}(u) = \texttt{none}\ \forall u \in V$, and a goal $\texttt{goal} : V \to \{\texttt{none}, \texttt{white}, \texttt{black}\}$.
QUESTION: Is there a sequence of actions π applicable on s_0 such that $\texttt{painted}_{s_0[\pi]} = \texttt{goal}$?

BPE-FLOORTILE
INPUT: An initial Floortile state s_0 such that $\texttt{painted}_{s_0}(u) = \texttt{none}\ \forall u \in V$, a goal $\texttt{goal} : V \to \{\texttt{none}, \texttt{white}, \texttt{black}\}$ and a positive inter K.
QUESTION: Is there a sequence of actions π applicable on s_0 such that $\texttt{painted}_{s_0[\pi]} = \texttt{goal}$ and $\texttt{cost}(\pi) \leq K$?

3.1 PE-FLOORTILE is in **P**

We give necessary and sufficient conditions for the solvability of any instance of PE-FLOORTILE. These conditions can be checked in polynomial time.

Proposition 1. Let $(s_0, \texttt{goal})$ be a PE-FLOORTILE instance and s be a state reachable from s. Then $\forall\ \texttt{robot}_i \in \texttt{Robots} : \texttt{painted}_s(\texttt{robots\_pos}_s(\texttt{robot}_i)) = \texttt{none}$.

Proof. Let $G = (V, E)$ be the grid of s_0. We divide the proof into two parts:

1. The claim holds for s_0.
2. If the claim holds for a state s, then it also holds for $s[a]$ where a is an action applicable in s.

- Proof of (1):
  Follows from the fact that $\{u : \texttt{painted}_{s_0}(u) = \texttt{none}\} = V$.
- Proof of (2):
  Assume the claim holds for s. Let a be applicable in s. If a is a movement action, i.e. if it moves a robot from u to v then the claim holds for $s[a]$ since a has the precondition that $\texttt{painted}_s(v) = \texttt{none}$ and a's effect does not alter the $\texttt{painted}$ function. Next, if $a \in \{\texttt{CHANGE\_TO\_BLACK}(\cdot), \texttt{CHANGE\_TO\_WHITE}(\cdot)\}$ then the claim also holds since these actions do not change the robot's positioning nor the $\texttt{painted}$ function. Finally, if $a \in \{\texttt{PAINT\_UP}(\cdot), \texttt{PAINT\_DOWN}(\cdot)\}$ then the claim would not hold if a cell is painted on which a robot is on, but both paint actions have the precondition that the target cell must be empty.

□

Definition 15 (hole)**.** *Let $I = (s_0, \texttt{goal})$ be a PE-FLOORTILE instance. We say a vertex $u \in V$ is a hole if $\texttt{goal}(u) = \texttt{none}$.*

The following result gives necessary conditions for an instance of Tidybot to have a plan.

Lemma 4. *Let $I = (s_0, \texttt{goal})$ be a solvable PE-FLOORTILE instance with N robots and H holes. Then $H \geq N$ and on each column of G contains a hole.*

Proof. Let $G = (V, E)$ be the grid of s_0. We divide the proof into two parts, namely

a. I is solvable $\implies H \geq N$
b. I is solvable $\implies$ on each column of G there exists a vertex u with $\texttt{goal}(u) = \texttt{none}$.

- Proof of (a):
  Assume by contradiction that I is solvable and $H < N$. Let s be a goal state, then by Proposition 1, we have that at least N vertices u are such that $\texttt{goal}(u) = \texttt{none}$ which is impossible.

- Proof of (b):
  Assume by contradiction that I is solvable and that there exists a column C in G such that for all vertices u in that column it holds that $\texttt{goal}(u) \neq \texttt{none}$. Let $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ be a plan for I and u be the last cell that is painted in C in the plan π. Let a be the action that paints u. If $a = \texttt{PAINT\_UP}(\cdot)$ then the cell above u must be clear, which is impossible since a is the last cell in C to be painted. Similarly if $a = \texttt{PAINT\_DOWN}(\cdot)$ then the cell below a must be clear, which contradicts the fact that u is the last cell painted.

□

In the remainder of this section, we show that the conditions of Lemma 4 are also sufficient. Our strategy is the following. We first show that this is true if s_0 is comprised of a $1 \times n$ grid and one robot, then that it is true on $1 \times n$ grids with multiple robots, and finally, we tackle the general case. In each instance, we provide an algorithm and justify its correctness.

Lemma 5. *Let $I = (s_0, \texttt{goal})$ be a PE-FLOORTILE instance with one robot, one column, and at least one hole. Then I is solvable.*

Proof. We claim that Algorithm 1 produces a plan for any such instance I. Let $I = (s_0, \texttt{goal})$ be an instance with one robot, one column, and at least one hole. Let $G = (V, E)$ be the grid of s_0 and $h \in V$ be its bottom-most hole. After calling paint_while_facing_down(1), it is easy to see that the robot paints all the targets below h. After reaching h, the while loop breaks and the function terminates, since h is the bottom-most cell that is not a target. Note that we assume the function paint paints the target cell with the appropriate color. Next, when calling paint_while_facing_up(1), the robot first moves up as far as he is allowed to, and then iteratively paints all of the target cells above h. Once the robot reaches h, the while loop breaks since the cell below h, if it exists, is colored. Thus, the algorithm terminates and we reach a goal state. □

The behavior of the function paint_while_facing_down is such that the robot with index i moves up and paints the cell below him until he reaches a hole. On the other hand, in paint_while_facing_up the ith robot moves down and paints the cell above him if needed, and only stops once he is no longer allowed to move down. The behavior of Algorithm 1 is demonstrated in Figure 3.1.

Lemma 6. *Let $I = (s_0, \texttt{goal})$ be a PE-FLOORTILE instance with H holes, $N \leq H$ robots and one column. Then I is solvable.*

Proof. We justify that Algorithm 2 produces plans for such instances. Note that we reuse the functions paint_while_facing_down and paint_while_facing_up from Algorithm 1.

Algorithm 1 Solve a Floortile instance with one robot one column, and at least one hole.

```
function SOLVE($s_0$, goal)
    PAINT_WHILE_FACING_DOWN(1)
    PAINT_WHILE_FACING_UP(1)
end function

function PAINT_WHILE_FACING_DOWN(idx)
    $r \leftarrow$ ROBOT_WITH_INDEX(idx)
    MOVE_UNTIL_COLLISION($r$, down)
    while COORDINATES_OF($r$) $\in$ targets do
        MOVE($r$, up)
        PAINT($r$, down)
    end while
end function

function PAINT_WHILE_FACING_UP(idx)
    $r \leftarrow$ ROBOT_WITH_INDEX(idx)
    MOVE_UNTIL_COLLISION($r$, up)
    while there is a cell below $r$ and it is clear do
        MOVE($r$, down)
        $u \leftarrow$ the cell above $r$
        if $u \in$ targets then
            PAINT($r$, up)
        end if
    end while
end function
```

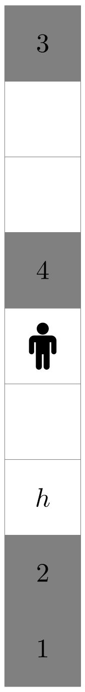


Figure 3.1: An initial state s_0 of an instance with one robot, one column, and at least one hole. Cells colored gray are those that need to be painted with either black or white, the cell marked with h is the bottom-most hole. The numbers indicate the order in which Algorithm 1 paints the targets.

The function `move_to_initial_positions` moves each robot such that the N bottom-most holes are each occupied by a robot. This is possible since by assumption $N \leq H$. Each robot is then relabeled such that the bottom-most robot's index becomes 1, its closest neighbor's index becomes 2, and so on. For each i, $\texttt{robot}_i$ then paints the targets between itself and $\texttt{robot}_{i-1}$. Finally, `paint_while_facing_up`(N) is called, which makes the topmost robot paint the remaining cells. □

An example of an execution of Algorithm 2 is shown in Figure 3.2.

Algorithm 2 Solve a Floortile instance with H holes, $N \leq H$ robots and one column.

```
1: function SOLVE($s_0$, goal)
2:     MOVE_TO_INITIAL_POSITIONS_IN_COLUMN( )
3:     RELABEL_ROBOTS( )
4:     for $i = 1, \ldots, N$ do
5:         PAINT_WHILE_FACING_DOWN($i$)
6:     end for
7:     PAINT_WHILE_FACING_UP($N$)
8: end function
```

Lemma 7. *Let $I = (s_0, \texttt{goal})$ be a PE-FLOORTILE instance with N robots and H holes with at least one hole per column and $H \geq N$. Then I is solvable.*

Proof. We expand our previous two algorithms to handle grids of any size. We show that Algorithm 3 produces plans for instances with H holes, $1 \leq N \leq H$ robots, and at least one hole per column. Let H_i be the number of holes in each column i. The function `distribute_among_columns` moves the robots in the grid such that at each column i there are at most H_i robots, it also moves at least one robot to the first column. This is possible due to our assumptions since $\sum H_i = H$. The algorithm then iterates over each column, starting at the left-most one. On iteration i, the column i is painted with the procedure defined in Lemma 6. After the ith column is painted, if there is no robot in the $i+1$th column, a robot from the ith column migrates by moving one unit to the right. □

Algorithm 3 Solve a Floortile instance with H holes, $N \leq H$ robots, and at least one hole per column.

```
1: function SOLVE($s_0$, goal)
2:     DISTRIBUTE_AMONG_COLUMNS( )
3:     for $i = 1, \ldots,$ num_columns do
4:         PAINT_COLUMN($i$)
5:         if $i <$ num_columns and there are no robots in column $i + 1$ then
6:             MIGRATE($i$)
7:         end if
8:     end for
9: end function
```

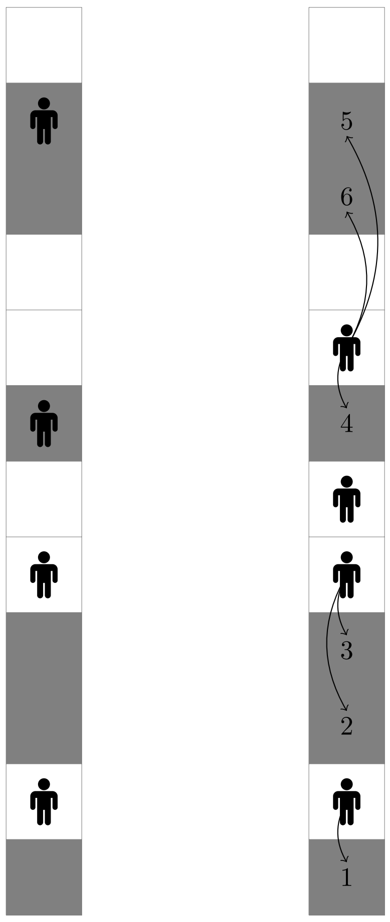


Figure 3.2: Left: An initial state s_0 of an instance with H holes, $N \leq H$ robots, and one column. Cells colored gray are those that need to be painted with either black or white. Right: after calling `move_to_initial_positions()`. The numbers indicate the order in which Algorithm 2 paints the targets. The arrows indicate which robot paints which cell.

Action a	a **is applicable in** s **if**	**Effect of** a	**Cost of** a
$\texttt{MOVE\_UP}(\texttt{robot}_i)$	$\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{UP}) \in V \wedge$ $\texttt{clear}\big(\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{UP})\big)$	$\texttt{robots\_pos}(\texttt{robot}_i) \leftarrow \texttt{robots\_pos}_s(\texttt{robot}_i) + \mathbf{UP}$	3
$\texttt{MOVE\_DOWN}(\texttt{robot}_i)$	$\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{DOWN}) \in V \wedge$ $\texttt{clear}\big(\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{DOWN})\big)$	$\texttt{robots\_pos}(\texttt{robot}_i) \leftarrow \texttt{robots\_pos}_s(\texttt{robot}_i) + \mathbf{DOWN}$	1
$\texttt{MOVE\_RIGHT}(\texttt{robot}_i)$	$\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{RIGHT}) \in V \wedge$ $\texttt{clear}\big(\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{RIGHT})\big)$	$\texttt{robots\_pos}_s(\texttt{robot}_i) \leftarrow \texttt{robots\_pos}(\texttt{robot}_i) + \mathbf{RIGHT}$	1
$\texttt{MOVE\_LEFT}(\texttt{robot}_i)$	$\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{LEFT}) \in V \wedge$ $\texttt{clear}\big(\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{LEFT})\big)$	$\texttt{robots\_pos}(\texttt{robot}_i) \leftarrow \texttt{robots\_pos}_s(\texttt{robot}_i) + \mathbf{LEFT}$	1

Table 3.1: Floortile actions, preconditions, and costs. Here **UP, DOWN, LEFT, RIGHT** are constants that stand for $(0,1), (0,-1), (-1,0), (1,0)$.

$\texttt{PAINT\_UP}(\texttt{robot}_i, c)$	$\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{UP}) \in V \land$ $\texttt{clear}\big(\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{UP})\big) \land$ $\texttt{avail\_color}_s(\texttt{robot}_i) = c$	$\texttt{painted}(\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{UP}) \leftarrow c$	2
$\texttt{PAINT\_DOWN}(\texttt{robot}_i, c)$	$\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{DOWN}) \in V \land$ $\texttt{clear}\big(\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{DOWN})\big) \land$ $\texttt{avail\_color}_s(\texttt{robot}_i) = c$	$\texttt{painted}(\texttt{robots\_pos}_s(\texttt{robot}_i + \mathbf{DOWN}) \leftarrow c$	2
$\texttt{CHANGE\_TO\_WHITE}(\texttt{robot}_i)$	$\texttt{avail\_color}_s(\texttt{robot}_i) = \texttt{black}$	$\texttt{avail\_color}(\texttt{robot}_i) \leftarrow \texttt{white}$	2
$\texttt{CHANGE\_TO\_BLACK}(\texttt{robot}_i)$	$\texttt{avail\_color}_s(\texttt{robot}_i) = \texttt{white}$	$\texttt{avail\_color}(\texttt{robot}_i) \leftarrow \texttt{black}$	2

Table 3.2: Continuation of Table 3.1.

4
TERMES

The TERMES domain models a simplified version of the termite-inspired Harvard TERMES robots [19]. In the general case, this is a multi-agent domain, however, the IPC variant that we work on in this thesis considers a single agent. The goal is to arrange blocks in a rectangular graph in a desired configuration. It is possible to stack more than one block on a single vertex. There is an agent that can pick up, place, create or destroy blocks. Blocks are created and destroyed at the depot. The agent can only move from a vertex u to a neighboring vertex v if $|\texttt{num-blocks}(u) - \texttt{num-blocks}(v)| \leq 1$. There are similar restrictions on the pickup and place actions.
We start by formalizing TERMES states and actions.

Definition 16 (TERMES state)**.** *A TERMES state* $s = (G, \texttt{max-height}, \texttt{robot-pos}_s,$ $\texttt{num-blocks}_s, \texttt{depot}, \texttt{has-block}_s)$ *is comprised of*

- *A rectangular graph* $G = (V, E)$
- *The maximal height* $\texttt{max-height} \in \mathbb{N}$
- *The robot position* $\texttt{robot-pos}_s \in V$
- *The height function* $\texttt{num-blocks}_s : V \to \{0, \ldots, \texttt{max-height}\}$
- *The depot position* $\texttt{depot} \in \{u \in V : \texttt{num-blocks}_s(u) = 0\}$
- *The has-block variable* $\texttt{has-block}_s \in \{\mathbf{T}, \mathbf{F}\}$

Definition 17 (empty TERMES state)**.** *A TERMES state s is empty if* $\texttt{num-blocks}_s(u) = 0 \; \forall u \in V$.

A graphical representation of two TERMES states is shown in Figure 4. A table listing the TERMES actions and preconditions and effects is shown in 4.1.
For the decision problem of TERMES, we consider two variants which we call TERMES−ES, which limits initial states to be empty, and one with no restrictions on `num-blocks`, which we call TERMES−NES. We do not prove any results for the decision problems of TERMES−ES, however, we conjecture that its bounded plan existence problem is **NP**-hard. This is discussed further in Section 8.1.1.

Action a	a **is applicable in** s **if**	**Effect of** a
$\texttt{MOVE}(u)$	$(\texttt{robot-pos}_s, u) \in E \wedge$ $\lvert\texttt{num-blocks}_s(\texttt{robot-pos}_s) - \texttt{num-blocks}_s(u)\rvert \leq 1$	$\texttt{robot-pos} \leftarrow u$
$\texttt{PLACE}(u)$	$(\texttt{robot-pos}_s, u) \in E \wedge$ $\texttt{has-block}_s = \mathbf{T} \wedge$ $\texttt{num-blocks}_s(u) = \texttt{num-blocks}_s(\texttt{robot-pos}_s) \wedge$ $u \neq \texttt{depot} \wedge$ $\texttt{num-blocks}_s(u) < \texttt{max-height}$	$\texttt{num-blocks}(u) \leftarrow \texttt{num-blocks}_s(u) + 1 \wedge$ $\texttt{has-block} \leftarrow \mathbf{F}$
$\texttt{PICKUP}(u)$	$(\texttt{robot-pos}_s, u) \in E \wedge$ $\texttt{has-block}_s = \mathbf{F} \wedge$ $\texttt{num-blocks}_s(u) + 1 = \texttt{num-blocks}_s(\texttt{robot-pos}_s)$	$\texttt{num-blocks}(u) \leftarrow \texttt{num-blocks}_s(u) - 1 \wedge$ $\texttt{has-block} \leftarrow \mathbf{T}$
$\texttt{DESTROY}$	$\texttt{robot-pos}_s = \texttt{depot} \wedge$ $\texttt{has-block}_s = \mathbf{T}$	$\texttt{has-block} \leftarrow \mathbf{F}$
$\texttt{PICKUP\_FROM\_DEPOT}$	$\texttt{robot-pos}_s = \texttt{depot} \wedge$ $\texttt{has-block}_s = \mathbf{F}$	$\texttt{has-block} \leftarrow \mathbf{T}$

Table 4.1: TERMES actions and preconditions for each $u \in V$.

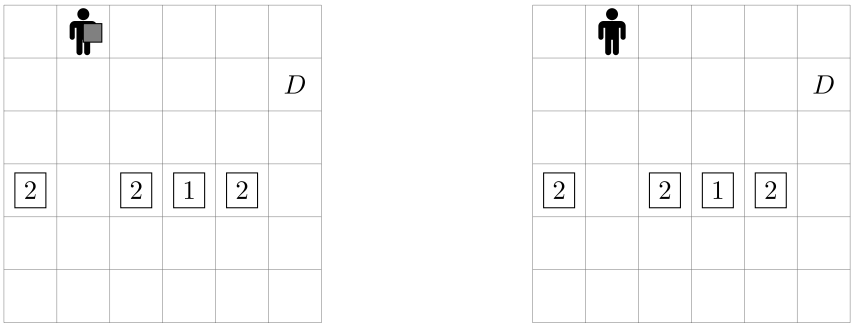


Figure 4.1: A graphical representation of two TERMES states. In the figure on the left, we have `has-block` $= \mathbf{T}$, and on the right `has-block` $= \mathbf{F}$. The numbers on the vertices indicate the values of `num-blocks`. Vertices u without numbers are those for which $\texttt{num-blocks}_s(u) = 0$. The cell marked with D is the depot.

4.1 PE-TERMES-NES & BPE-TERMES-NES are **NP**-hard

The decision problem formulation for the non-empty start plan existence and bounded plan existence problems are given below.

PE-TERMES-NES
INPUT: An initial state s_0 such that and a goal configuration $\texttt{goal} : V \to \mathbb{N}$.
QUESTION: Is there a sequence of actions π applicable on s_0 such that $\texttt{num-blocks}_{s_0[\pi]} = \texttt{goal}$?

BPE-TERMES-NES
INPUT: An initial state s_0 such that, a goal configuration $\texttt{goal} : V \to \mathbb{N}$ and an integer $K > 0$.
QUESTION: Is there a sequence of actions π applicable on s_0 such that $\texttt{num-blocks}_{s_0[\pi]} = \texttt{goal}$ and $\|\pi\| \leq K$?

To show the **NP**-hardness of both problems we use two similar reductions from DEG−4−GHP, the Hamiltonian path problem for grid graphs, which is **NP**-complete due to Theorem 1. The construction for s_0 and `goal` are depicted graphically in Figure 4.2. The goal in both reductions is to build a stack of height $|V| - 1$ which has to be constructed through a ramp, forming a simple path of length $|V|$. Here V is the vertex set of the grid graph whose Hamiltonian cycle we seek. This path is constrained to vertices of V only since s_0 contains tall stacks, which the agent cannot interact with.
Before proving hardness, we first need some intermediary results. It makes sense to first consider and prove results about TERMES instances defined over arbitrary simple graphs instead of rectangular grids. The invariant of Lemma 8 was hinted to us by Helmert [1].

Definition 18 (generalized TERMES state)**.** *A generalized TERMES state is a TERMES state whose grid can be any simple graph.*

Lemma 8. *Let s, s' be generalized TERMES states such that s' is reachable from s. Let (V, E) be the simple graph of s and s'. Then for each*

$$\texttt{num-blocks}_s(\texttt{robot-pos}_s) \leq n \leq \texttt{num-blocks}_{s'}(\texttt{robot-pos}_{s'})$$

there exists a vertex $u \in V$ with $\texttt{num-blocks}_{s'}(u) = n$.

Proof. We have $s' = s[\pi]$ for some finite sequence of actions $\pi = \langle a_1, \dots, a_N \rangle$ applicable in s. Our proof is by induction on N.
Base case ($N = 0$):
We have that π is the empty sequence, so $s = s'$, and the claim holds with $u = \texttt{robot-pos}_s$.
Inductive step ($N \rightsquigarrow N + 1$):
For $k \leq N+1$ let $\pi_k = \langle a_1, \dots, a_k \rangle$. If $k = 0$ then π_k is the empty sequence. Further, define state at step k as $s_k := s_{\pi_k} := (G, \texttt{max-height}, \texttt{robot-pos}_k, \texttt{num-blocks}_k, \texttt{depot}, \texttt{has-block}_k)$, and the agent's height at step k as $h_k = \texttt{num-blocks}_k(\texttt{robot-pos}_k)$. Finally define the set $S_k := \{n : h_0 \leq n \leq h_k\}$.
We may assume the claim holds for s_N. We now do a case analysis on a_{N+1}.

- If $a_{N+1} \in \{\texttt{DESTROY}, \texttt{PICKUP\_FROM\_DEPOT}\}$ then $\texttt{num-blocks}_N = \texttt{num-blocks}_{N+1}$ and $\texttt{robot-pos}_N = \texttt{robot-pos}_{N+1}$ so the claim holds by the IH (inductive hypothesis).
- If $a_{N+1} = \texttt{MOVE}(v)$ for some $v \in V$ then $\texttt{robot-pos}_N = \texttt{robot-pos}_{N+1}$, hence we only need to find a vertex u for each $n \in S_{N+1} \setminus S_N$ such that $\texttt{robot-pos}_N(u) = n$, all other options for n are covered by the IH. Note that for each k we have $|h_k - h_{k+1}| \leq 1$, so the following distinction covers all possibilities:
Case 1: $h_{N+1} = h_N$. Then $S_{N+1} \setminus S_N = \emptyset$ so there is nothing to do.
Case 2: $h_{N+1} = h_N \pm 1$. Then either $S_{N+1} \setminus S_N = \emptyset$, or $S_{N+1} \setminus S_N = \{h_{N+1}\}$. In the latter, we can set $u = \texttt{robot-pos}_{N+1}$.
- Now we consider the case $a_{N+1} = \texttt{PLACE}(v)$ for some $v \in V$. We observe that v and $\texttt{robot-pos}_N$ must be neighbors since otherwise, this action would not be applicable. In particular $v \neq \texttt{robot-pos}_N$, since G does not contain any self-loops. Then $h_{N+1} = h_N$ so $S_{N+1} = S_N$. Take an $n \in S_{N+1}$. By the IH there exists a u such that $\texttt{num-blocks}_N(u) = n$. If $u \neq v$ then $\texttt{num-blocks}_{N+1}(u) = n$ since $\{u' : \texttt{num-blocks}_N(u') \neq \texttt{num-blocks}_{N+1}(u')\} = \{v\}$. Now we consider the case $v = u$. We notice that $\texttt{num-blocks}_N(v) = h_N = n$, otherwise $\texttt{PLACE}(v)$ would not be applicable on s_N. Then since $h_N = h_{N+1}$ we have $\texttt{num-blocks}_{N+1}(\texttt{robot-pos}_{N+1}) = n$.
- Finally we consider the case $a_{N+1} = \texttt{PICKUP}(u)$. Like in the previous case, we observe that $\texttt{robot-pos}_N = \texttt{robot-pos}_{N+1}$ and $h_N = h_{N+1}$. Hence $S_{N+1} = S_N$. Take an $n \in S_{N+1}$. By the IH there exists a u such that $\texttt{robot-pos}_N(u) = n$. Using the same reasoning as in the previous case analysis we have that the claim holds if $u \neq v$. On the other hand, the case $u = v$ is impossible,since then $n = h_{N+1} + 1$ and $n \leq h_{N+1}$.

□

In particular, Lemma 8 shows that if the agent is on the ground and reaches a state where he is at height h, then there exists at least one block at height $0, 1, \dots, h$. We now use invariant to find a bound on the maximal stack height, thus giving necessary conditions of the reachability of certain states.

Corollary 1. *Let s be a generalized TERMES state with $\texttt{num-blocks}_s(\texttt{robot-pos}_s) = 0$ and $\texttt{num-blocks}_s(u) \geq |V|$ for some $u \in V$. Let s' be a state reachable from s. Then none of the following are applicable in s' : $\texttt{PICKUP}(u), \texttt{PLACE}(u)$, $\texttt{MOVE}(u)$.*

Proof. Assume by contradiction that one of these actions is applicable in s'. Let (V, E) be the simple graph of s and s'. We have $\texttt{num-blocks}_{s'}(\texttt{robot-pos}_{s'}) \geq |V|-1$. Then by Lemma 8 for each $n \in \{0, \ldots, |V|-1\}$ there exists a vertex $v \in V \setminus \{u\}$ with $\texttt{num-blocks}_{s'}(v) = n$. In total, there are $|V|$ choices for n that must be covered with $|V|-1$ choices for v. This is impossible due to the pigeonhole principle. □

Finally, we give necessary and sufficient conditions on the graph's topology for certain states to be reachable.

Lemma 9. *Let s_0 be an empty generalized TERMES state and s be a state reachable from s_0. Then, if s is such that there exists exactly one of each $u_1, \ldots, u_N$ satisfying $\texttt{num-blocks}_s(u_1) = 1, \ldots, \texttt{num-blocks}_s(u_N) = N$, and it holds that $\texttt{robot-pos}_s = u_N$, we have that there exists a u_0 with $\texttt{num-blocks}_s(u_0) = 0$ and $\langle u_0, \ldots, u_N \rangle$ is a simple path in the graph of s_0.*

Proof. Fix an empty generalized TERMES s_0. Let its graph be $G = (V, E)$. We show the following by induction on N.

$$\forall N \in \mathbb{N}_{>0}$$

$$\forall s \text{ reachable from } s_0 \text{ s.t. } \exists!\ u_1, \ldots, u_N \in V : \texttt{num-blocks}_s(u_i) = i, 1 \leq i \leq N$$

$$\text{and } \texttt{robot-pos}_s = u_n$$

$$\exists\ u_0 \in V :$$

$$\texttt{num-blocks}_s(u_0) = 0 \text{ and } \langle u_0, u_1, \ldots, u_N \rangle \text{ is a simple path in } G$$

- Base case: $N = 1$
  Let u_0 be the cell on which the robot is on when he places the block on u_1. Then clearly $\langle u_0, u_1 \rangle$ is a simple path, and $\texttt{num-blocks}_s(u_0) = 0, \texttt{num-blocks}_s(u_1) = 1$.

- Inductive step: $N \rightsquigarrow N + 1$
  Let s be a state reachable from s_0 with exactly one $u_1, \ldots, u_{N+1}$ with $\texttt{num-blocks}_s(u_i) = i, 1 \leq i \leq N + 1$. We now combine the facts that there exists only one u with $\texttt{num-blocks}_s(u) = N + 1$, and that s is reachable from s_0 and get that there must be a cell v that is a neighbor of u with $\texttt{num-blocks}_s(v) = N$. If this was not the case then there would not be a single action applicable on s, which is a contradiction since s is reachable from s_0 iff s_0 is reachable from s. Now define the state s' that results when applying the action $\texttt{MOVE}(v)$ in s. By the IH we have that $\langle u_0, u_1, \ldots, u_N \rangle$ is a simple path in G for some u_0 with $\texttt{num-blocks}_s(u_0) = 0$ this path can be extended to $\langle u_0, \ldots, u_N, u_{N+1} \rangle$, which finishes the proof.

□

Lemma 10. *Let s be a state whose rectangular graph is $G = (V, E)$ and $V' \subset V$ s.t. $\texttt{depot} \in V'$. Assume that*

$$\texttt{num-blocks}_s(u) = \begin{cases} 0 & u \in V' \\ M & u \notin V' \end{cases}$$

for some $M \geq |V|$. Let $G' = (V', E')$ be the graph induced by V'. Then for each $u \in V'$ s.t. $u \neq$ `depot` *it holds that*

$$G' \text{ has a simple path of length at least } N \text{ starting in } \texttt{depot} \text{ and ending in } u$$

$$\Longrightarrow$$

$$\text{There exists a state } s' \text{ reachable from } s \text{ such that } \texttt{num-blocks}_s(u) = N - 1$$

Furthermore, if G has a simple path of length at least N, it takes at most $N^2|V'| + N^2 - N|V'| - N + |V'|$ steps to reach s' from s and $2N^2|V'| + 2N^2 - 4N|V'| - 4N + 2|V'| + 3$ steps to reach a state s'' where

$$\texttt{num-blocks}_{s''}(v) = \begin{cases} 0 & v \in V' \setminus \{u\} \\ N-1 & v = u \\ M & v \notin V' \end{cases}$$

Proof. Let $P = \langle p_1, \ldots, p_M \rangle$ be a simple path in G consisting of only of vertices from V' of length $M \geq N$ where $p_1 = \texttt{depot}, p_M = u$ and truncate it from the front to create the subpath $P' = \langle p'_1, \ldots, p'_N \rangle$ of length N where $p'_N = u$. Let s_{ramp} be the state where

$$\texttt{num-blocks}_{s_{\text{ramp}}}(v) = \begin{cases} 0 & v = p'_1 \\ 1 & v = p'_2 \\ \vdots & \\ N-1 & v = p'_N \\ M & v \notin V' \\ 0 & \text{otherwise} \end{cases}$$

and $\texttt{robot-pos}_{s_{\text{ramp}}} = p'_{N-1}$. We show that s_{ramp} is reachable from s and that s'' is reachable from s_{ramp} while giving a bound on the maximum number of steps needed to reach each of these two states.

- Part 1: s_{ramp} is reachable from s: The agent can build the ramp in $N-1$ rounds. After the i'th round, each vertex in $\{p'_i, \ldots p'_N\}$ contains one block more than in the previous round.
- Part 2: s'' is reachable from s_{ramp}: The agent will now destroy the part of the ramp that leads to u. This task can also be done in rounds. After round i each vertex in $\{p'_i, \ldots, p'_{N-1}\}$ contains one block fewer than in the previous round.

In both cases, it is easy to see that each round is reachable from the previous one. We let $d^*_{G'}$ be the shortest path distance in G'.
The total number of actions needed to reach s_{ramp} from s is

$$A = \underbrace{d^*_{G'}(\texttt{robot-pos}_s, \texttt{depot})}_{\text{go to depot}} + \sum_{j=1}^{N-1} \sum_{i=j}^{N-1} \Big[\underbrace{1}_{\text{pickup from } \texttt{depot}} + \underbrace{d^*_{G'}(\texttt{depot}, p'_i)}_{\text{go to } p'_i} + \underbrace{1}_{\text{place block in } p'_{i+1}} +$$

$$\underbrace{d^*_{G'}(p'_i, \texttt{depot})}_{\text{go back to } \texttt{depot}}\Bigg]$$

The total number of actions needed to reach s'' from s_{ramp} is

$$B = \underbrace{1}_{\text{go to } p'_{N-2}} + \sum_{j=1}^{N-3} \sum_{i=j}^{N-2} \Bigg[\underbrace{1}_{\text{pickup from } p'_{N-i}} + \underbrace{d^*_{G'}(p'_{N-1-i}, \texttt{depot})}_{\text{go to } \texttt{depot}} +$$

$$\underbrace{1}_{\text{destroy block}} + \underbrace{d^*_{G'}(\texttt{depot}, p'_{N-2-i})}_{\text{go to } p'_{N-2-i}}\Bigg] + \underbrace{1 + d_{\ell_1}(p'_1, \texttt{depot}) + 1}_{\text{last round}}$$

The bound on the number of steps for reaching $s' = s_{\text{ramp}}$ follows from simplifying A while noticing that $d^*_{G'}(\texttt{depot}, \cdot) \leq |V'|$. Similarly, the bound on the steps needed to reach s'' follows from adding A and B. In the end, we get

$$A \leq N^2|V'| + N^2 - N|V'| - N + |V'|$$

$$A + B \leq 2N^2|V'| + 2N^2 - 4N|V'| - 4N + 2|V'| + 3$$

□

Theorem 5. *PE-TERMES-NES is **NP**-hard.*

Proof. We show the reduction

$$\texttt{DEG-4-GHP} \leq \texttt{PE-TERMES-NES}$$

We recall that DEG-4-GHP is **NP**-complete due to Theorem 1. Let $G = (V, E)$ be a grid graph and $s \neq t \in V$. We construct a TERMES state s_0 and target `goal` as follows:

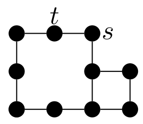
- $G_{s_0} = R(\texttt{width}(G), \texttt{height}(G))$. Let the V_{s_0}, E_{s_0} be the vertex and edge set of this graph.
- $\texttt{max-height} = |V_{s_0}|$
- $\texttt{depot} = s$
- $\texttt{robot-pos}_{s_0} = s$
- $\texttt{num-blocks}_{s_0}(u) = \begin{cases} \texttt{max-height} & u \in V_{s_0}, u \notin V \\ 0 & u \in V_{s_0}, u \in V \end{cases}$
- $\texttt{has-block}_{s_0} = \mathbf{F}$
- $\texttt{goal}(u) = \begin{cases} \texttt{max-height} & u \in V_{s_0} \setminus V \\ |V| - 1 & u = t \\ 0 & \text{otherwise} \end{cases}$

It is clear that this construction can be done in polynomial time. A graphical representation of s_0 and $\texttt{goal}$ is shown in Figure 4.2. By Lemma 10 we know that if G has an s-t Hamiltonian path, then a state s' with $\texttt{num-blocks}_{s'} = \texttt{goal}$ can be reached. Assume now that $(s_0, \texttt{goal})$ is solvable. By Corollary 1, we may restrict G_{s_0} to G, since the robot cannot interact with the vertices of $V_{s_0} \setminus V$. Next, since the instance can be solved, we can reach a state s' where $\texttt{robot-pos}_{s'} = t$ and $\texttt{num-blocks}_{s'}(t) = |V| - 1$. Then by Lemma 8, we get that there are $u_0, u_1, \ldots, u_{|V|-1}$ with $\texttt{num-blocks}_{s'}(u_i) = i, 0 \leq i \leq |V| - 1$, and by Lemma 9 we have that $u_0, \ldots u_{|V|-1}$ is a simple path. This simple path is an s-t Hamiltonian path since it has length $|V|$ and $u_0 = \texttt{depot}, u_{|V|-1} = t$. □

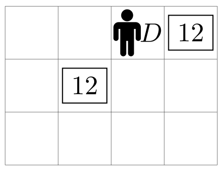
Theorem 6. *BPE−TERMES−NES is **NP**-hard.*

Proof. We show the reduction GHC $\leq$ BPE−TERMES−NES and use the same construction for s_0 and $\texttt{goal}$ as in Theorem 5. We set $K = 2|V|^3 - 2|V|^2 - 2|V| + 3$, where $X = \max_{u \in V} d_{\ell_1}(\texttt{depot}, u)$, and V is the vertex set of the grid of state s_0. The Theorem follows from the bound in Lemma 10 using the same reasoning as in Theorem 5. where

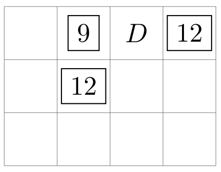
□



(a) A DEG−4−GHP instance G.



(b) The initial TERMES state.



(c) The goal $\texttt{goal}$.

Figure 4.2: Example reduction from DEG−4−GHP to PE−TERMES−NES

4.2 A special case of BPE−TERMES−NES is **NP**-hard

We show that BPE−TERMES−NES remains **NP**-hard even if all initial states have at most 1 block per vertex.

Definition 19 (Free neighbor direction, shifting function)**.** *Let $G = (V, E)$ be a grid graph such that* $\deg(u) \leq 3 \; \forall u \in V$. *We define*

$$\texttt{free\_neighbor\_direction}_G : V \to \{(0,1), (0,-1), (1,0), (-1,0)\}$$

which maps each $u \in V$ to a $z \in \{(0,1), (0,-1), (1,0), (-1,0)\}$ such that $(u, u + z) \notin E$. Next, we define the shifting function $\texttt{shift}_G : V \times \mathbb{N} \to \mathbb{Z}^2$ as follows:

$$\texttt{shift}_G(u, \texttt{amont}) = 3 \cdot u + \texttt{amount} \cdot \texttt{free\_neighbor\_direction}_G(u)$$

Lemma 11. *BPE−TERMES−NES is **NP**-hard even on grids where the initial state has at most 1 block per vertex.*

Proof. We reduce from DEG−3−GHP which is **NP**-complete according to Theorem 2. Let $I = (G, s, t)$ be an instance of DEG−3−GHP with $G = (V, E)$. We define the initial state s_0 and bound K as follows.

- The rectangular graph of s_0 is $G' = R(m, n) = (V', E')$ where $m = 3 \cdot \texttt{width}(G) + 2$ and $n = 3 \cdot \texttt{height}(G) + 2$.
- $\texttt{max-height}s_0 = 2$
- $\texttt{depot} = 3 \cdot s$
- $\texttt{robot-pos}_{s_0} = 3 \cdot s$
- $\texttt{has-block}_{s_0} = \mathbf{F}$

We define $\texttt{num-blocks}_{s_0}$ and $\texttt{goal}$ as follows. Let $S_i = \{\texttt{shift}_G(u, i) : u \in V, u \neq t, u \neq s\}$, Then

$$\texttt{num-blocks}_{s_0}(u') = \begin{cases} 1 & u' \in S_0 \cup S_1 \\ 0 & \text{otherwise} \end{cases}$$

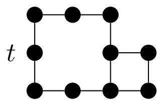
and

$$\texttt{goal}(u') = \begin{cases} 2 & u' \in S_1 \\ 1 & u' = \texttt{shift}_G(t, 1) \\ 0 & \text{otherwise} \end{cases}$$

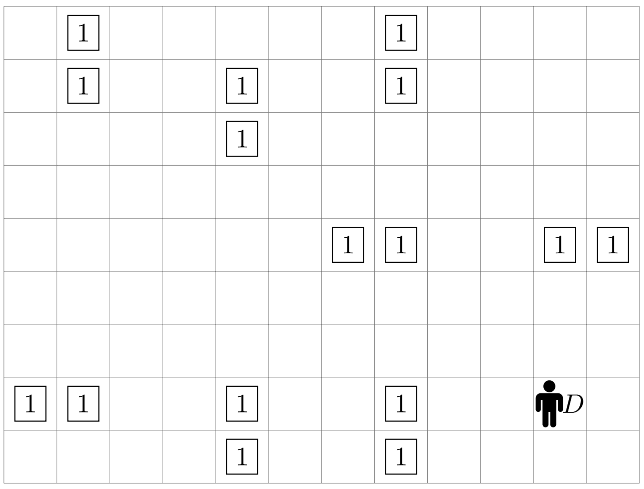
The bound is $K = 5 \cdot |V| - 5$. An example of the construction is shown in Figure 4.3. It is easy to see that $(s_0, \texttt{goal}, K)$ can be defined in polynomial time w.r.t. $\|I\|$.
We assume first that I is solvable. Let $P = \langle p_1, \ldots, p_{|V|} \rangle$ be an s-t Hamiltonian path in G. We construct a plan π for I' of length K as follows. The robot first grabs a block from the depot. Then, for each $i \in 1, \ldots, |V| - 2$, the robot moves to $3 \cdot p_{i+1}$, places a block in $\texttt{shift}(p_{i+1}, 1)$, moves 1 unit in the direction closest to $3 \cdot p_{i+2}$, and picks up the block from $3 \cdot p_i$. Finally, the robot moves to $3 \cdot t$ and places the final block. It is easy to see that π is a plan. Further, we have that $\|\pi\|$ is the sum of the total number of movement, pickup, and placement actions. From our argument above we can see that

$$\begin{aligned} \text{number of movement actions } &= 3 \cdot |V| - 1 \\ \text{number of pickup actions } &= |V| - 2 \\ \text{number of placement actions } &= |V| - 1 \end{aligned}$$

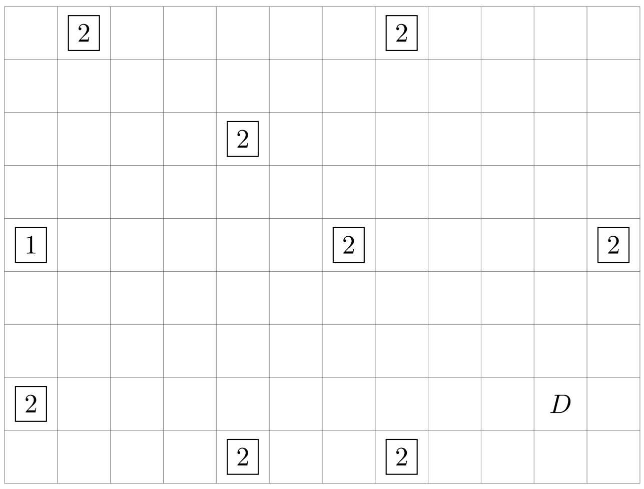
Adding the three shows that $\|\pi\| = K$, so we are done.
Next, we assume that $(s_0, \texttt{goal})$ is solvable in at most K steps. Let π be a plan of length at most $\|K\|$. We will first show that this bound is sharp, i.e. $\|\pi\|$ cannot be less than K. Since there are $|V| - 2$ blocks in places that do not match their goal, and since in any plan the robot must at some point pick up a block from the depot, the number of pickup actions in $|V|$ must be at least $|V| - 1$. Also, it is necessary to put an additional block on each $\texttt{shift}(u, 1), u \in V \setminus \{s\}$, to reach the goal, so a minimum of $|V| - 1$ placement actions is needed. Finally, by the spacing of the grid, the robot needs to move at least $3 \cdot (|V| - 1)$ squares. Adding these three values yields exactly K. In particular, since we need $|V| - 1$ movement actions, the robot's movement must match a Hamiltonian path in G, since each time he places a block, he must do so by moving 3 squares, and thus not visit any cell with



(a) The instance (G, s, t) of `DEG-3-GHP`.



(b) The `BPE-TERMES-NES` initial state s_0 with $K = 5 \cdot |V| - 5 = 45$.



(c) The TERMES goal `goal`.

Figure 4.3: An example reduction from `DEG-3-GHP` to `BPE-TERMES-NES`. In this case, $(s_0, \texttt{goal})$ cannot be solved in $\leq K$ steps since G does not have an s-t Hamiltonian path.

an already placed stack of 2 blocks. Next, this path must end in t, since otherwise the number of pickup actions would be more than $|V| - 1$.

□

5 Tidybot

The Tidybot domain models a multi-agent logistics transportation problem on rectangular graphs. It is comprised of robots, grippers obstacles, base obstacles, carts, and objects. The goal is for the robots to move some transportation objects from their initial positions to one of many goal positions. Robots have grippers, which allow them to pick up and drop objects at a distance up to some fixed radius. Base obstacles and gripper obstacles limit the mobility of the robots and their grippers respectively. Robots can hold at most one object at a time. Carts are movable objects that can be used to store an unlimited amount of objects. They can be pushed but not pulled.

The domain PDDL file of Tidybot allows for multiple items to be located on the same cell. We believe this is a modeling oversight, hence we enforce that at most one item is located in each cell. Similarly, the PDDL file also allows grippers to navigate freely around their corresponding robots up to some fixed radius. An example of this is shown in Figure 5. We believe this models unrealistic scenarios, hence we do not allow for this. We do so by adding a constraint that does not allow the gripper to make turns, making grippers only able to move in straight lines.

We add some additional changes, in the PDDL domain file, a robot must be parked in order to pick up or place an object, and each gripper movement takes one action. We simplify this, each robot can pick up or drop an object using a single action instead of having to park, move the gripper on top of the object, move the gripper back to the robot's base, and un-park.

Definition 20 (Tidybot state)**.** *A Tidybot state s is a* 10*-tuple comprised of*

- *A rectangular grid* $G_s = R(m, n)$ *for some* $m, n \in \mathbb{N}$
- *The gripper radius* $\texttt{gripper\_rad}_s \in \{1, \dots, \max\{m, n\}\}$
- *The set of robots* $\texttt{Robots}_s = \{\texttt{robot}_1, \dots, \texttt{robot}_M\}$ *for some* $M \leq |V|$
- *The set of carts* $\texttt{Carts}_s = \{\texttt{cart}_1, \dots, \texttt{cart}_N\}$ *for some* $N \leq |V|$.
- *The set of objects* $\texttt{Objects}_s = \{\texttt{object}_1, \dots, \texttt{object}_O\}$ *for some* $O \leq |V|$
- *The position of each robot* $\texttt{robots\_pos}_s : \texttt{Robots}_s \to V$

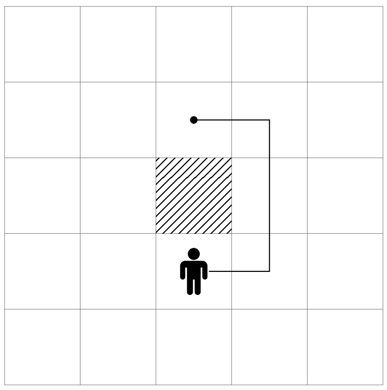


Figure 5.1: A legal state in Tidybot as described in its PDDL domain file. The robot's gripper can reach the cell that is two units north of the robot, despite there being a gripper obstacle in the way.

- *The position of each cart:* $\texttt{carts\_pos}_s : \texttt{Carts}_s \to V$
- *The position of each object* $\texttt{objects\_pos}_s : \texttt{Objects}_s \to V \cup \texttt{Carts}_s \cup \texttt{Robots}_s$
- *The set of base obstacles* $\texttt{base\_obstacles}_s \subset V$
- *The set of gripper obstacles* $\texttt{gripper\_obstacles}_s \subset V$

Definition 21 (holding an object)**.** *Let s be a Tidybot state. We define*

$$\texttt{holding}_s : \texttt{Robots}_s \to \texttt{Objects} \cup \{\texttt{none}\}$$

$$\texttt{holding}_s(\texttt{robot}_i) = \begin{cases} \texttt{object}_j & \textit{if } \texttt{objects\_pos}_s(\texttt{object}_j) = \texttt{robot}_i \\ \texttt{none} & \textit{if } \texttt{objects\_pos}_s^{-1}\Big(\{\texttt{robot}_i\}\Big) = \emptyset \end{cases}$$

Definition 22 (empty cell)**.** *Let s be a Tidybot state whose grid is $G = (V, E)$. We say that a cell $u \in V$ is empty, written $\texttt{empty}_s(u)$, if there is no robot, object, or cart occupying that cell, i.e. if*

$$\texttt{empty}_s(u) \iff \nexists\, \texttt{robot}_i, \texttt{object}_j, \texttt{cart}_k :$$

$$\texttt{robots\_pos}_s(\texttt{robot}_i) = u \lor \texttt{objects\_pos}_s(\texttt{cart}_j) = u \lor \texttt{carts\_pos}_s(\texttt{cart}_k) = u$$

The table listing actions and action preconditions for Tidybot are given in Tables 5.2 and 5.2. The decision problems for Tidybot are

PE-TIDYBOT
INPUT: A Tidybot state s_0, a set of objects $S \subset \texttt{Objects}_{s_0}$ and a goal $\texttt{goal} : S \to 2^V$
QUESTION: Is there a sequence of actions π applicable on s_0 such that $\forall\ \texttt{object}_i \in S$: $\texttt{objects\_pos}_{s_0[\pi]}(\texttt{object}_i) \in \texttt{goal}(\texttt{object}_i)$?

BPE-TIDYBOT
INPUT: A Tidybot state s_0, a set of objects $S \subset \texttt{Objects}_{s_0}$, a goal $\texttt{goal}s : S \to 2^V$ and an integer $K > 0$
QUESTION: Is there a sequence of actions π applicable on s_0 such that $\forall\ \texttt{object}_i \in S$: $\texttt{objects\_pos}_{s_0[\pi]}(\texttt{object}_i) \in \texttt{goal}(\texttt{object}_i)$ and $\|\pi\| \leq K$?

Although the state and goal definitions are very general. In the PDDL instance files, there is exactly one robot, the radius is one, and there is exactly one cart. Further, the initial positions s_0 are such that the robot's gripper and cart are empty. We call the domain that is comprised of these sub-instances `IPC-TIDYBOT`.

5.1 The computational complexity of `IPC-TIDYBOT` and related problems

In this section, we show that `BPE-IPC-TIDYBOT` is **NP**-hard. Also, this domain becomes **NP**-complete if we do not allow for obstacles. This stems from the fact that the plan existence problem `IPC-TIDYBOT` can be solved in polynomial time if there are no obstacles. The following reduction follows a similar idea to that used in Lemma 11.

Lemma 12. *`BPE-IPC-TIDYBOT` is **NP**-hard.*

Proof. We reduce from `DEG-3-GHP`, which is **NP**-complete according to Theorem 2. This decision problem takes as input a grid graph $G = (V, E)$ where $\deg(u) \leq 3\ \forall u \in V$ and $s \neq t \in V$. It asks whether there exists an s-t Hamiltonian path in G. Let (G, s, t) be an instance from `DEG-3-GHP` and V, E be the vertex and edge set of G. We now show that there is a `BPE-IPC-TIDYBOT` instance $(s_0, \texttt{goal}, K)$ that can be constructed in polynomial time and is solvable iff there is an s-t Hamiltonian path in G.
Let $f : V \to \mathbb{Z}^2$ with $f(u) = 4u - (2, 2)$.
The corresponding Tidybot state s_0 is defined as follows:

- The grid $G_{s_0} = R(m, n)$ where $m = 4 \cdot \texttt{width}(G) + 2, n = 4 \cdot \texttt{height}(G) + 2$
- The robot is placed in $f(s)$.
- For each vertex u in $V \setminus \{s\}$ create an object $\texttt{object}_u$ placed in $f(u)$.
- The cart is placed at the top left of the grid.
- There are no obstacles.

We make use of the function `free_neighbor_direction`, which stems from Definition 19. The goal is `goal` is defined as follows:

$$\texttt{goal}(\texttt{object}_u) = \begin{cases} f(u) + 2 \cdot \texttt{free\_neighbor\_direction}_G(u) & u = t \\ f(u) + \texttt{free\_neighbor\_direction}_G(u) & u \neq t \end{cases}$$

Further, let $K = 5|V| - 4$. It is clear that $(s_0, \texttt{goal}, K)$ is a polynomial time construction. A graphical example is shown in Figure 5.2.
We now argue that $(s_0, \texttt{goal}, K)$ is solvable if (G, s, t) is solvable.
Assume (G, s, t) is solvable. Then there exists an s-t Hamiltonian path $P = \langle p_1, \ldots, p_{|V|} \rangle$ in G. Let $P' = \langle f(p_1), f(p_2), \ldots, f(p_{|V|}) \rangle$. For $i = 2, \ldots, |V|$ the robot can now repeat the following: Move to the closest neighbor of p'_i relative to the robot, pick up the block, move one unit to where the object was before picking it up, and place it in the desired location.

Note that no object ever interferes with the robot's path. The total number of actions needed to achieve this is:

$$\left[\sum_{i=1}^{|V|-2} \underbrace{3}_{\text{move to a neighbor of } p'_i} + \underbrace{1}_{\text{pick up } \texttt{object}_{p_i}} + \underbrace{1}_{\text{place } \texttt{object}_{p_i}}\right]$$

$$+ \underbrace{1}_{\text{move to neighbor of } f(t)} + \underbrace{1}_{\text{pick up } \texttt{object}_t} + \underbrace{1}_{\text{move one unit closer to } \texttt{goal}(\texttt{object}_t)} + \underbrace{1}_{\text{place the object}}$$

$$= 5(|V|-2)+4 = K$$

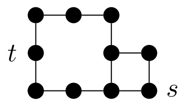
Assume now that $(s_0, \texttt{goal})$ is solvable in at most K steps. Let π be an optimal plan. Since π is optimal, it contains $|V|-1$ pickup actions, $|V|-1$ place actions and at least $3(|V|-1)+1$ movement actions. But since $|V|-1+|V|-1+3(|V|-1)+1 = K$ we have that π contains exactly $3(|V|-1)+1$ move actions. From this, we can see that G must have a Hamiltonian path starting at s, since otherwise, the robot would need at least $3(|V|-1)+1+3$ movement actions, which is impossible. Also, this Hamiltonian path must end in t, since otherwise we would need at least $3(|V|-1)+2$ movement actions. □

Lemma 13. *All solvable instances of* `IPC-TIDYBOT` *defined on rectangular grids of width and height at least two and with no obstacles have polynomial length plans.*

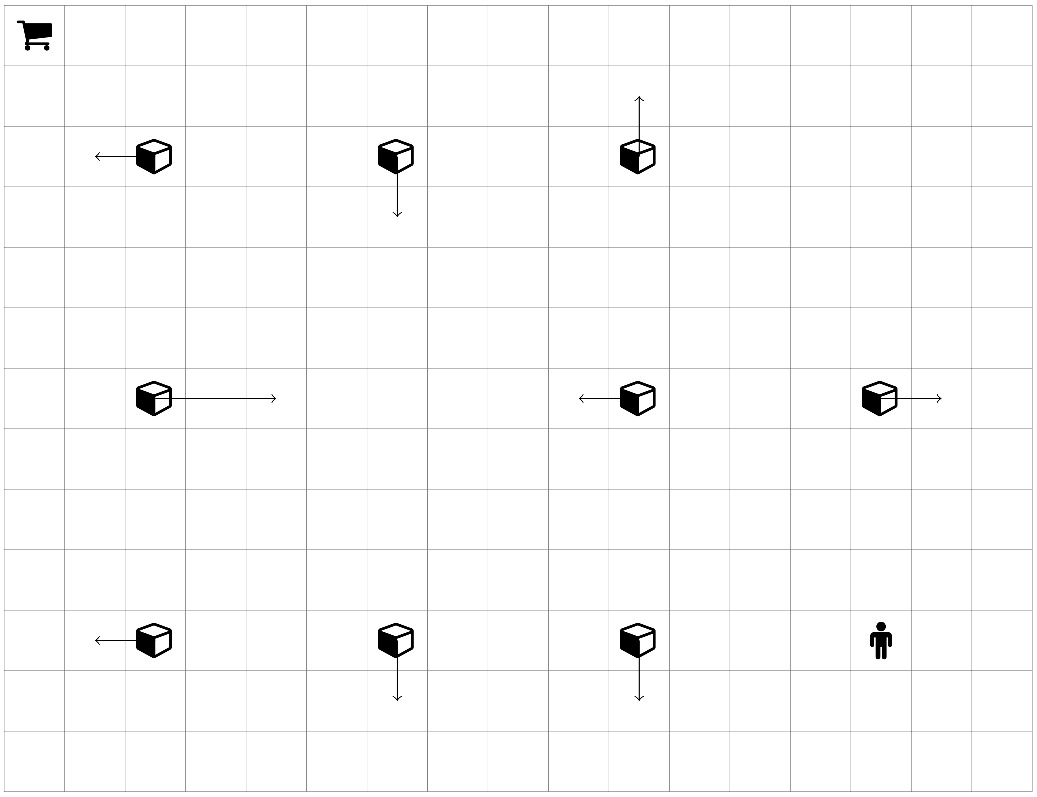
Proof. Let $(s_0, \texttt{goal})$ be a solvable `IPC-TIDYBOT` instance with no obstacles. Let π be a plan for $(s_0, \texttt{goal})$. We show that there exists a new plan π' of polynomial length. We first demonstrate that the robot can reach any cell in $V \setminus \{\texttt{cart\_pos}_{s_0}\}$ in a polynomial amount of steps by using a grab-move-place strategy without having to push the cart. The grab-move-place strategy is the following: if the robot is on a cell u and wants to move to a cell v without pushing the cart, consider a path $P = \langle p_1, \dots, p_n\rangle$ from u to v on the graph G' induced by $V \setminus \{\texttt{cart\_pos}_{s_0}\}$. Note that since the width and height of G are at least two, G' is connected, and thus P exists. For $i = 1, ..., n-1$ the robot can move from p_i to p_{i+1}, assuming p_{i+1} is occupied by an object by grabbing the object in p_{i+1}, moving to p_{i+1} and placing the object in p_i. The plan π' is divided into three phases. We begin defining the first phase. From the grab-move-place argument, it is easy to see that it is possible to put all of the objects in the grid inside the cart without having to push it in a polynomial amount of steps. Now we move on to the second phase. Let v be the position of the cart in $s_0[\pi]$. The robot can push the cart from its initial position to v in the now empty grid. It is easy to see that this can also be done in a polynomial amount of steps. Now we describe the third and final phase. Let $H = \langle h_1, \dots, h_{|V|}\rangle$ be a Hamiltonian path on G starting at $\texttt{cart\_pos}_{s_0}$. This path exists due to Lemma 1. For each vertex $u \in V$ let $\texttt{object}_u$ be the object in `Objects` that is placed in u in $s_0[\pi]$ provided that it exists. Now for $i = |V|, \dots, 2$ the robot now can check if $\texttt{object}_{h_i}$ exists, if so, he places it in h_i. By following this Hamiltonian path, the robot is never trapped and thus can place all of the objects. □

Corollary 2. `BPE-IPC-TIDYBOT` *is* ***NP****-complete if we add the additional constraint that there are no obstacles.*

Proof. Hardness is shown in Lemma 12, and membership to **NP** follows from Lemma 13. □



(a) The instance (G, s, t) of `DEG-3-GHP`



(b) The corresponding Tidybot instance. The arrows indicate where each object must be placed to achieve the goal. All the goal destinations except the one for the object that represents t are one unit away from their corresponding objects.

Figure 5.2: Example reduction for `BPE-IPC-TIDYBOT`.

In the remainder of this section, we build up the necessary definitions and lemmas that give rise to Corollary 4. The idea of using bipartite matching in order to prove Corollary 3 was given to us by Eppstein and Jeřábek [9].

Definition 23 (Simplified Tidybot task)**.** *A simplified Tidybot task is a tuple $\tau = (T, L, m)$ where*

1. *T is a finite set of transportation objects.*
2. *L is a finite set of locations such that $T \cap L = \emptyset$.*
3. *$m : T \to 2^L$ is a map that assigns a set of candidate locations to each object.*

A solution to τ is an injective function $a : T \to L$ such that $\forall t \in T : a(t) \in m(t)$. We say that τ is solvable if such an assignment function a exists.

Definition 24 and Theorem 7 are taken from Galil [10].

Definition 24 (Graph matching, maximal matching)**.** *Let $G = (V, E)$ be a finite bipartite graph. A matching of G is a subset M of edges such that no two edges in M are incident, i.e. they do not share their start or endpoints. M is maximal if there does not exist a matching M' with $|M'| > |M|$.*

Theorem 7. *Finding a maximal matching of a bipartite graph (V, E) can be done in time $O\big(|V| \cdot |E|\big)$.*

Corollary 3. *Deciding whether a simplified Tidybot task $\tau = (T, L, m)$ is solvable can be done in time $O\Big(|T|^2 + |L|^2 + |L| \cdot |T|\Big)$.*

Proof. Define the graph $G = (V, E)$ as follows: $V = T \cup L$ and $(t, l) \in E \iff l \in m(t)$. It is easy to see that G is bipartite. By the definitions of V and E, we immediately get that τ is solvable iff there exists a matching on G of cardinality $|T|$. The existence of such matching can be decided using Theorem 7 in time $O\big(|V| \cdot |E|\big) = O\Big((|T| + |L|) \cdot (|T| \cdot |L|)\Big) = O\Big(|T|^2 + |L|^2 + |L| \cdot |T|\Big)$. □

Definition 25 (cart range)**.** *Let $G = (V, E)$ be a rectangular graph of width and height at least two and u be a vertex in V. Define $\texttt{cart\_range}(u)$ as the set of vertices to which a cart placed in u can be pushed to after any number of actions. That is, if the cart is located in a corner, then $\texttt{cart\_range}(u) = \{u\}$. If the cart is located in the boundary of G but not in a corner, then its range is the whole boundary column or row that is incident to u. Finally, if the cart is an inner point of G, the cart range of u is V.*

Figure 5.3 shows the cart ranges of distinct points in a grid.

Theorem 8. *Let $I = (s_0, \texttt{goal})$ be an PE-IPC-TIDYBOT instance where S is the set on which $\texttt{goal} \subset \texttt{Objects}$ is defined, i.e. $\texttt{goal} : S \to 2^V$. Then I is solvable iff $\exists u \in \texttt{cart\_range}_{s_0}$ such that the simplified Tidybot task $\tau_u = \Big(S, V, s \mapsto \texttt{goal}(s) \backslash \{u\}\Big)$ is solvable.*

Proof. Assume I is solvable. Let π be a plan and u be the position of the cart in $s_0[\pi]$. Then clearly u is part of the cart's range and moreover, since there is no object placed in u, the task τ_u is solvable.
Now we argue in the other direction, we assume there exists a u in the cart's range such that τ_u is solvable. We now proceed to create a plan π for I. We define π in three phases in the same manner as in Lemma 13. In the first phase, the robot places all objects in the cart. In the second phase, the robot pushes the cart such that it is placed in u, this is possible since otherwise u would not be in the cart's range. Finally, the robot follows a Hamiltonian path starting at u and places all the objects from S to their targeted goal positions. □

Corollary 4. *PE-IPC-TIDYBOT can be solved in polynomial time.*

Proof. Follows from Theorem 8 and Corollary 3. □

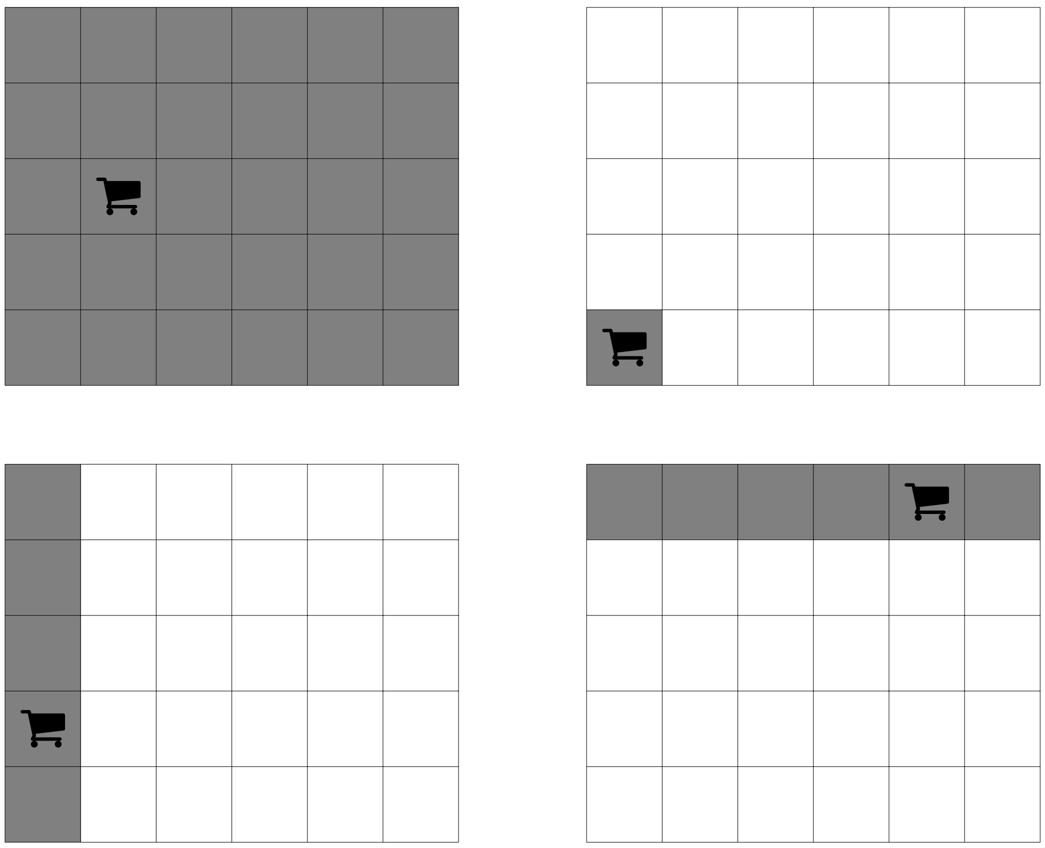


Figure 5.3: Cart ranges (colored in gray) for different values of u, indicated by the cart drawing.

5.2 PE-TIDYBOT is **PSPACE**-complete

We first show that PE-TIDYBOT is **NP**-hard. We reduce from 3CNF-SAT, given below.

3CNF-SAT
INPUT: A set $\mathcal{V} = \{x_1, \ldots, x_m\}$ of propositional variables, and a set $\mathcal{C} = \{C_1, \ldots, C_n\}$ of clauses over $\mathcal{V}$, where each C_i contains at most three literals, i.e. $C_i = \{c_{i,1}, c_{i,2}, c_{i,3}\}, c_{i,j} \in \mathcal{V} \cup \{\bar{x} : x \in \mathcal{V}\}$ $1 \leq i \leq n$.
QUESTION: Is there a satisfying assignment for $(\mathcal{V}, \mathcal{C})$?

The problem 3CNF-SAT is **NP**-complete [11].

Lemma 14. *PE-TIDYBOT is **NP**-hard even if there is only one robot.*

Proof. We reduce from 3CNF-SAT. Let $\Phi = (\mathcal{V}, \mathcal{C})$ be an instance of 3CNF-SAT, i.e. $\mathcal{V} = \{x_1, \ldots, x_M\}$ is a set of M propositional variables and $\mathcal{C} = \{C_1, \ldots, C_N\}$ is comprised of size 3 clauses over $\mathcal{V}$, namely $C_i = \{c_{i,1}, c_{i,2}, c_{i,3}\}, i = 1, \ldots, N$. We may assume that no clause contains a variable x_i and its negation $\bar{x}_i$, since then that clause is a tautology. We may also assume that every variable x_i in $\mathcal{V}$ appears in some clause either as x_i or in its negated form $\bar{x}_i$. We make use of the choice gadget, shown in Figure 5.4. Our construction for s_0 stacks M choice gadgets horizontally, one for each variable. We also add a corridor that is two units wide which allows the agent to reach each gadget. There are N objects $\texttt{object}_1, \ldots, \texttt{object}_N$ placed in the corridor. The robot is also placed in the corridor and has a grip radius of 1. For $1 \leq j \leq N$ let $\ell_{j_1,j}, \ell_{j_2,j}, \ell_{j_3,j}$ be the literals that appear in the clause C_j. We define $\texttt{goal}(\texttt{object}_j) = \{\ell_{j_1,j}, \ell_{j_2,j}, \ell_{j_3,j}\}$. An example of a construction of $s_0, \texttt{goal}$ for a fixed Φ is given in Figure 5.5. The construction can be computed in polynomial time since the height of the grid is fixed and its width is bounded by $M \cdot N \cdot C$ for some constant C.

Assume that Φ is satisfiable. We now construct a plan π for s_0. Let $A : \mathcal{V} \to \{\mathbf{T}, \mathbf{F}\}$ be a satisfying assignment for Φ. The agent proceeds to do the following for each variable x_i: if $A(x_i) = \mathbf{T}$, move to the choice gadget for x_i and push the cart one unit down, and if $A(x_i) = \mathbf{F}$, move to the choice gadget for x_i and push the cart one unit to the right. No cart will be moved again for the rest of the plan. Now it is easy to see that each object can be placed in one of its goal destinations since otherwise, A would not be a satisfying assignment.

Now assume that the Tidybt construction $(s_0, \texttt{goal})$ is solvable. Let s be a goal state. We define an assignment A on $\mathcal{V}$ as follows: for each variable x_i, if the cart in the choice gadget for x_i is pushed down in s or if the cart is not pushed at all, set $A(x_i) = \mathbf{T}$, otherwise if the cart is pushed to the right, set $A(x_i) = \mathbf{F}$. We now claim that A is a satisfying assignment for Φ. Assume by contradiction that A is not a satisfying assignment. Then there exists some clause $C_j = \{a, b, c\}$ such that none of the literals a, b, c are made true in A. But for s to be reachable, at least one of the cells marked with a_j, b_j, c_j must be reachable from s_0, thus pushing the cart in the appropriate direction, making that literal true. Hence a contradiction. □

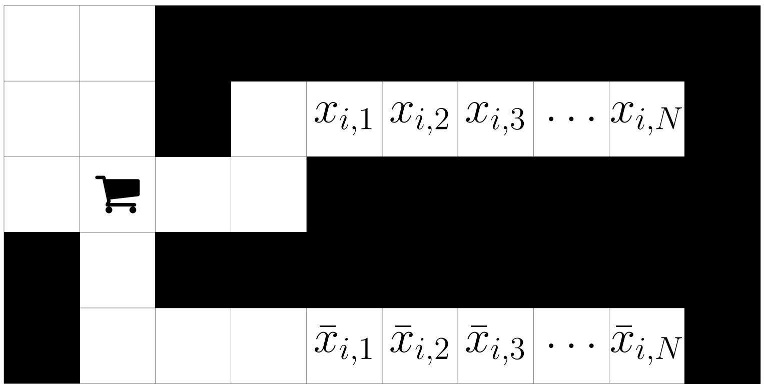


Figure 5.4: Choice gadget for the propositional variable x_i. The cells colored with black are base obstacles. Placing an object in the cell marked $x_{i,j}$ can be seen as making the literal x_i true in the clause C_j. Similarly, placing an object in the cell marked $\bar{x}_{i,j}$ can be seen as making the literal $\bar{x}_{i,j}$ true in the clause C_j. Note that carts cannot be pulled, so pushing the cart to the right makes it impossible for the robot to reach the cells marked with $x_{i,1}, \ldots, x_{i,N}$, and pushing the cart downwards makes it impossible to reach $\bar{x}_{i,1}, \ldots, \bar{x}_{i,N}$. This reflects the fact that an assignment for Φ cannot make a variable both true and false simultaneously.

The previous theorem seems to suggest that a source for Tidybot is the number of objects. However, the next proposition shows that Tidybot remains hard even if there is only 1 object.

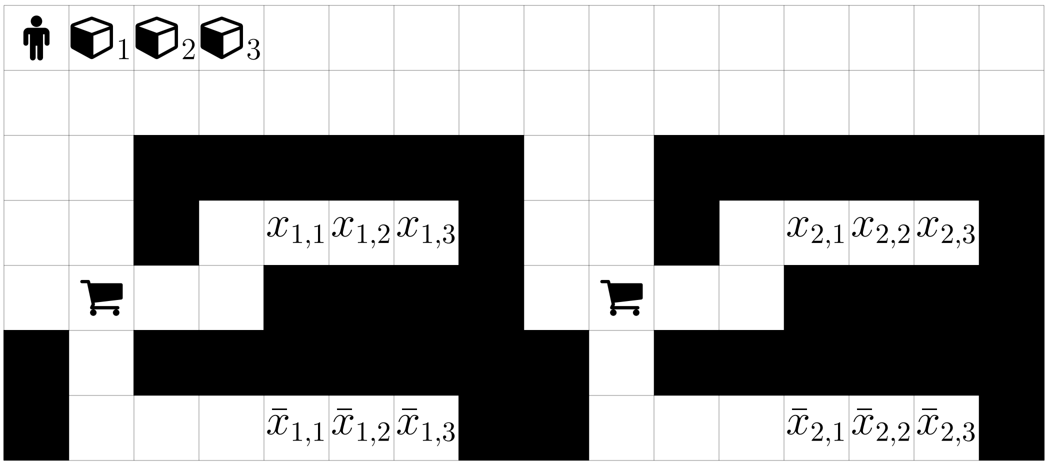

1
2
3
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$
$\bar{x}_{1,1}$ $\bar{x}_{1,2}$ $\bar{x}_{1,3}$
$\bar{x}_{2,1}$ $\bar{x}_{2,2}$ $\bar{x}_{2,3}$

Figure 5.5: Full reduction for $\Phi = (\{x_1, x_2\}, \{\{x_1, \bar{x}_2\}, \{\bar{x}_1, x_2\}, \{x_1\}\})$. We have $\texttt{goal}(o_1) = \{x_{1,c_1}, \bar{x}_{2,C_1}\}$, $\texttt{goal}(o_2) = \{\bar{x}_{1,C_2}, x_{2,C_2}\}$, $\texttt{goal}(o_3) = \{x_{1,C_3}\}$. The black-colored cells are base obstacles. There are no gripper obstacles.

Proposition 2. PE-IPC-TIDYBOT is **PSPACE**-complete even if there is only one single robot and only one object.

Proof. All problems of classical (propositional) planning are members of **PSPACE** [5]. Thus we need only show hardness. We do so by reducing from the puzzle Push -1F. This puzzle is a planning domain similar to Sokoban, with the only difference being that in Push -1F the goal is reached once the agent reaches a desired position in the grid, rather than when the blocks are put in some goal locations. The plan existence for Push -1F is **PSPACE**-complete [2].

The reduction is straightforward. Let $(s_0, \texttt{agent\_goal})$ be a Push -1F plan existence instance. Let $G_{P1F} = (V_{P1F}, E_{P1F})$ be the grid graph of s_0. We now define a plan existence I of Tidybot that is solvable iff $(s_0, \texttt{agent\_goal})$ is solvable.

Define $I = (s'_0, \texttt{goal})$ as follows:

1. The graph G of s_0 is $G = R(m, n)$ where $m = \texttt{width}(G_{P1F}), n = \texttt{height}(G_{P1F})$.
2. For each hole in G_{P1F} put a base obstacle in G.
3. I does not contain gripper obstacles.
4. The gripper radius is 1.
5. The robot location in I is the same as the robot location in $(s_0, \texttt{agent\_goal})$
6. There is only one object, which the robot is holding.
7. The goal is to put the object in $\texttt{agent\_goal}$.

This construction can clearly be done in polynomial time. It is easy to see that a plan for I exists iff a plan for $(s_0, \texttt{agent\_goal})$ exists. □

Action a	a **is applicable in** s **if**	**The effect of** a
$\texttt{MOVE\_BASE}(\texttt{robot}_i, u)$	$u \notin \texttt{base\_obstacles}_s \wedge$ $\big(\texttt{robots\_pos}_s(\texttt{robot}_i), u\big) \in E \wedge$ $\texttt{empty}_s(u)$	$\texttt{robots\_pos}_s(\texttt{robot}_i) \leftarrow u$
$\texttt{PUSH}(\texttt{robot}_i, \texttt{cart}_j, \texttt{dir})$	$\texttt{robot}_i + \texttt{dir} = \texttt{cart}_j \wedge$ $\texttt{empty}_s(\texttt{cart}_j + \texttt{dir}) \wedge$ $\texttt{cart}_j + \lambda \cdot \texttt{dir} \in V \setminus \texttt{base\_obstacles}$ $\lambda \in \{0, 1\}$	$\texttt{robots\_pos}(\texttt{robot}_i) \leftarrow \texttt{robot}_i - (1, 0) \wedge$ $\texttt{carts\_pos}(\texttt{cart}_j) \leftarrow \texttt{carts\_pos}(\texttt{cart}_j - (1, 0))$
$\texttt{PICKUP}(s)(\texttt{robot}_i, \texttt{object}_j, \texttt{dir}, d)$	$\texttt{robot}_i + d \cdot \texttt{dir} = \texttt{object}_j \wedge$ $\texttt{holding}_s(\texttt{robot}_i) = \texttt{none} \wedge$ $\texttt{robot}_i + \lambda \cdot \texttt{dir} \in V \setminus \texttt{gripper\_obstacles}$ $\lambda \in \{1, \ldots, d\}$	$\texttt{objects\_pos}(\texttt{object}_j) \leftarrow \texttt{robot}_i$
$\texttt{DROP}(\texttt{robot}_i, \texttt{object}_j, \texttt{dir}, d)$	$\texttt{empty}_s(\texttt{robot}_i + d \cdot \texttt{dir}) \wedge$ $\texttt{holding}_s(\texttt{robot}_i) = \texttt{object}_j \wedge$ $\texttt{robot}_i + \lambda \cdot \texttt{dir} \in V \setminus \texttt{gripper\_obstacles}$ $\lambda \in \{1, \ldots, d\}$	$\texttt{objects\_pos}(\texttt{object}_j) \leftarrow \texttt{robot}_i + d \cdot \texttt{dir}$

Table 5.1: Tidybot action preconditions and effects.

$\texttt{PICKUP\_FROM\_CART}(\texttt{robot}_i, \texttt{object}_j, \texttt{cart}_k, \texttt{dir}, d)$	$\texttt{carts\_pos}_s(\texttt{cart}_k) = \texttt{robot}_i + d \cdot \texttt{dir} \wedge$ $\texttt{objects\_pos}_s(\texttt{object}_j) = \texttt{cart}_k \wedge$ $\texttt{holding}_s(\texttt{robot}_i) = \texttt{none} \wedge$ $\texttt{robot}_i + \lambda \cdot \texttt{dir} \in V \setminus \texttt{gripper\_obstacles}$ $\lambda \in \{1, \dots, d\}$	$\texttt{objects\_pos}(\texttt{object}_j) \leftarrow \texttt{robot}_i$
$\texttt{DROP\_IN\_CART}(\texttt{robot}_i, \texttt{object}_j, \texttt{cart}_k, \texttt{dir}, d)$	$\texttt{carts\_pos}_s(\texttt{cart}_k) = \texttt{robot}_i + d \cdot \texttt{dir} \wedge$ $\texttt{holding}_s(\texttt{robot}_i) = \texttt{object}_j \wedge$ $\texttt{robot}_i + \lambda \cdot \texttt{dir} \in V \setminus \texttt{gripper\_obstacles}$ $\lambda \in \{1, \dots, d\}$	$\texttt{objects\_pos}(\texttt{object}_j) \leftarrow \texttt{cart}_k$

Table 5.2: Continuation of Table 5.2

6
Nurikabe

Nurikabe is the name of a Japanese logic puzzle. The name of the puzzle translates to 'painting wall', which refers to the puzzle's aim of coloring particular cells within some constraints. More precisely, a Nurikabe puzzle is comprised of a rectangular graph $G = (V, E)$, and a set of groups. A group is a pair (u, n) where u is a vertex in G and n is a natural number. The goal is to find a white-black coloring of G, i.e. a map $m : V \to \{\texttt{white}, \texttt{black}\}$ satisfying two conditions. The first is that the subgraph of G induced by the set of vertices $\{u : m(u) = \texttt{black}\}$ is connected, does not contain any 2×2 square subgraph, and does not contain any vertices belonging to a group. The second is that the subgraph induced by the set of vertices $\{u : m(u) = \texttt{white}\}$ is made out of N connected components, where N is the number of groups. For each group (u, n) there exists exactly one component C such that $u \in C$ and C has order n.

The IPC Nurikabe domain, which we call `IPC Nurikabe`, is a variant of the original Nurikable puzzle. It differs from Nurikabe in two ways. First, the painting constraints of the set of black cells are not enforced. Second, all cells are initially painted black, and there is an agent (a robot) that moves around the grid and can paint cells he is on with white. The actions of the robot are limited: he is not allowed to move to white-colored cells, he can only start painting cells with white when he is on a cell belonging to a group, and he can only stop painting once the number of cells painted for its respective group matches the group size.

Nurikabe is **NP**-complete [16], however, the relaxation of some of the painting constraints mixed with the added complexity of the agent makes a reduction from Nurikabe to `IPC Nurikabe` nontrivial. We start by formalizing a `IPC Nurikabe` states, goals, and decision problems.

Definition 26 (`IPC Nurikabe`, initial state)**.** *An* `IPC Nurikabe` *state is s is comprised of*

- *A square grid $G = R(n, n)$ for some $n \in \mathbb{N}$*
- *The position of the robot* $\texttt{robot}_s \in V$
- *A boolean variable that indicates if the robot is currently painting* $\texttt{painting}_s \in \{\mathbf{T}, \mathbf{F}\}$

Action a	a **is applicable in** s **if**	**Effect of** a
$\texttt{MOVE}(u)$	$(u, \texttt{robot}_s) \in E \land$ $\texttt{painting}_s = \mathbf{F} \land$ $\texttt{painted}_s(u) = \mathbf{F}$	$\texttt{robot} \leftarrow u$
$\texttt{MOVE\_AND\_PAINT}(u)$	$(u, \texttt{robot}_s) \in E \land$ $\texttt{painting}_s = \mathbf{T} \land$ $\texttt{painted}_s(u) = \mathbf{F}$	$\texttt{robot} \leftarrow u \land$ $\texttt{painted}(u) \leftarrow \mathbf{T}$
$\texttt{START\_PAINTING}$	$\exists n : (\texttt{robot}_s, n) \in \texttt{groups} \land$ $\texttt{painting}_s = \mathbf{F}$	$\texttt{painting} \leftarrow \mathbf{T} \land$ $\texttt{painted}(\texttt{robot}) \leftarrow \mathbf{T}$
$\texttt{STOP\_PAINTING}$	$\texttt{painting}_s = \mathbf{T} \land$ $\exists g \in \texttt{groups} : g$ is solved and $\texttt{robot}_s \in_s g$	$\texttt{painting} \leftarrow \mathbf{F}$

Table 6.1: IPC Nurikabe actions and preconditions for each $u \in V$. The notation $\in_s$ is introduced in Definition 29.

- *A function which indicates which vertices have been painted white* $\texttt{painted}_s : V \to \{\mathbf{T}, \mathbf{F}\}$
- *The set of groups* $\texttt{groups} \subset V \times \mathbb{N}_{>0}$

For a $(u, m) \in \texttt{groups}$, we say that u is the source of the group and m is the group size. All IPC Nurikabe initial states s_0 are such that $\texttt{painted}_{s_0}(u) = \mathbf{F}$ $\forall u \in V$, $\texttt{painting}_{s_0} = \mathbf{F}$ and for each $(u, n), (u', n') \in \texttt{groups}$ we have $d_{\ell_1}(u, u') \geq 2$, so in particular $u \neq u'$.

The actions and preconditions of IPC Nurikabe are shown in Table 6.1. Intuitively, a group (u, m) indicates that the robot must paint a simple path P starting at u of length m. If there are adjacent vertices v in P and v' in P' for distinct group paths P and P', we enter a dead state, i.e. a state from which the goal cannot be reached.

Definition 27 (coloring graph, group graph)**.** *Let s be an IPC Nurikabe state and $u \in V$. The coloring graph of s is the graph induced by the set of vertices $\{v : \texttt{painted}_s(v) = \mathbf{T}\}$. The group graph of u in s is the connected component of the coloring graph of s that includes u. If $\texttt{painted}_s(u) = \mathbf{F}$, we define its group graph to be the empty graph $(\emptyset, \emptyset)$.*

Definition 28 (solved group, solved state)**.** *Let s be an IPC Nurikabe state. A group (u, n) is solved in s if the group graph (V, E) for u in s has order n and if there does not exist another group (u', n') with $u' \in V$. We say that s is solved if all the groups of $g \in \texttt{groups}$ are solved in s.*

Definition 29 (membership to a group)**.** *Let s be a an IPC Nurikabe and $g \in \texttt{groups}$ whose group graph is (V, E). We say that a vertex u is a member g in s, written $u \in_s g$, if $u \in V$.*

The decision problems for IPC Nurikabe are given below.

PE-IPC-NURIKABE
INPUT: An initial IPC Nurikabe state s_0.
QUESTION: Is there a sequence of actions π applicable on s_0 such that $s_0[\pi]$ is a solved state?

BPE-IPC-NURIKABE

INPUT: An initial `IPC Nurikabe` state s_0 and a bound $K > 0$.

QUESTION: Is there a sequence of actions π applicable on s_0 such that $s_0[\pi]$ is a solved state and $\|\pi\| \leq K$?

6.1 BPE-IPC-NURIKABE is **NP**-complete

We show **NP** completeness of `IPC Nurikabe`. A straightforward argument shows that all solvable instances have polynomial length plans. **NP** hardness is shown by reducing from $\mathtt{GHP}_1$, which was introduced in Section 2.2.

Lemma 15. *Any solvable instance I of `IPC Nuriake` has a polynomial length plan w.r.t. $\|I\|$.*

Proof. Let s_0 be a solvable instance of `IPC Nurikabe` with N groups and π be a plan for s_0. Let G be the square grid of s_0. We construct a new plan π' of length $O(|V|^2)$. Let $g_1, \ldots, g_N$ be the ordering of `groups` where g_i is the i'th group painted in π. The plan π' is defined in N iterations as follows. In iteration i the robot moves to g_i and paints the group g_i in the same pattern as in π. Notice that moving to g_i takes at most $|V|$ actions, and painting the group g_i also takes at most $|V|$ actions. The fact that it is possible to move to g_i and paint the group g_i every time follows from the fact that π is a plan. □

Lemma 16. *BPE-IPC-NURIKABE is **NP**-complete.*

Proof. Membership to **NP** is proven in Lemma 15. We show **NP** hardness by reducing from $\mathtt{DEG\text{-}4\text{-}GHP}_1$. This problem takes a grid graph G, and vertices $s \neq t$ as input such that t has degree one. It asks whether there exists an s-t Hamiltonian path in G. We have shown that this problem is **NP**-complete in Lemma 2.

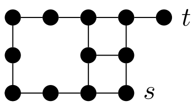
Let (G, s, t) be a $\mathtt{DEG\text{-}4\text{-}GHP}_1$ instance where $G = (V, E)$. We construct an instance $I = (s_0, \mathtt{goal}, K)$ as follows.

- Let $G' = R(n, n)$ be the grid of s_0 where $n = 2 \cdot \max\{\mathtt{wdith}(G), \mathtt{height}(G)\}$.
- Place the robot in $2 \cdot s$
- Let $\mathtt{groups} = \{(2 \cdot u, 1) : u \in V\}$
- $\mathtt{painted}_{s_0}(u) = \mathbf{F}\ \forall u \in V'$
- $\mathtt{painting}_{s_0} = \mathbf{F}$
- $K = 4|V| - 2$

An example of this construction is shown in Figure 6.1. We now argue that G has an s-t Hamiltonian path iff s_0 can be solved in at most K steps. Assume first that G has a s-t Hamiltonian path $P = \langle p_1, \ldots, p_{|V|}\rangle$. We construct a plan π for s_0 as follows. On iteration $i \in \{1, \ldots, |V|\}$ we move the robot to $2 \cdot p_i$ and paint the group the robot is on. The number of actions on iteration 1 are one for painting and one to stop painting, on all other iterations the robot must also move two squares. In total we have

$$2 + \sum_{i=1}^{|V|-1} 4 = 4|V| - 2 = K$$

Next we assume that s_0 can be solved in at most K steps. Note that we have $|V|$ groups so we need $2|V|$ actions to paint each group. Moreover, since the distance between each group is at least two we also need at least $2|V|-2$ movement actions. But since $2|V|+2|V|-2 = K$, the requirement on the number of movement actions is tight. Because of this, the robot must move two squares between each group painted, which is only possible if he follows a Hamiltonian path. Further, this path must end in $2 \cdot t$ since $\deg(t) = 1$. $\square$



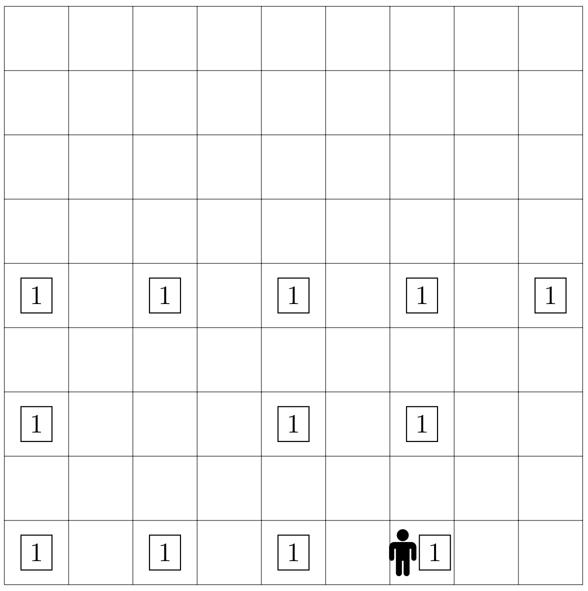


Figure 6.1: An instance (G, s, t) of DEG-4-GHP$_1$ on the right and the corresponding BPE-IPC-NURIKABE instance on the right.

6.2 PE-IPC-NURIKABE is **NP**-complete

We recall the problem DEG-3-SHP introduced in Section 1.1.4. It asks if a given subgrid graph (i.e. a connected edge induced subgraph of the integer lattice) whose vertices have degree at most three has an s-t Hamiltonian path. In this section, we consider a special case of this problem, where s and t have degree one, and t's neighbor has degree 2. We show that this restricted variant remains **NP**-complete. For an instance (G, s, t) of this problem, we engineer an initial IPC Nurikabe state s_0 in the following way. There is a group (u, n) where u depends on s and n is a large number that depends on the order of G. There are also many groups of the form $(v, 1)$, scattered around the grid of s_0 where the values for v depend on the other vertices of G. Intuitively, these 1-groups force the robot to paint (u, n) in such a way that its coloring is a scaled-up s-t Hamiltonian path from G on the grid of s_0.

Until now, we always labeled (sub) grid graphs such that their smallest x and y coordinates are 1. For simplicity, we make a change of convention in this section only, in that we label graphs such that the smallest x and y coordinates of a grid are 0 instead.
We define the following decision problem:

DEG-3-SHP$_3^*$
INPUT: A subgrid graph $G = (V, E)$ and $s \neq t \in V$ such that $\deg(t) = \deg(s) = 1$, $\deg(u) \in \{2, 3\}$ $\forall u \in V \setminus \{s, t\}$, and the neighbor of t has degree 2.
QUESTION: Is there an s-t Hamiltonian path in G?

Lemma 17. *DEG-3-SHP$_3^*$ is **NP**-complete.*

Proof. For an instance (G, s, t) of DEG-3-SHP$_3^*$, a non-deterministic algorithm can guess a sequence of vertices of length $|V|$ and check whether it forms an s-t Hamiltonian path in polynomial time. In the remainder of this text, we show that

$$\texttt{DEG-3-SHC} \leq \texttt{DEG-3-SHP}_3^*$$

which finishes the proof since DEG-3-SHC is **NP**-complete according to Theorem 3. Let $G_1 = (V_1, E_1)$ be an instance of DEG-3-SHC with $|V_1| > 2$. If G_1 contains a degree 1 vertex, then G_1 cannot have a Hamiltonian cycle, so we thus may assume that all vertices of G_1 have degree either 2 or 3. We now provide a polynomial time construction of an instance (G_3, s, t) of DEG-3-SHP$_3^*$ that can be solved iff G_1 has a Hamiltonian cycle. Let s be the right-top corner of G_1, i.e. s is at the top of the right-most column. Since s is a corner vertex it must have degree two. Then s has a neighbor below it, we call it u. Let $G_2 = \big(V_1, E_1 \setminus \{(s, u), (u, s)\}\big)$. Define two new vertices $t_0 = \big(u + (1, 0)\big)$, $t = \big(u + (2, 0)\big)$ and let $G_3 = (V_2 \cup \{t_0, t\}, E_2 \cup \{(u, t_0), (t_0, u), (t_0, t), (t, t_0)\})$. An example of this is shown in Figure 6.2. It is easy to see that this construction can be done in polynomial time and that (G_3, s, t) is a valid instance of DEG-3-SHP*. We now argue that

$$(i)\ G_1 \text{ has a Hamiltonian Cycle}$$

$$\Longleftrightarrow$$

$$(ii)\ G_2 \text{ has an } s\text{-}u \text{ Hamiltonian Path}$$

$$\Longleftrightarrow$$

$$(iii)\ G_3 \text{ has an } s\text{-}t \text{ Hamiltonian Path}$$

- Proof of $(i) \iff (ii)$. Note that G_1 has a Hamiltonian cycle iff G_1 has an s-u Hamiltonian path P. This path cannot make use of the edge (s, u) because u must be visited last and may thus be removed.

- Proof of $(ii) \iff (iii)$. Trivial since t's only neighbor is only adjacent to t and u.

□

The remainder of this chapter is devoted to defining a polynomial time transformation T from DEG-3-SHP$_3^*$ and proving its correctness.

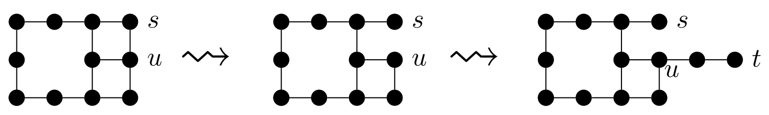


Figure 6.2: Transformation of graphs G_1, G_2 and G_3.

Definition 30 (Scaling function f). *We define the scaling function*

$$f : \mathbb{Z}^2 \to \mathbb{Z}^2$$

$$f(x, y) = (4x + 4, 4y + 4)$$

Definition 31 (Degree k vertex counting function). *Let $G = (V, E)$ be a simple graph, $U \subseteq V$ and $k \in \mathbb{N}$, define the degree k counting function on G as*

$$D^k_G(U) = |\{u \in U : \deg(u) = k\}|$$

Definition 32 (Blockade vertices, bridge vertices). *Let (G, s, t) be a* DEG-3-SHP* *instance with $G = (V, E)$. We define the following sets*

- $\texttt{Left\_blockades} = \{f(u) - (2, 0) : u \in V, \big(u, u - (1, 0)\big) \notin E\}$
- $\texttt{Right\_blockades} = \{f(u) + (2, 0) : u \in V, \big(u, u + (1, 0)\big) \notin E\}$
- $\texttt{Top\_blockades} = \{f(u) + (0, 2) : u \in V, \big(u, u + (0, 1)\big) \notin E\}$
- $\texttt{Bottom\_blockades} = \{f(u) - (0, 2) : u \in V, \big(u, u - (0, 1)\big) \notin E\}$
- $\texttt{Horizontal\_bridges} = \{f(u) + (\pm 2, 2) : \big(u, u + (1, 0)\big) \in E\}$
- $\texttt{Vertical\_bridges} = \{f(u) + (2, \pm 2) : \big(u, u + (0, 1)\big) \in E\}$

Definition 33 (`IPC Nurikabe` transformation T). *Let $I = (G, s, t)$ be an* DEG-3-SHP* *instance where $G = (V, E)$. We define a transformation $T(I)$ from I to an initial Nurikabe state $s_0 = (G', \texttt{robot}_{s_0}, \texttt{painting}_{s_0}, \texttt{painted}_{s_0}, \texttt{groups})$ as follows.*

- $G' = R(n, n)$ *where* $n = 4 \cdot \max\{\texttt{width}(G), \texttt{height}(G)\} + 12$
- $\texttt{robot}_{s_0} = (0, 0)$
- $\texttt{painting}_{s_0} = \mathbf{F}$
- $\texttt{painted}_{s_0}(u) = \mathbf{F}\ \forall u \in V'$
- $\texttt{groups} = \big\{(f(s), 4 \cdot |V| + 2 \cdot D^3_G(V) - 3)\big\} \cup \big\{(u, 1) : u \in S\big\}$

where S is the set given by the union

$$S = \texttt{Blockades} \cup \texttt{Bridges} \subset V'$$

which decomposes further into

$$\texttt{Blockades} = \texttt{Left\_blockades} \cup \texttt{Right\_blockades} \cup \texttt{Top\_blockades} \cup \texttt{Bottom\_blockades}$$

and

$$\texttt{Bridges} = \texttt{Horizontal\_bridges} \cup \texttt{Vertical\_bridges}$$

A graphical example of the transformation T is shown in Figure 6.3. We have built a program that constructs such instances[2].

Proposition 3. Let $I = (G, s, t)$ be an $\texttt{SHP}_3^*$ instance. Then $T(I)$ is well-defined and can be computed in polynomial time w.r.t $\|I\|$.

Proof. Trivial since the dimensions of the grid in $T(I)$ are linear with respect to $\|T\|$, and determining whether an $(u, n) \in \texttt{groups}$ can also be done in linear time w.r.t $\|I\|$. □

We now turn to proving the correctness for T. Our strategy is the following. For an instance $I = (G, s, t)$ of $\texttt{DEG-3-SHP}^*$, and $s_0 = T(I)$, we consider the graph G_T induced by the set of vertices that the robot could paint in order to solve the group whose source is $f(s)$. We call this graph the T graph of s_0 and make claims about the relationship between simple paths in G and simple paths in G_T that eventually prove the hard direction of Theorem 21.

Definition 34 (1-blocked vertex)**.** *Let s_0 be an* `IPC Nurikabe` *state. We say that a vertex $u \in V$ is 1-blocked if $(u, 1) \in \texttt{groups}$ or if u is adjacent to a vertex v such that $(v, 1) \in \texttt{groups}$.*

Definition 35 (T-reachable vertex)**.** *Let $I = (G, s, t)$ be an* $\texttt{DEG-3-SHP}_3^*$ *instance where $G' = (V', E')$ is the square grid of $T(I)$. A vertex $u' \in V'$ is T-reachable if there exists a path (not necessarily simple) from $f(s)$ to u' that does not pass through a 1-blocked vertex.*

Remark 1. Let $I = ((V, E), s, t)$ be an $\texttt{DEG-3-SHP}^*$ instance and $s_0 = T(I)$. The set of all the T-reachable vertices is

$$\texttt{Corresponding\_vertices} \cup$$

$$\texttt{Corresponding\_vertical\_edges} \cup$$

$$\texttt{Corresponding\_horizontal\_edges} \cup$$

$$\texttt{Horizontal\_tentacles} \cup$$

$$\texttt{Vertical\_Tentacles}$$

where

- $\texttt{Corresponding\_vertices} = \{f(u) : u \in V\}$
- $\texttt{Corresponding\_vertical\_edges} = \big\{f(u) + (0, i) : 1 \le i \le 3, \big(u, u + (0, 1)\big) \in E\big\}$
- $\texttt{Corresponding\_horizontal\_edges} = \big\{f(u) + (i, 0) : 1 \le i \le 3, \big(u, u + (1, 0)\big) \in E\big\}$
- $\texttt{Horizontal\_tentacles} = \big\{f(u) + (i, j) : i \in \{1, 3\}, j \in \{1, -1\}, \big(u, u + (1, 0)\big) \in E\big\}$
- $\texttt{Vertical\_tentacles} = \big\{f(u) + (i, j) : i \in \{1, -1\}, j \in \{1, 3\}, \big(u, u + (0, 1)\big) \in E\big\}$

[2] This can be found online under https://github.com/TravisPetit/Nurikabe.

Definition 36 (T-graph, T'-graph)**.** *Let $I = (G, s, t)$ be an* DEG−3−SHP* *instance and $s_0 = T(I)$. The T-graph of s_0, written G^I_T is the subgraph of the grid of s_0 induced by the vertex set $\{u : u \text{ is } T\text{-reachable}\}$. The T'-graph of s_0, written $G^I_{T'}$ is the subgraph of G^I_T induced by the set of vertices $\{u : \deg_{G^I_T}(u) \geq 2, u \neq f(s), u \neq f(t)\}$*

Remark 2. Let $I = (G, s, t)$ be an DEG−3−SHP* instance where the only vertices with degree one are s and t, and let $s_0 = T(I)$. The T' graph of s_0 is a grid graph induced by the union of the following sets of vertices:

- `Corresponding_vertices`
- `Corresponding_vertical_edges`
- `Corresponding_horizontal_edges`
- $\{f(u) + (1, 1) : \big(u, u + (1, 0)\big) \in E \wedge \big(u, u + (0, 1)\big) \in E\}$
- $\{f(u) + (-1, 1) : \big(u, u + (-1, 0)\big) \in E \wedge \big(u, u + (0, 1)\big) \in E\}$
- $\{f(u) + (-1, -1) : \big(u, u + (-1, 0)\big) \in E \wedge \big(u, u + (0, -1)\big) \in E\}$
- $\{f(u) + (1, -1) : \big(u, u + (1, 0)\big) \in E \wedge \big(u, u + (0, -1)\big) \in E\}$

We note that in the T'-graph $G^{(G,s,t)}_{T'}$ of an instance $T((G, s, t))$, a vertex u in G corresponds to a set of vertices in $G_{T'}$, which can be seen clearly in Figure 6.3. This motivates the following definitions.

Definition 37 (bridge midpoint)**.** *Let $I = (G, s, t)$ be an* DEG−3−SHP* *instance with $G = (V, E)$ and $s_0 = T(I)$. Let $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of s_0. The bridge midpoint of an $e \in E$, written $m(e)$ is given below:*

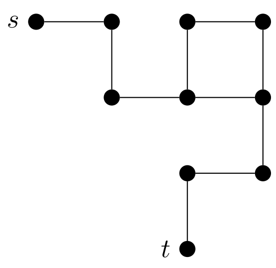
$$m(e) = \begin{cases} f(u) + (2, 0) & e = \big(u, u + (1, 0)\big) \text{ for some } u \in V \\ f(u) + (0, 2) & e = \big(u, u + (0, 1)\big) \text{ for some } u \in V \end{cases}$$

Definition 38 (corresponding vertex zone)**.** *Let $I = (G, s, t)$ be a some* DEG−3−SHP* *instance with $G = (V, E)$, and let $s_0 = T(I)$, $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of I, and $u \in V$. The corresponding vertex zone of u, written $z(u)$ is the following set of vertices*

$$z(u) = \{v \in V_{T'} : \exists \text{ a path } P \text{ in } G_{T'} \text{ from } u \text{ to } v \text{ with } \|P\| \leq 3\} \backslash \{m(e) : e \text{ is incident to } u\}$$

Remark 3. Let $I = (G, s, t)$ be a some DEG−3−SHP* instance with $G = (V, E)$, and let $s_0 = T(I)$, $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of I, and $u, v, w \in V$ be a vertices with degree $1, 2,$ and 3 respectively in G. An easy computation shows that

$$z(u) = \begin{cases} \{f(u) + (1, 0)\} & \big(u, u + (1, 0)\big) \in E \\ \{f(u) + (0, 1)\} & \big(u, u + (0, 1)\big) \in E \\ \{f(u) + (-1, 0)\} & \big(u, u + (-1, 0)\big) \in E \\ \{f(u) + (0, -1)\} & \big(u, u + (0, -1)\big) \in E \end{cases}$$



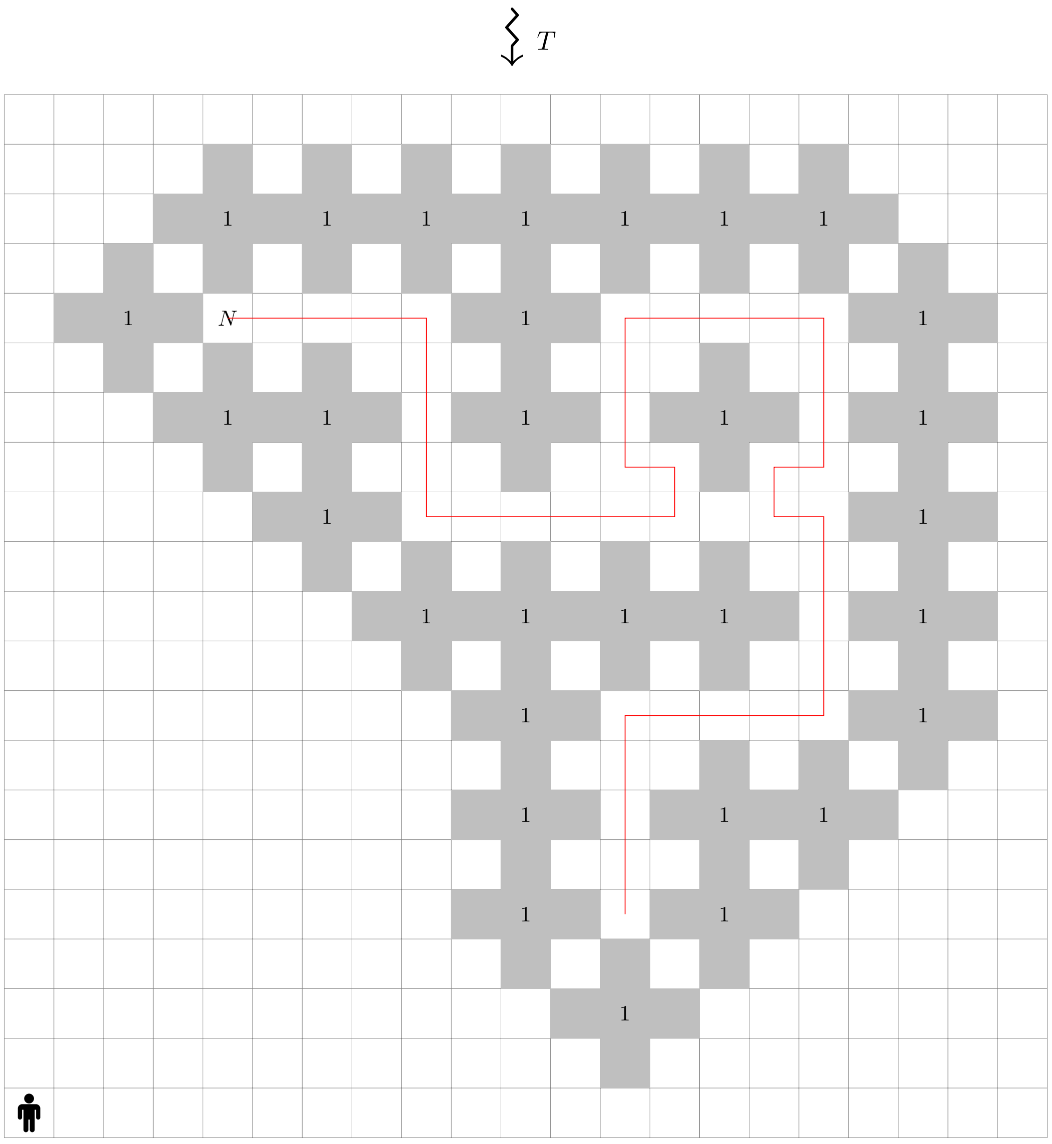


Figure 6.3: Example transformation from an instance $I = (G, s, t)$ of `DEG-3-SHP*` to the corresponding `IPC Nurikabe` state $T(I)$. The cells colored in gray are 1-blocked. The value of N is $4 \cdot |V| + 2 \cdot D^3_G(V) - 3 = 4 \cdot 10 + 2 \cdot 2 - 3 = 41$. The path traced in red is a possible coloring for the group $(f(s), N)$.

$$z(v) = \begin{cases} \{f(v) + (0,y) : -1 \leq y \leq 1\} & \big(v, v+(0,1)\big) \in E \wedge \\ & \big(v, v+(0,-1)\big) \in E \\ \{f(v) + (x,0) : -1 \leq x \leq 1\} & \big(v, v+(1,0)\big) \in E \wedge \\ & \big(v, v+(1,0)\big) \in E \\ \{f(v) + (x,y) : 0 \leq x \leq 1, 0 \leq y \leq 1\} & \big(v, v+(1,0)\big) \in E \wedge \\ & \big(v, v+(0,1)\big) \in E \\ \{f(v) + (x,y) : -1 \leq x \leq 0, 0 \leq y \leq 1\} & \big(v, v+(-1,0)\big) \in E \wedge \\ & \big(v, v+(0,1)\big) \in E \\ \{f(v) + (x,y) : -1 \leq x \leq 0, -1 \leq y \leq 0\} & \big(v, v+(-1,0)\big) \in E \wedge \\ & \big(v, v+(0,-1)\big) \in E \end{cases}$$

$$z(w) = \begin{cases} \{f(w) + (x,y) : -1 \leq x \leq 1, -1 \leq y \leq 0\} & \big(w, w+(-1,0)\big) \in E \wedge \\ & \big(w, w+(0,-1)\big) \in E \wedge \\ & \big(w+(0,1)\big) \in E \\ \{f(w) + (x,y) : -1 \leq x \leq 1, 0 \leq y \leq 1\} & \big(w, w+(-1,0)\big) \in E \wedge \\ & \big(w, w+(0,1)\big) \in E \wedge \\ & \big(w+(0,1)\big) \in E \\ \{f(w) + (x,y) : 0 \leq x \leq 1, -1 \leq y \leq 1\} & \big(w, w+(0,-1)\big) \in E \wedge \\ & \big(w, w+(0,1)\big) \in E \wedge \\ & \big(w+(1,0)\big) \in E \\ \{f(w) + (x,y) : -1 \leq x \leq 0, -1 \leq y \leq 1\} & \big(w, w+(-1,0)\big) \in E \wedge \\ & \big(w, w+(0,-1)\big) \in E \wedge \\ & \big(w+(1,0)\big) \in E \end{cases}$$

Remark 4. Let $I = (G, s, t)$ be an `DEG-3-SHP`* instance with $G = (V, E)$ and $s_0 = T(I)$. Let $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of s_0. The following family of sets is a disjoint partition of $V_{T'}$

$$\{z(u) : u \in V\} \cup \{\{m(e)\} : e \in E\}$$

From the previous remark we can see that for a simple path to visit a vertex in a corresponding vertex zone, it must do so by first visiting a bridge midpoint. This is used to define the notion of entering and leaving a corresponding vertex zone.

Definition 39 (entering/leaving a corresponding vertex zone)**.** *Let $I = (G, s, t)$ be an `DEG-3-SHP`* instance with $G = (V, E)$ and $s_0 = T(I)$. Let $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of s_0. Let $u \in V$. We say that a simple path $P = \langle p_1, \ldots, p_n \rangle$ in $G_{T'}$ enters $z(u)$ if there exists an $1 \leq i \leq n$ such that p_k is a bridge midpoint and $p_{k+1} \in z(u)$. Similarly, we say that P leaves $z(u)$ if there exists an $1 \leq i \leq n$ such that $p_k \in z(u)$ and p_{k+1} is a bridge midpoint.*

Lemma 18. *Let $I = (G, s, t)$ be an* DEG-3-SHP* *instance with $G = (V, E)$ and $s_0 = T(I)$. Let $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of s_0 and P be a simple path in $G_{T'}$ starting in $f(s)$. Let $u \in V$ with $\deg_G(u) = k, u \neq s$. The following hold:*

a) If $k = 1$, P cannot leave $z(u)$.

b) If $k = 2$, P cannot enter $z(u)$ twice.

c) If $k = 3$, P cannot enter $z(u)$ twice unless P ends in a vertex of $z(u)$.

Proof. Note that by definition P must enter and leave $z(u)$ through a bridge midpoint, and a vertex u of degree k has k bridge midpoints adjacent to one of the vertices in $z(u)$. Next, since P must enter $z(u)$ before leaving it we have that P can enter $z(u)$ at most $\lceil \frac{k}{2} \rceil$ times, and leave it $\lfloor \frac{k}{2} \rfloor$ times. Setting $k = 1, 2, 3$ proves *a*), *b*), and *c*) respectively. □

Clearly, a path P as defined in Lemma 18 cannot enter $z(s)$, thus, it may enter at most $|V| - 1$ corresponding vertex zones.

Proposition 4. Let $I = (G, s, t)$ be an DEG-3-SHP* instance with $G = (V, E)$ and $s_0 = T(I)$. Let $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of s_0 and P be a simple path in $G_{T'}$ starting in $f(s)$. Then P can enter at most $|V| - 1$ distinct corresponding vertex zones, and it can enter corresponding vertex zones at most $|V| - 1$ times.

Proof. P cannot enter $z(s)$ since it starts in $f(s)$ and $\deg_G(s) = 1$, thus it may enter at most $|V| - 1$ corresponding vertex zones. Furthermore, if the last corresponding vertex zone $z(u)$ that P enters is such that $\deg_G(u) = 3$, then P could enter $z(u)$ twice, however, since G has a vertex of degree 1 other than s, it would enter $|V| - 2$ corresponding vertex zones and would enter a total amount of $|V| - 2 + 1$ times. □

From the previous two results, the next remark is immediate.

Remark 5. Let $I = (G, s, t)$ be an DEG-3-SHP* instance with $G = (V, E)$ and $s_0 = T(I)$. Let $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of s_0. Let P be a simple path in $G_{T'}$ starting at $f(s)$, and let $z(u_1), \ldots z(u_n)$ be the corresponding vertex zones that P enters (in that order). Then if $\deg_G(u_n) \neq 3$, we get that $\langle s, u_1, \ldots, u_n \rangle$ is a simple path in G.

Lemma 19. *Let $I = (G, s, t)$ be an* DEG-3-SHP* *instance with $G = (V, E)$ and $s_0 = T(I)$. Let $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of s_0. Let $u \in V$ with $\deg_G(u) = k$, $k \in \{2, 3\}$, and $e_1, \ldots, e_k$ be the edges incident to u. Let $G_{z(u)}$ be the subgraph of $G_{T'}$ induced by the following set of vertices: $z(u) \cup \{m(e_i) : 1 \leq i \leq k\}$. Let $L_{i,j}$ be the length of the longest path from one of $m(e_i)$ to one of $m(e_j)$ in $G_{z(u)}$, $i \neq j \in \{1, \ldots, k\}$. Then*

1. If $k = 2$, $L_{i,j} = 5 \; \forall i, j$

2. If $k = 3$, $L_{i,j} = 7 \; \forall i, j$

Proof. By inspection. Possible longest simple paths are drawn in Figure 6.5 up to symmetries in rotations and relfections. □

Lemma 20. *Let $I = (G, s, t)$ be an* DEG-3-SHP* *instance with $G = (V, E)$ and $s_0 = T(I)$. Let $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of s_0. Let $P = \langle p_1, \ldots, p_m\rangle$ be a simple path in G' that starts in $f(s)$ and ends in a bridge midpoint. Let $z(u_1), \ldots, z(u_n)$ be the corresponding vertex zones that P enters. Then $\|P\| \leq 4n + 2 \cdot D^3_G(\{u_1, \ldots, u_n\}) + 3$*

Proof. We notice that each of $u_1, \ldots, u_n$ must have degree at least two since otherwise, P could not end in a bridge midpoint. Further, it is easy to see that there is only one simple path from $f(s)$ to $b\big((s, u_1)\big)$ in $G_{T'}$, and it has length 3. For $1 \leq i < n$ let $e_i = (u_i, u_{i+1})$. By Lemma 5 we get that $e_i \in E\ \forall i$. Further, let e_n be an edge incident to u_n other than e_{n-1}. Let

$$\ell_i = \begin{cases} 4 & \deg_G(u_i) = 2 \\ 6 & \deg_G(u_i) = 3 \end{cases}$$

Using Lemma 19 we get

$$\|P\| \leq 3 + \sum_{i=1}^{n} \ell_i = 3 + 4n + 2 \cdot D^3_G\big(\{u_1, \ldots, u_n\}\big)$$

as desired. □

Lemma 21. *Let $I = (G, s, t)$ be an* DEG-3-SHP* *instance with $G = (V, E)$ and $s_0 = T(I)$. Let $G_{T'} = (V_{T'}, E_{T'})$ be the T' graph of s_0. Let $P = \langle p_1, \ldots, p_m\rangle$ be a simple path in G' sthat starts in $f(s)$ of length at least $L = 4 \cdot (|V| - 1) + 2D^3_G(V) + 1$ and $z(u_1), \ldots, z(u_n)$ be the corresponding vertex zones that P visits. Then $\langle s, u_1, \ldots, u_n\rangle$ is an s-t Hamiltonian path in G.*

Proof. Let $z(u)$ be the last corresponding vertex zone that P enters. We first show that $u = t$. Assume by contradiction that $u \neq t$. We make two case distinctions.

- Case 1: $\deg(u) = 2$. We have that P cannot enter any vertex twice by Lemma 18. Moreover, P cannot have entered $z(t)$ since it would have to leave it. Hence P can enter at most $|V| - 2$ corresponding vertex zones, without repetitions. Then by Lemma 20
$$\|P\| \leq 4 \cdot (|V| - 2) + 2 \cdot D^3_G(V) + 3 = 4 \cdot (|V| - 1) + 2 \cdot D^3_G(V) - 1 < L$$
- Case 2: $\deg(u) = 3$. Let v be the neighbor of t. We have that P cannot enter $z(t)$ nor $z(v)$, so P can enter at most $|V| - 3$ corresponding vertex zones. However, P could enter $z(v)$ twice, thus entering corresponding vertex zones at most $|V| - 2$ times. Using Lemma 20 we get the same bound as above.

Next, we let $e = (u_{n-1}, u_n)$ and notice that $e \in E$ due to Lemma 5. We let $1 \leq m \leq i$ be the index such that $p_i = m(e)$ and $P' = \langle p_{i+1}, \ldots, p_m\rangle$. By Lemma 20 we have

$$\|P\| \leq 4(n-1) + 2D^3_G(V) + 3 + \|P'\|$$

And since $\|P'\| \leq |z(u_n)| = |z(t)| = 2$ we get

$$\|P\| \leq 4n + 2D^3_G(V) + 1$$

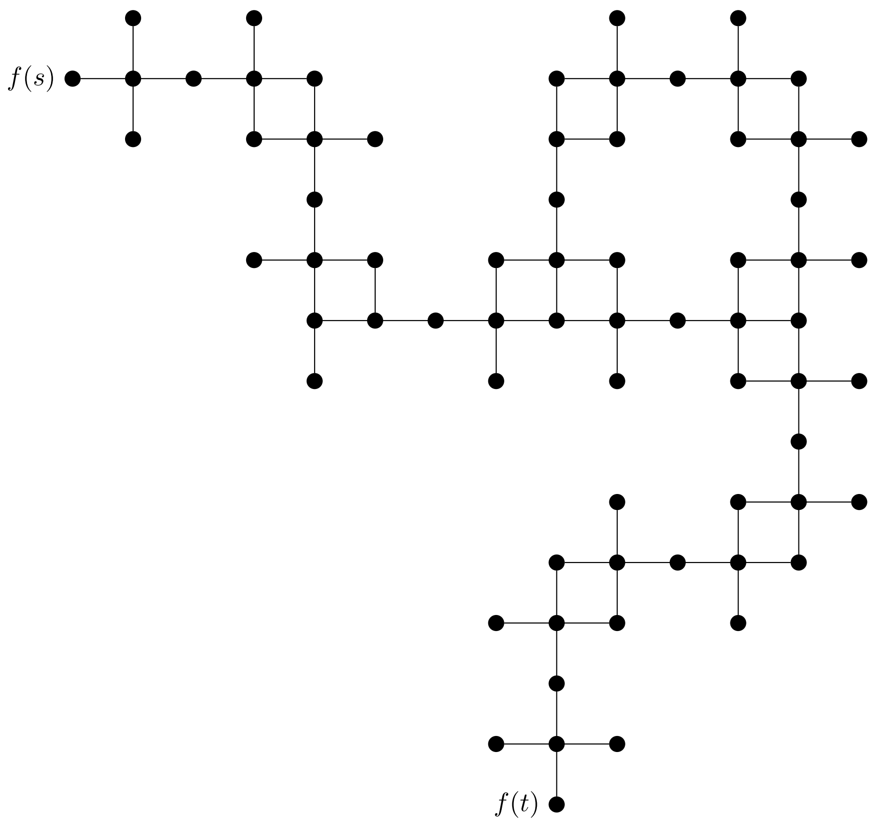
Thus $n = |V| - 1$ so P enters $|V| - 1$ distinct corresponding vertex zones. We conclude using Lemma 5. □

Theorem 9. *`PE-IPC-NURIKABE` is **NP**-complete.*

Proof. Membership to **NP** follows from Lemma 15. We reduce from `DEG-3-SHP`* using the transformation T from Definition 33. We know that T can be computed in polynomial time due to Proposition 3. Let $I = (G, s, t)$ be an `DEG-3-SHP`* instance and $s_0 = T(I)$ and let $G_{s_0} = (V_{s_0}, E_{s_0})$ be the square grid of s_0. We show that I is solvable iff s_0 is solvable.
Assume first that s_0 is solvable. We show that I is solvable too. This follows from Lemma 21.
Assume now that I is solvable, i.e. there exists an s-t Hamiltonian path $P = \langle p_1, \dots, p_n \rangle$ in G. We devise an algorithm that produces a plan π for s_0 as follows: Note that for each $(u, 1), (v, 1) \in$ `groups` we have that $d_{\ell_1}(u, v) > 1$, so in particular, we get that the subgraph G' of G_{s_0} induced by the set of vertices $\{u_{s_0} \in V_{s_0} : (u_{s_0}, 1) \in \texttt{groups}\}$ is connected, so the robot can paint each of the 1-groups without running into a dead state. Thus, the algorithm devises π such that the robot first paints each of the 1 groups. Next, the robot moves to $f(s)$ and starts painting the group in the following way. At iteration $i \in \{1, \dots, |V| - 1\}$, if $\deg_G(p_{i+1}) = 2$, the robot moves from $f(p_i)$ to $f(p_{i+1})$ in a straight line, thus painting four squares. And if $\deg_G(p_{i+1}) = 3$, the robot moves from $f(p_i)$ to $f(p_{i+1})$ by doing a turn as shown in Figure 6.5. This clearly produces a plan since in total, the number of cells that the robot paints is

$$\begin{aligned} 1 + \sum_{i=1}^{|V|-1} 4 + 2 \cdot \mathbf{1}_{\deg_G(p_{i+1})=3} &= 4 \cdot (|V| - 1) + 2 \cdot D_G^3(V) + 1 \\ &= 4 \cdot |V| + 2 \cdot D_G^3(V) - 3 \end{aligned}$$

as desired. □



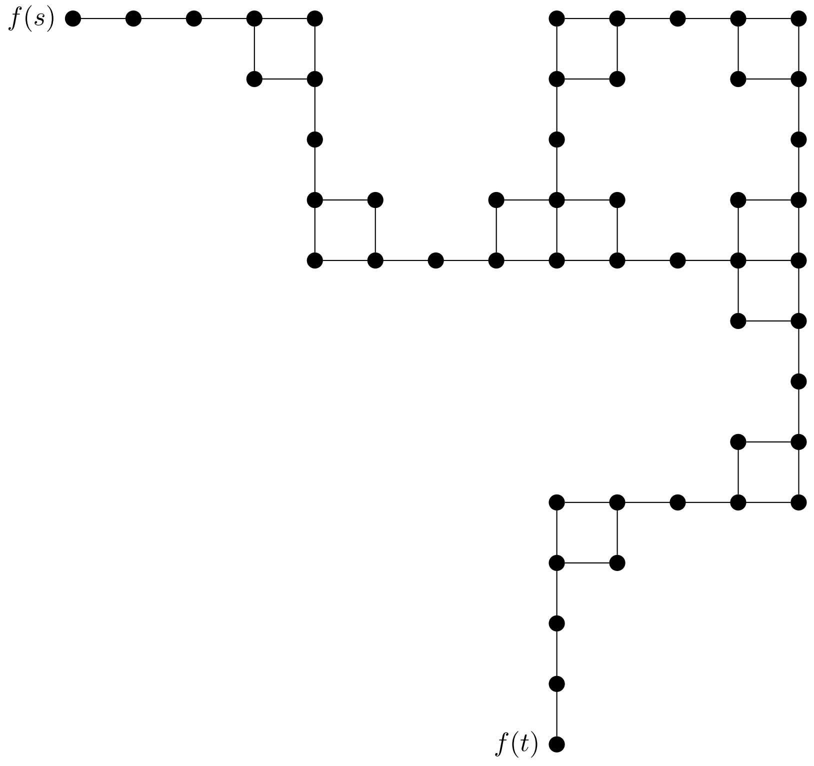


Figure 6.4: The T and T'-graphs of s_0, where s_0 is the `IPC Nurikabe` state from Figure 6.3.

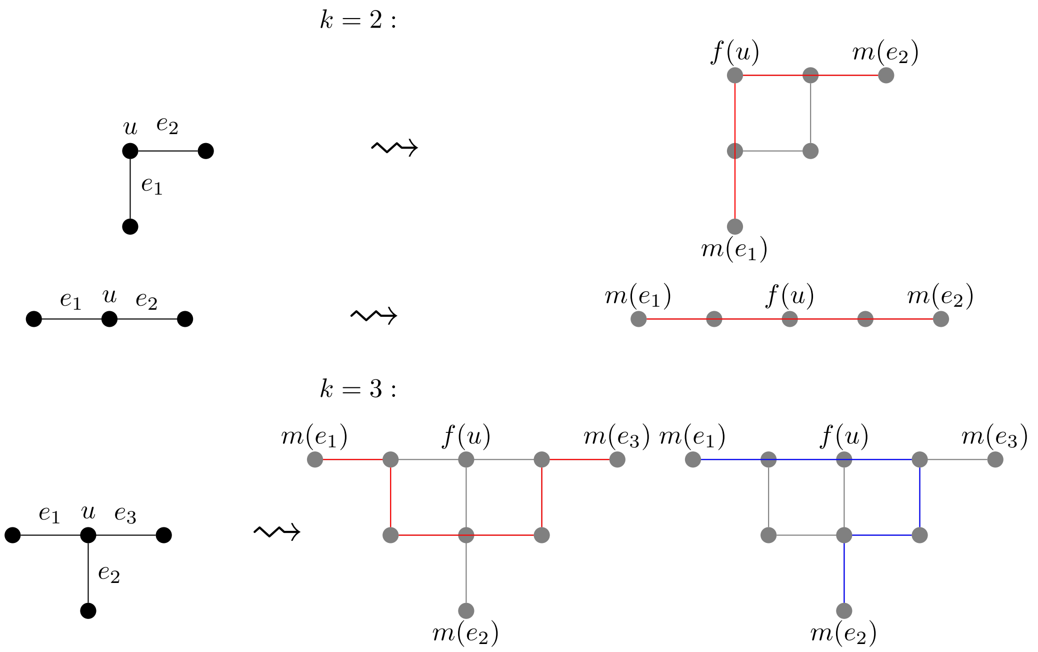


Figure 6.5: We trace possible longest simple paths from each bridge midpoint to another in $G_{z(u)}$ up to symmetries in rotations and mirroring where $k = \deg_G(u)$. In the case $k = 2$, possible longest simple paths from one of $m(e_1)$ to one of $m(e_2)$ are traced in red on the right-hand side. In the case of $k = 3$, a path from $m(e_1)$ to $m(e_3)$ and vice-versa is traced in red. Similarly, a path from $m(e_2)$ to $m(e_3)$ and vice-versa is traced in blue. Note that a path from $m(e_2)$ to $m(e_3)$ can be obtained considering the blue path mirrored along the y axis.

7 Other Domains

We do a literature review of other classical planning domains based on grids that appear in previous IPCs.

7.1 Snake

Snake is the name of a 2D single-player video game where the player controls the movement of the head of a simple path (a snake) along a grid. Apples spawn in arbitrary positions of the grid, and when the head of the path intersects an apple (i.e. when the snake eats an apple), the simple path grows in length. The goal is to eat as many apples as possible without colliding with obstacles or the snake's body.

The game of Snake has many variants. In the most well-known variant, which we call the classical variant, the grid on which the game is played is rectangular, there is a single apple in the initial state, and a new apple spawns in a random cell each time an apple is eaten. In the variant called Nibbler, there can be an arbitrary number of apples in the initial state, and new apples do not spawn once other apples have been eaten. The IPC version of Snake (which we call `IPC Snake` throughout the rest of this text) is similar to that of the classical version, the only difference is that this variant removes any uncertainty by predeterminedly selecting the spots where the apples will spawn.

The plan existence problem for Nibbler is **NP**-hard if played on solid grid graphs [3]. This has been shown by reducing from `GHP`. With similar arguments, the authors show that Nibbler is **NP**-hard if played on rectangular grids, assuming that the snake's growth rate after eating each apple is at least two, and assuming that the snake's body can be of arbitrary size in the initial state. Using the nondeterministic constraint logic framework, the authors also show that Nibbler is **PSPACE**-complete if played on arbitrary grid graphs. A key observation in these reductions is that they are made such that in any plan, some apples must be eaten in a certain order. This forces the snake to move in a way that it avoids certain cells. Imposing an ordering on when the apples are eaten gives rise to complexity in a way that cannot be done in `IPC Snake` since in that variant there is always at most one apple in the grid at any given reachable state.

We consider the following decision problem posed by Du et. al. [8].

Partial Path Problem
INPUT: A rectangular graph G and a simple path P on G.
QUESTION: Is there a Hamiltonian cycle on G that contains P as a subpath?

It is conjectured that the partial path problem is **NP**-hard, and that there exists a reduction from this problem to the plan existence problem `IPC Snake`[3] [8]. However, an outline for both the hardness proof and the reduction are missing.

7.2 Ricochet Robots

Ricochet Robots is the name of a multiplayer sliding game puzzle published by Rio Grande Games[4]. In the game, there is a rectangular grid G of fixed size, 4 robots of colored red, green blue, and yellow placed in vertices of G, and a set of "walls", represented as a set of edges in G. An instance of the game is solved once a sequence of robot movements has been found such that a predetermined robot reaches a predetermined square. The movement of the robots is limited: they may only move one at a time in one of the four cardinal directions, and they slide once they start moving, i.e. they can only stop once they are in front of another robot, or when they collide with a wall, or once they reach the boundary of the grid. An instance of Ricochet Robots is depicted in Figure 7.1. Ricochet Robots is played in multiple rounds between two or more players. At the start of each round, a new instance is shown to the players. Players who find short solutions (i.e. solutions with a low number of robot movements) gain tokens. After all rounds are over, the player with the most amount of tokens wins the game.
We note that the number of players does not affect the complexity of the game. There is a straightforward single-player variant of the game, where the goal is simply to either determine whether an instance is solvable (plan existence problem) or whether an instance is solvable in at most K steps (bounded plan existence problem). Throughout the remainder of this text, we treat Ricochet Robots as a single-player game.
Ricochet Robots has been solved via the means of brute force computation [6]. This is only possible due to the fixed size of the grid. The game needs to be generalized to make it interesting from a computational complexity point of view, there are several ways of doing this. We mention the work by Masseport et. al. [21], which generalizes the puzzle in several directions, as follows.

Definition 40 (Generalized Ricochet Robots domain). *The domain $\mathit{GRR}(n, c_r, m, c_t)$ is comprised of all Ricochet Robot instances on rectangular grids of arbitrary size with n robots $\{r_1, \dots, r_n\}$, where each robot r_i is chosen from a list of c_r total colors, and m target tiles $\{t_1, \dots, t_m\}$, each of which is chosen from a list of a total of c_t colors. An instance of this domain is solved once a state s' is reached such that each robot r_i is on a target tile t_j satisfying* $\mathbf{robot\_color}(r_i) = \mathbf{tile\_color}(t_j)$.

In the IPC variant of Ricochet Robots, we have $n = 4, c_r = 4, m = 1, c_t = 4$. To the best of

[3] The authors do not make it clear whether they mean `IPC Snake` or the classical variant of Snake.
[4] The game seems to be discontinued, as it is no longer listed on Rio Grande Games's website.

our knowledge, the complexity of this problem has yet to be analyzed. The following results are known. The plan existence problem for GRR$(n, 1, 1, 1)$ is fixed-parameter tractable for fixed values of n [15] and **PSPACE**-complete for arbitrary values of n [21].

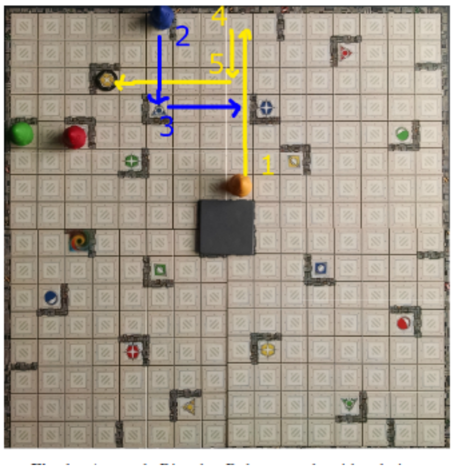


Figure 7.1: [15]. An optimal solution is drawn on top of the grid.

7.3 Labyrinth

Labyrinth is a board game published by Ravensburger [18]. The game is comprised of a set of labeled square cards, each with one of three shapes, shown in Figure 7.2 and four-player tokens. At the start of the game, all but one of the cards are randomly placed such that they form a rectangular grid, thus, forming a labyrinth. We call the leftover card the *turnover card*. Each card is placed in a random orientation. Moreover, the token of each player is placed in a corner of the card grid, and a list of random labels is assigned to each player. Labyrinth is played in turns, and the first player to visit all of the cards with their respective labels in order wins the game. In each turn, each player performs two actions. First, they change the shape of the labyrinth by choosing one column or row, and one direction (north/south or east/west respectively), each card in that column or row then slides by one unit in the direction indicated by the player. The turnover card fills in the missing spot and the card that slides out of the grid becomes the new turnover card. As for the second action, the turn player can then move his token to any card in the grid to which a path exists. An example of a turn in Labyrinth is shown in Figure 7.3.
The IPC variant of Labyrinth is inspired by this game, yet it differs in three ways. First, there is no turnover card. Instead, after a shift, the card that leaves the grid fills in the missing spot. Second, the IPC variant is a single-player puzzle, where the goal is for the player to maneuver their token from the top left corner to the bottom right corner. Finally, a shift cannot be performed on the column or row that contains the token.
The computational complexity of a Labyrinth variant, which we call `Labrynth`* has been analyzed [25]. It is a single-agent version of the original Labyrinth game, where the goal is to maneuver the token from one given card to another. The bounded plan existence

of `Labrynth`* is **NP**-hard. The reduction is based on `3CNF-SAT`[5]. Despite there being a turnover card, the corresponding instance is always defined in such a way that the movement of the agent never visits a newly added card, making the behavior of the maze similar to that of the IPC variant. Moreover, the instances are always such that the token must move from the top left corner to the bottom left corner of the grid.

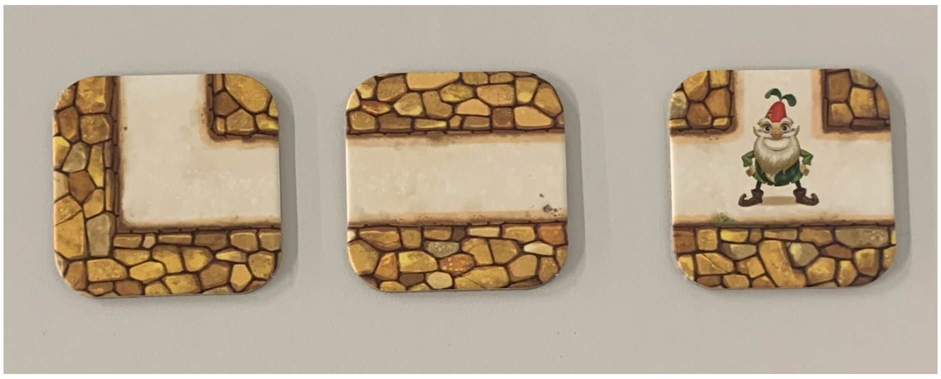


Figure 7.2: The three types of tiles in Labyrinth.

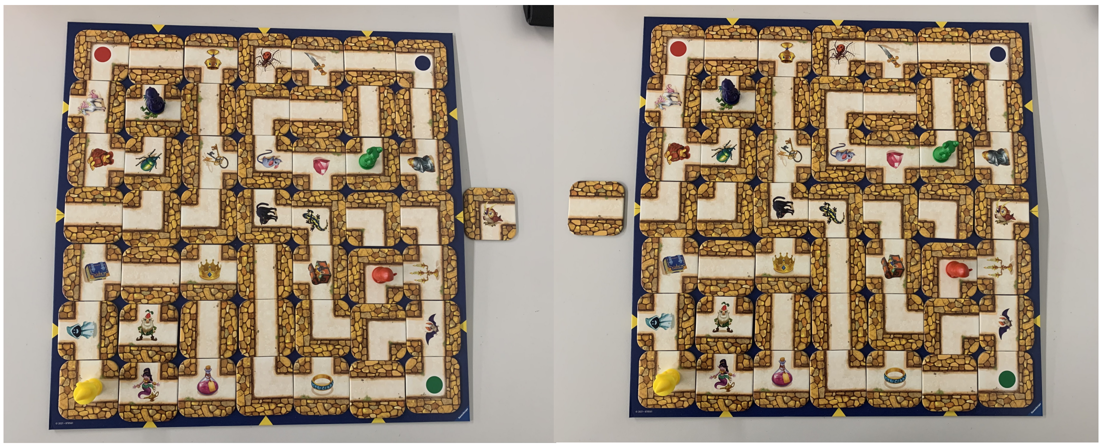


Figure 7.3: A full turn in Labyrinth. The turn player shifts the middle row to the left and decides not to move his token.

[5] This problem was introduced in Section 5.2.

8 Conclusion

We studied the computational complexity of five classical planning domains based on grids stemming from past IPCs. Table 8.1 summarizes our results. We also did a literature review on three other domains. The IPC domains Nurikabe, TERMES, Snake, Ricochet Robots, and Labyrinth are inspired by existing puzzles/domains. However, with the exception of maybe Labyrinth, from a computational complexity point of view, they are different enough from their counterparts to the point where they are hardly comparable.

A special case of the work by Itai et. al. [17] shows that on any rectangular grid of order at least three, there is a Hamiltonian path starting at any vertex. We showed that this implies that the bounded plan existence problem for VisitAll is solvable in polynomial time.

There are bounded plan existence instances of VisitAll that can be solved only if the robot's movement matches some Hamiltonian path on a grid graph. When the goal is to visit a preselected set of cells, we used this fact to make a reduction from $\texttt{DEG-4-GHP}_1$, i.e. the s-t Hamiltonian path problem on grid graphs where t has degree 1, which we have shown to be intractable.

Floortile is the only domain in this thesis that features non-uniform action costs. We provided necessary and sufficient conditions for the plan existence problem which can be computed in polynomial time. We also provided an algorithm that, given a solvable instance of Floortile, provides a plan of polynomial length. The bounded plan existence problem is open. A source of difficulty here is that, due to the action costs, a reduction from a Hamiltonian path/cycle problem is challenging.

We divided (the IPC variant of) TERMES into two sub-domains. One where the state s_0 is empty, and one where s_0 is allowed to contain blocks. The complexity of the plan bounded existence and bounded plan existence for the former are open, however, we conjecture that the bounded plan existence problem is **NP**-hard. As for the latter, we showed that the plan existence and bounded plan existence problems are **NP**-hard by reducing from DEG-4-GHP. In our hardness reduction, we placed tall, unreachable stacks of blocks in the grid, essentially adding obstacles, and as a goal, we set the agent to build a tall stack, which must be built through a ramp that follows a simple path in a grid graph.

The PDDL definition of the Tidybot domain allows for multiple robots or objects to be on the same cell. Furthermore, it defines the gripper's movement in such a way that it behaves strangely when the gripper's radius is more than one. We thus modeled this domain by adding further constraints that enforce that at most one object/robot can be on a single cell at all times. We also redefined the movement of grippers when the radius is more than one. Under these new constraints, we showed that Tidybot has the game Push -1F as a special case, and is thus **PSPACE**-complete. In settings with exactly one cart, one robot, and no obstacles, we came up with a polynomial time procedure that decides whether an instance is solvable. However, finding optimal plans in this setting is **NP**-complete.

Nurikabe is a Japanese logic puzzle. In the IPC variant of Nurikabe, there is a robot that moves through a square grid and paints cells with white. There is a straightforward argument that shows that solvable instances of this variant have polynomial length plans. Moreover, an easy reduction from $\texttt{DEG-4-GHP}_1$ shows that the bounded plan existence problem is **NP**-hard. The plan existence problem is **NP**-complete. We reduced from a new Hamiltonian path problem on subgrid graphs, which we have shown to be **NP**-complete. In our reduction, the agent must paint a group by following a simple path, which is forced to correspond to a simple path in a subgrid graph due to the presence of 1-groups scattered along the grid.

Some of the reductions in this thesis featured new decision problems, which we have shown to be intractable (see Lemma 17 and Lemma 2). Since IPCs serve as a testing suite for domain-independent classical planning algorithms, the results from Table 8.1 can be used by researchers and practitioners to assess new algorithms and heuristics in these areas. Moreover, the hardness proofs in this thesis highlight hard instances of these domains. We categorized various domains into complexity classes, thus expanding the list of **P**, **NP**, **NP**-complete, **PSPACE**, and **PSPACE**-complete problems, which allows researchers to use them for hardness reductions in the future.

8.1 Future Work

The computational complexity of the three IPC domains in the literature review has yet to be studied, namely Snake, Ricochet Robots, and Labyrinth. All three of them are inspired by existing domains whose complexity has been partially explored. There are more domains based on grids featured in previous IPCs that have not been covered in this thesis, namely Slitherlink, Protein Folding, and Spanner. In this thesis, we worked with the complexity classes **P**, **NP**, and **PSPACE**. Future work could also include a more granular classification of complexity classes (e.g. **L**, **NL**) to the domains studied in this thesis.

We present a conjecture that we found during the thesis, but were unable to prove.

8.1.1 TERMES

Bob is a mechanical engineer and wants to put his new invention: a TERMES robot, to use. He wants to coordinate his robot to build structures as efficiently as possible on an empty field. Unfortunately for him, this task is likely to be intractable.

Conjecture 1. `BPE-TERMES-ES` is **NP**-hard.

Proof idea. Let (G, s, t) be an instance of of `DEG-3-GHP` with $G = (V, E)$. Further, Let

	PlanEx	**Bounded PlanEx**
VisitAll	**P**	**P**[a] **NP**-complete[b]
TERMES	**NP**-hard[c] ?[d]	**NP**-hard[c] ?[d]
Tidybot	**P**[e] **PSPACE**-complete[f]	**NP**-complete[e] **PSPACE**-complete[f]
Floortile	**P**	?
Nurikabe	**NP**-complete	**NP**-complete

[a] If the goal is to visit all the cells in the grid.
[b] In the general case, where the goal can be any subset of cells.
[c] If the initial state is allowed to contain blocks.
[d] If the initial state is empty.
[e] If there is only one cart and no obstacles.
[f] In the general case.

Table 8.1: Overview of the complexity classes of each domain studied in this thesis. Note that in each entry, we always mean the IPC variant of the corresponding domain. In Tidybot we add the constraint that no two robots/objects can be in the same cell, and we also redefined the behavior of the gripper when its radius is more than one.

$(s'_0, \texttt{goal}', K')$ be the BPE-TERMES-NES instance defined in the proof of Lemma 11. We define an instance $I = (s_0, \texttt{goal}, K)$ of BPE-TERMES-ES as follows. We let s_0 be like s'_0, except that $\texttt{num-blocks}_{s_0}(u) = 0$ for all u. Next, we let $\texttt{goal} = \texttt{goal}'$ and

$$K = K' + 4 \cdot |V| - 4 + 12 \sum_{u \in V \setminus \{s,t\}} d_{\ell_1}(u, s)$$

$$= 7 \cdot |V| - 9 + 12 \sum_{u \in V \setminus \{s,t\}} d_{\ell_1}(u, s)$$

If G has an s-t Hamiltonian path, then $(s_0, \texttt{goal})$ can be solved in K steps. To justify this, it is easy to see that s'_0 is reachable from s_0 using a total of $K - K'$ actions. The fact that we can reach a goal state from s'_0 in K' steps has been shown in Lemma 11. For the other implication, we conjecture that if I is solvable, then there exists an optimal plan for I that starts with π, where π is a sequence of actions of length $K - K'$ such that $s_0[\pi] = s'_0$. $\square$

Bibliography

[1] Malte Helmert. malte.helmert@unibas.ch.

[2] Joshua Ani, Lily Chung, Erik D Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch. Pushing blocks via checkable gadgets: PSPACE-completeness of Push-1F and Block/Box Dude. In *FUN 2022*, pages 3:1–3:30.

[3] Marzio De Biasi and Tim Ophelders. The complexity of Snake. In *FUN 2016*, pages 11:1–11:13.

[4] Michael Buro. Simple Amazons endgames and their connection to Hamilton circuits in cubic subgrid graphs. In *Computers and Games 2000*, pages 250–261.

[5] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[6] Yibo Chen, Wee-Chong Oon, and Wenbin Zhu. Ricochet Robots is solved. *J. Int. Comput. Games Assoc.*, 34(4):223–226, 2011.

[7] Massimiliano de Leoni and Andrea Marrella. Aligning real process executions and prescriptive process models through automated planning. *Expert Syst. Appl.*, 82:162–183, 2017.

[8] Kevin Du, Ian Gemp, Yi Wu, and Yingying Wu. Alphasnake: Policy iteration on a nondeterministic NP-hard Markov decision process. *CoRR*, abs/2211.09622, 2022.

[9] David Eppstein and Emil Jeřábek. Name and complexity of a stone placement puzzle. Theoretical Computer Science Stack Exchange. URL https://cstheory.stackexchange.com/q/53215. Last accessed 18 Nov. 2023.

[10] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Comput. Surv.*, 18(1):23–38, 1986.

[11] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[12] Patrik Haslum. Reducing accidental complexity in planning problems. In *IJCAI 2007*, pages 1898–1903.

[13] Malte Helmert. Complexity results for standard benchmark domains in planning. *Artif. Intell.*, 143(2):219–262, 2003.

[14] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006.

[15] Adam Hesterberg and Justin Kopinsky. The parameterized complexity of Ricochet Robots. *J. Inf. Process*, 25:716–723, 2017.

[16] Markus Holzer, Andreas Klein, and Martin Kutrib. On the NP-completeness of the Nurikabe pencil puzzle and variants thereof. In *FUN 2004*, pages 77–89.

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

[18] Max J Kobbert. *Das verrückte Labyrinth.* Ravensburger Spieleverlag, 1986.

[19] Sven Koenig and S Kumar. A case for collaborative construction as testbed for cooperative multi-agent planning. In *ICAPS 2017*.

[20] Arman Masoumi, Megan Antoniazzi, and Mikhail Soutchanski. Modeling organic chemistry and planning organic synthesis. In *GCAI 2015*, pages 176–195, 2015.

[21] Samuel Masseport, Benoit Darties, Rodolphe Giroudeau, and Jorick Lartigau. Ricochet Robots game: complexity analysis. Technical Report, 2019. hal-02191102.

[22] Christos H. Papadimitriou and Umesh V. Vazirani. On two geometric problems related to the Traveling Salesman Problem. *J. Algorithms*, 5(2):231–246, 1984.

[23] Gerald Paul, Gabriele Röger, Thomas Keller, and Malte Helmert. Optimal solutions to large logistics planning domain problems. In *SOCS 2017*, pages 73–81.

[24] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[25] Florian D. Schwahn and Clemens Thielen. The complexity of escaping labyrinths and enchanted forests. In *FUN 2018*, pages 30:1–30:13.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Travis Rivera Petit

Matriculation number — Matrikelnummer
2015-117-427

Title of work — Titel der Arbeit
Computational Complexity of Classical Planning Domains Based on Grids

Type of work — Typ der Arbeit
Master's thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, December 12, 2023

Signature — Unterschrift