Complexity results for standard benchmark domains in planning

Malte Helmert

Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Georges-Köhler-Allee, Gebäude 052, 79110 Freiburg, Germany

Abstract

The efficiency of AI planning systems is usually evaluated empirically. For the validity of conclusions drawn from such empirical data, the problem set used for evaluation is of critical importance. In planning, this problem set usually, or at least often, consists of tasks from the various planning domains used in the first two international planning competitions, hosted at the 1998 and 2000 AIPS conferences. It is thus surprising that comparatively little is known about the properties of these benchmark domains, with the exception of BLOCKSWORLD, which has been studied extensively by several research groups.

In this contribution, we try to remedy this fact by providing a map of the computational complexity of non-optimal and optimal planning for the set of domains used in the competitions. We identify a common *transportation* theme shared by the majority of the benchmarks and use this observation to define and analyze a general transportation problem that generalizes planning in several classical domains such as LOGISTICS, MYSTERY and GRIPPER. We then apply the results of that analysis to the actual transportation domains from the competitions. We next examine the remaining benchmarks, which do not exhibit a strong transportation feature, namely SCHEDULE and FREECELL.

Relating the results of our analysis to empirical work on the behavior of the recently very successful **FF** planning system, we observe that our theoretical results coincide well with data obtained from empirical investigations.

Key words: complexity of planning, planning benchmarks, planning domains, transportation problems

Email address: helmert@informatik.uni-freiburg.de (Malte Helmert).

1 Introduction

Empirical methods have become the standard for performance evaluation in AI planning. Running time on tasks from classical planning domains such as LOGISTICS and BLOCKSWORLD has often been (and still is) used for comparing the relative merits of planning systems, or, put a bit more provocatively, to draw the line between good and bad ones. However, this kind of comparison has its difficulties. If no planning system performs well in a given domain, does that mean that they are all poor, or is that domain intrinsically hard? If they all perform well, is this because of their strength or because of the simplicity of the task?

Because of these issues it is rather surprising that so far, comparatively little research has been conducted on the complexity of planning in the classical domains. In contrast, *domain-independent* complexity results for STRIPS planning and related formalisms are quite well-known. Different variants of the planning problem have been investigated by several researchers [1–3]. However, they focus on special cases of planning defined by purely *syntactical* features such as the number of operator preconditions or effects, while the actual representation of a planning domain is of no concern to our work.

The complexity of planning has also been studied in the case where some planning domain is fixed in advance (which is essentially domain-dependent planning, but for an *unspecified* domain) [2]. The results show that planning in a domain that can be represented in the STRIPS subset of PDDL can be **PSPACE**-complete, but no worse. This provides an upper bound for the complexity of all planning domains.

However, there is one classical planning domain, BLOCKSWORLD, which has been studied extensively by several researchers. The existing analyses cover the domain in its standard form as well as extensions, such as blocks of different size. A classical reference for this line of research is [4]. Other references emphasize the important distinction between optimal and non-optimal planning in general and in the BLOCKSWORLD domain in particular [5,6], where the latter article contains a deep analysis of the algorithmics and empiricism of BLOCKSWORLD planning.

Theoretical knowledge of the complexity of planning in a particular domain is not only useful for judging the runtime behavior of planning systems. It also helps in addressing the question whether planning systems that plan quickly or planning systems that generate short plans should be preferred. Of course it should be pointed out that it is not possible to provide a general answer here: in some application contexts high quality plans might be critically important, whereas in other cases, being able to plan (and act) quickly is preferable.

Year	Domain name	Year	Domain name
1998	ASSEMBLY	2000	BLOCKSWORLD
	GRID		FREECELL
	GRIPPER		LOGISTICS
	LOGISTICS		MICONIC-10
	MOVIE		SCHEDULE
	MYSTERY		
	MYSTERY'		

Fig. 1. Domains from the AIPS 1998 and AIPS 2000 planning competitions.

But still, theoretical results can contribute to this discussion in several ways. On the one hand, in domains where generating optimal plans is a problem that can be solved in polynomial time, there are fewer reasons to be content with non-optimal plans and generating overly long plans can be viewed as a deficiency of a planning algorithm. On the other hand, in domains where there is a difference in computational complexity between non-optimal and optimal planning, striving for optimal plans clearly demands a price in runtime performance, which the user of a planning system might not always be willing to pay.

Another point in favor of theoretical analyses is their potential to expose *sources* of hardness in planning domains. For instance, if we discovered that in a hypothetical PAC-MAN domain, plans can be generated in polynomial time if there is a single ghost, while the corresponding problem with multiple ghosts is **NP**-hard, then this would allow us to draw the conclusion that one source of hardness in this domain is the number of ghosts.

Seeing that domain-specific complexity results for planning problems appear to be useful, the question arises *which* domains should be analyzed. It is evident that it is impossible to investigate the complexity of every single planning domain imaginable. So we try to select a set of domains which are especially important in AI planning, ones that are considered *standard benchmarks*.

The domains from the AIPS planning competitions (Figure 1) certainly satisfy this criterion and hence form the core of our domain catalogue. The competition domains are of interest because they are well-known and because there is a high amount of empirical data available for them in the form of the competition results. This makes it possible to compare empirical performance and theoretical complexity, and hence identify shortcomings of current planning systems: If we can prove polynomial complexity results in a given domain, but observe poor scaling behavior of a system, this might tell us what kind of

issues should be addressed in order to improve overall performance. The competition featured a few domains which might be called atypical for planning (like FreeCell), but nevertheless we will try to cover all of them to provide a more complete picture. We will, however, not be able to make a decisive statement regarding the complexity of the Assembly domain from the AIPS 1998 competition, because the domain definition has several flaws which make the intended semantics unclear. We will come back to this point in Section 4.1.

Of course it would be possible to look at each competition domain in isolation. However, we will see that there are strong commonalities between some of them, which can be exploited to get a clearer picture of the sources of hardness in those benchmarks. In this case, it makes more sense to investigate a *family* of related planning domains, including competition domains as well as natural generalizations. This motivates the definition of the Transport domain family in Section 3.

We proceed as follows. In the following section, we will define the formal framework of our analysis and address the (non-trivial) question what we understand by planning in a given domain.

In Section 3 we investigate transportation planning by introducing the Transport domain family, analyzing its complexity and applying the results to the competition domains Gripper, Logistics, Mystery and Mystery'. The other two competition benchmarks with a transportation theme, Grid and Miconic-10, are covered in greater detail at the end of that section.

The remaining competition domains, which are not naturally subsumed under "transportation planning", are treated in Section 4. Movie, Assembly and Blocksworld are only discussed briefly for different reasons, but Schedule and FreeCell are discussed in depth.

We summarize and discuss our results in Section 5, which also hints at possible directions for future research.

2 Planning domains

To get started, we will formalize the concept of planning in a given domain. For each domain, which can be understood as an infinite set of planning tasks, we are interested in two decision problems:

- PlanEx-Domain: Given a planning task from Domain, does the task have a solution, i. e. an operator sequence transforming the initial situation into

one that meets the goal requirements?
- PLANLEN-DOMAIN: Given a planning task from DOMAIN and an integer K, does the task have a solution of length at most K?

2.1 Formal definitions

These informal statements show that solving a planning task corresponds to finding a path in some state space, leading from a specified initial state to some goal state. The following definition captures the semantics of a planning task.

Definition 1 *Planning task state model*

*A **planning task state model** is a four-tuple $\mathcal{M} = (S, s_0, S_G, A)$, where*

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

*$\mathcal{M}$ defines the **transition graph** $\mathcal{T}(\mathcal{M})$, a labeled digraph with vertex set S and an arc labeled a from s to $a(s)$ for all actions a and all states s in the domain of a.*

We can now define what we mean by a planning domain.

Definition 2 *Planning domain*

*A **planning domain** $\mathcal{D}$ is a function that maps words over some encoding language to planning task state models. A word T that is part of the domain (in the usual mathematical sense) of $\mathcal{D}$ is called a **planning task** of $\mathcal{D}$. We write $||T||$ to denote its length.*

In practice, the encoding language for planning tasks is usually PDDL. However, since we are not interested in representational issues here, we will describe planning tasks in terms of structures that are natural to the domain at hand (such as roadmap graphs or fuel functions) rather than encode them in propositional logic. For our results to be applicable to planning in PDDL, we must make sure that encoding lengths of planning tasks in this two different representation are polynomially equivalent. Fortunately, this is the case for all the domains we will investigate, and we will not explicitly mention encoding lengths in the following definitions of planning domains. This point is discussed in more detail in the reference [7].

Having defined planning domains, we can now formally define the decision problems introduced before.

Definition 3 PLANEX-$\mathcal{D}$ *decision problem*

Let $\mathcal{D}$ be a planning domain. Then the PLANEX-$\mathcal{D}$ *decision problem is defined as follows:*

Given a planning task T from $\mathcal{D}$, with $\mathcal{D}(T) = \mathcal{M} = (S, s_0, S_G, A)$, is there a path in $\mathcal{T}(\mathcal{M})$ leading from s_0 to some state from S_G?

We call PLANEX-$\mathcal{D}$ the *plan existence* problem for domain $\mathcal{D}$. It is the decision problem counterpart of *plan generation*. Informally, we ask if there is a sequence of actions that transforms the initial situation of a given planning task into a situation where the goal requirements are met, i. e. we ask about the existence of a *solution* to the planning task. Such a sequence of actions will also be called a *plan* in the following.

Definition 4 PLANLEN-$\mathcal{D}$ *decision problem*

Let $\mathcal{D}$ be a planning domain. Then the PLANLEN-$\mathcal{D}$ *decision problem is defined as follows:*

Given a planning task T from $\mathcal{D}$, with $\mathcal{D}(T) = \mathcal{M} = (S, s_0, S_G, A)$, and an integer K, is there a path of length at most K in $\mathcal{T}(\mathcal{M})$ leading from s_0 to some state from S_G?

We call PLANLEN-$\mathcal{D}$ the *bounded plan existence* problem for domain $\mathcal{D}$. It is the decision problem counterpart of *optimal plan generation* in the same sense that the Traveling Salesman decision problem is the counterpart of the Traveling Salesman optimization problem. Informally, we ask about the existence of a *short solution* to the planning task.

Most of the results we will present in the following are *polynomial-time reducibility* results for different versions of the plan existence and bounded plan existence decision problems. Throughout the article, we use the notation PROBLEM1 $\leq_p$ PROBLEM2 to denote that there is a polynomial-time reduction from PROBLEM1 to PROBLEM2. We complement our definitions with two simple reducibility results of this kind.

Theorem 5 *Plan existence vs. bounded plan existence*

Let $\mathcal{D}$ be a planning domain. If there is a function $f : \mathbb{N} \to \mathbb{N}$ which can be computed in polynomial time such that for all tasks $T \in \mathcal{D}$ that have a solution, the length of the shortest solution can be bounded by $f(||T||)$, then PLANEX-$\mathcal{D}$ $\leq_p$ PLANLEN-$\mathcal{D}$.

PROOF. *The function mapping T to $(T, f(||T||))$ is a polynomial reduction.* □

The conditions of the previous theorem are satisfied for all planning domains in this paper, and indeed for all planning domains that can be concisely encoded in propositional PDDL, using the function $f : n \mapsto 2^n$. We will thus in the following apply this theorem without giving an explicit reference. The same is true for the following straight-forward result.

Theorem 6 *Generalization/Specialization*

Let $\mathcal{D}$ and $\mathcal{D}'$ be planning domains such that all tasks of $\mathcal{D}$ are tasks of $\mathcal{D}'$, and for all such tasks T, $\mathcal{D}(T) = \mathcal{D}'(T)$. Then PLANEX-$\mathcal{D} \leq_p$ PLANEX-$\mathcal{D}'$ *and* PLANLEN-$\mathcal{D} \leq_p$ PLANLEN-$\mathcal{D}'$.

PROOF. *The reductions are simply by embedding, i. e. T is mapped to T and (T, K) is mapped to (T, K).* □

2.2 What is a planning domain?

Having formalized planning domains, the question arises how the planning domains from the competitions fit into this framework. Which tasks should be considered part of a given domain? An obvious idea is to make use of the available PDDL [8] domain definitions to identify valid tasks. Only tasks that can be defined by coupling the (prespecified) PDDL domain file with a PDDL problem file can be tasks of the domain.

We consider this criterion necessary, but not sufficient for deciding whether a given task is part of a specific domain. In many domains, important pieces of information are not (and cannot be made) explicit in the PDDL domain file. For example, the fact that a block cannot sit on top of itself is an important property of BLOCKSWORLD tasks, yet the domain file does not require or imply this property. Similarly, in domains with a transportation theme like LOGISTICS, it is usually assumed but not made explicit that locations, carriers and portable objects are disjoint classes, and hence carriers cannot pick up other carriers or move locations around.

So there must be a second, restricting criterion to narrow the choice of tasks. Unfortunately, for most domains this is not formalized in the literature, so we must identify the "intended" set of tasks in a given domain ourselves. We are guided in this process by the domain descriptions that are available in the literature and by the set of tasks that have been used as benchmarks to date.

The latter is of special relevance if our results are to be employed for judging the performance of planning systems on the existing benchmark suites. As an example, consider the GRIPPER domain, where a robot moves objects between different rooms. Although the PDDL definition allows for more general specifications, in all GRIPPER tasks which were used for benchmarking, there are only two rooms, all objects are initially located in the same room as the robot, and all objects need to be moved to the other room. What is more, the term "GRIPPER task" is generally used only in this restricted sense. For this reason, it makes more sense for our formal definition of the domain to mirror these implicit constraints and *not* exploit the full potential of the PDDL specification of the domain.

Fortunately, we do not have to resort to a lot of guesswork to capture the original intentions of the domain designers: We will be able to refer to the descriptions of McDermott [8] and Bacchus [9], where they are not ambiguous.

3 Transportation domains

We begin this section by quoting the descriptions of some of the AIPS 1998 planning domains from an article by the competition organizer, Drew McDermott [8]:

> GRID: There is a square grid of locations. A robot can move one grid square at a time horizontally and vertically. If a square is locked, the robot can move to it only by unlocking it, which requires having a key of the same shape as the lock. The keys must be fetched and can themselves be in locked locations. Only one object can be carried at a time. The goal is to get objects from various locations to various new locations.
>
> GRIPPER: Here, a robot must move a set of balls from one room to another, being able to grip two balls at a time, one in each gripper. There are three actions: (1) move, (2) pick, and (3) drop.
>
> LOGISTICS: There are several cities, each containing several locations, some of which are airports. There are also trucks, which can drive within a single city, and airplanes, which can fly between airports. The goal is to get some packages from various locations to various new locations.

The following description by Drew McDermott was taken from the web resource of the AIPS 1998 competition:

> MYSTERY: There is a planar graph of nodes. At each node are vehicles, cargo items, and some amount of fuel. Objects can be loaded onto vehicles

(up to their capacity), and the vehicles can move between nodes; but a vehicle can leave a node only if there is a nonzero amount of fuel there, and the amount decreases by one unit. The goal is to get cargo items from various nodes to various new nodes.

And the last domain from the 1998 competition that we want to address in this section, again quoted from the article by McDermott [8]:

> MYSTERY': This is the MYSTERY domain with one extra action, the ability to squirt a unit of fuel from any node to any other node, provided the originating node has at least two units.

Finally, here is a brief description of the MICONIC-10 domain from the AIPS 2000 competition, discussed in more detail in Section 3.7:

> MICONIC-10: There is an elevator moving between the floors of a building. There are passengers waiting at various floors. The goal is to move each passenger to their destination floor. There are three variants of this domain, including one where special constraints such as VIP service restrict elevator movement.

It is evident that there are some commonalities between those planning domains. Specifically, they all share the following properties (terminology is borrowed from Long and Fox [10]).

- There is a set of *locations* (grid squares, rooms, airports, ...), which are connected by *roads* (adjacent grid squares, doors, airways, ...), forming a *roadmap* graph.
- There is a set of *mobiles* (robots, trucks, elevators, ...), which traverse the roadmap.
- There is a set of *portables* (keys, balls, passengers, ...), which can either be at a location or carried by a mobile.
- These classes of entities are disjoint, and there are no other entities.
- The goal is to move (a subset of) the portables to their respective *final destinations*.

Of course, there is also a number of differences, including the following:

- *Capacity constraints*: In GRID, mobiles can only carry one portable at a time, in GRIPPER two portables. In MYSTERY and MYSTERY', mobiles have varying capacities, and in LOGISTICS and MICONIC-10, capacity is unbounded.
- *Fuel constraints*: In MYSTERY and MYSTERY', fuel is consumed by and required for movement, unlike the other domains.
- *Number of mobiles*: In GRIPPER, GRID and MICONIC-10, there is only a single mobile. In the other domains, there can be several.

- *Type of roadmaps*: In MYSTERY and MYSTERY', the roadmap can be any planar graph, in GRID it must be a grid, in other domains a complete graph. In LOGISTICS, there is the additional restriction that different mobiles can use different roads: Roads within cities are only used by trucks, and airways only by airplanes.
- *Dynamic roadmaps*: In GRID, new links between locations can be established by opening doors. In the most complex variant of MICONIC-10, the set of accessible floors varies when passengers board or debark. In the other domains, dynamic changes of this kind do not occur.

3.1 The TRANSPORT *domain family*

In the rest of this section, we will define and analyze a hierarchy of planning domains combining the key features of the transportation planning benchmarks mentioned. Different members of that family model different constraints on capacity, fuel, number of mobiles and types of roadmaps. The dynamic roadmap features of GRID and MICONIC-10 are more unusual and will be discussed in greater detail further on. However, the following general results are also applicable to these two domains, as they have special cases without dynamic roadmaps, namely GRID without doors and MICONIC-10 without special passengers.

Definition 7 TRANSPORT *task*

A TRANSPORT ***task*** *is a 9-tuple* $(V, M, P, l_0, P_G, l_G, fuel_0, cap, road)$*, where*

- V *is a finite set of* ***locations***,
- M *is a finite set of* ***mobiles***,
- P *is a finite set of* ***portables***,
- $l_0 : (M \cup P) \to V$ *is the* ***initial location*** *function,*
- $P_G \subseteq P$ *is the set of* ***goal portables***,
- $l_G : P_G \to V$ *is the* ***goal location*** *function,*
- $fuel_0 : V \to \mathbb{N} \cup \{\infty\}$ *is the* ***initial fuel*** *function,*
- $cap : M \to \mathbb{N}$ *is the* ***capacity*** *function, and finally*
- $road : M \to \mathbb{P}(V \times V)$ *is the* ***roadmap*** *function.*

We require that V*,* M *and* P *are disjoint, and that* $(V, road(m))$ *is an undirected graph for all* $m \in M$ *(i. e., for all* $m \in M$*, the relation* $road(m)$ *is symmetric and irreflexive).*

The concepts of locations, mobiles and portables should be clear from the preceding discussion. All mobiles and portables have a specified initial location. Mobiles are initially unloaded and hence portables do not start within mobiles. Some portables, i. e. the set of goal portables, have a specified final

location, given by the goal location function. The location of other portables and of mobiles after the execution of a plan does not matter.

The fuel function bounds the number of times a given location can be left by a mobile. Fuel is associated with locations rather than mobiles because this is the way it is handled in MYSTERY and MYSTERY'. The carrying capacity function bounds the number of portables a given mobile can carry at the same time. The roadmap function specifies individual roads for each mobile. For mobiles m, we will call $(V, road(m))$ the *roadmap graph* or simply *roadmap* of that mobile. We only investigate undirected roadmap graphs because only these occur in the benchmark domains.

That said, we can now define the TRANSPORT planning domain.

Definition 8 TRANSPORT *domain*

The TRANSPORT *domain maps* TRANSPORT *tasks with locations V, mobiles M and portables P to planning task state models as follows.*

The set of states consists of all pairs $(l, fuel)$ of current location *functions $l : M \cup P \to V \cup M$ and* fuel reserve *functions $fuel : V \to \{1, \ldots, fuel_{\max}\} \cup \{\infty\}$, where $fuel_{\max}$ is the maximum amount of initial fuel of any location with finite fuel.*

The initial state is given by the initial location and initial fuel functions. The set of goal states consists of those states where the current location of all goal portables matches their goal location.

The set of actions consists of movement *actions $move_{m,v}$, which move mobile m to location v,* pickup *actions $pick_{m,p}$, which cause mobile m to pick up portable p, and* drop *actions $drop_p$, causing portable p to be dropped by the mobile currently carrying it.*

The action $move_{m,v}$ is defined in all states $(l, fuel)$ where $(l(m), v)$ is part of the roadmap of m and the fuel reserve at $l(m)$ is non-zero. Its application results in state $(l \oplus (m, v), fuel \oplus (l(m), fuel(l(m)) - 1))$, where we define $\infty - 1 = \infty$. The functional overloading *notation $f \oplus (a', b')$ denotes the function f' with $f'(a') = b'$ and $f'(a) = f(a)$ for all $a \neq a'$.*

The action $pick_{m,p}$ is defined in all states $(l, fuel)$ such that $l(m) = l(p)$ and the number of portables p' with current location m is strictly less than the capacity of m. It results in state $(l \oplus (p, m), fuel)$.

The action $drop_p$ is defined in all states $(l, fuel)$ such that $l(p) \in M$. It results in state $(l \oplus (p, l(l(p))), fuel)$.

i: capacity	1: one portable	∞: unbounded	$*$: varies
j: fuel units	1: one per location	∞: unbounded	$*$: varies
k: mobiles	1: one mobile	$+$: one roadmap	$*$: many roadmaps

Fig. 2. The TRANSPORT$_{ijk}$ domains.

As was seen in the description of the various transportation domains from the planning competitions, we usually do not need all features of TRANSPORT. Some benchmarks only feature a single agent (mobile). In others, there are no fuel or capacity restrictions. For this reason, we will now define some special cases of TRANSPORT that capture some of the most frequently occurring restrictions of the general theme.

Definition 9 *Special cases of* TRANSPORT

For $i, j \in \{1, \infty, *\}$ and $k \in \{1, +, *\}$, the TRANSPORT$_{ijk}$ *domain is defined as the restriction of* TRANSPORT *to the tasks which satisfy the following requirements:*

- *If $i = 1$, the capacity of all mobiles must be equal to one. The* GRID *domain has this property.*
- *If $i = \infty$, the capacity of all mobiles must be* unbounded*, i. e. equal to the number of portables. Examples are the* LOGISTICS *and* MICONIC-10 *domains.*
- *If $i = *$, there are no restrictions on the capacity function. Mobiles can have varying capacities, as in* MYSTERY *and* MYSTERY'.
- *If $j = 1$, the initial fuel of all locations must be equal to one. Consequently, each location can be left by a mobile at most once.*
- *If $j = \infty$, the initial fuel of all locations must be infinite. There are no fuel requirements, like in all benchmarks except* MYSTERY *and* MYSTERY'.
- *If $j = *$, there are no restrictions on the initial fuel function. Locations can have varying amounts of fuel, as in* MYSTERY *and* MYSTERY'.
- *If $k = 1$, there is only one mobile, as in* GRID, GRIPPER *and* MICONIC-10.
- *If $k = +$, there may be several mobiles, but they must all have the same roadmap, as in* MYSTERY *and* MYSTERY'.
- *If $k = *$, there are no restrictions on the roadmap function. Different mobiles can use different roads, as in* LOGISTICS.

Since there are three options for each of the three parameters that can be combined in any way, there is a total of 27 planning domains which form the TRANSPORT *domain family*. The definition is summarized in Figure 2.

Note that for all three parameters, $*$ is a generalization of the other options, that $k = +$ generalizes $k = 1$, and that apart from these, no option supersedes another. These relationships, illustrated in Figure 3, imply that

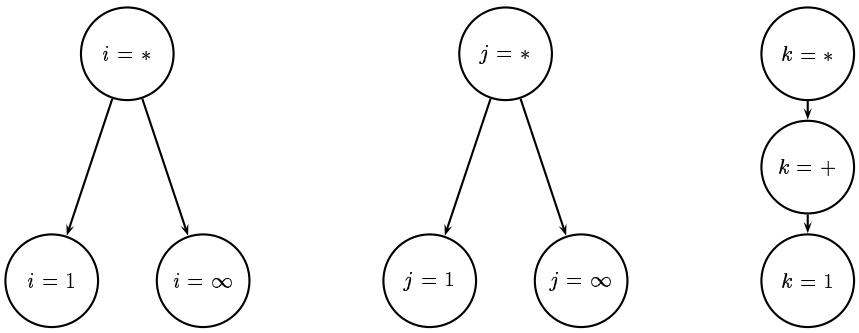

Fig. 3. Generalization relationships for the TRANSPORT$_{ijk}$ domains.

there is a single most general domain, TRANSPORT$_{***}$, which is identical to TRANSPORT, and there are four domains having no proper specializations in the family, namely TRANSPORT$_{111}$, TRANSPORT$_{1\infty1}$, TRANSPORT$_{\infty11}$, and TRANSPORT$_{\infty\infty1}$. This is formalized in the following corollary.

Corollary 10 *Inclusion relationships*

For $i, i', j, j' \in \{1, \infty, *\}$ and $k, k' \in \{1, +, *\}$, if the following three properties are true:

- *$i = i'$ or $i' = *$,*
- *$j = j'$ or $j' = *$, and*
- *$k = k'$ or $k' = *$ or $(k, k') = (1, +)$, then*

PLANEX-TRANSPORT$_{ijk} \leq_p$ PLANEX-TRANSPORT$_{i'j'k'}$ *and*
PLANLEN-TRANSPORT$_{ijk} \leq_p$ PLANLEN-TRANSPORT$_{i'j'k'}$.

The following result shows that shortest solutions to TRANSPORT tasks only consist of a polynomial number of steps, which is an important property of the task family.

Theorem 11 *Polynomial length plans for* TRANSPORT

There is a polynomial p such that for all TRANSPORT *tasks T that have a solution, there is a solution of length at most $p(||T||)$.*

PROOF. *Let T be a solvable* TRANSPORT *task with locations V, mobiles M and portables P. Assume $||T|| = |V| + |M| + |P|$, which is certainly an underestimation.*

Each portable only needs to be moved to each location at most once, which bounds the number of pickup and drop actions required in a plan by $|V| \cdot |P|$, or $O(||T||^2)$, each.

In between two pickup or drop actions (and before the first and after the last

such action), no mobile should visit a given location twice, which bounds the number of movement actions by $(2(|V| \cdot |P|) + 1) \cdot (|V| \cdot |M|)$*, or* $O(||T||^4)$*. Adding the bounds together results in a total upper bound of* $O(||T||^4)$*.* □

An immediate consequence of this polynomial solution length property is that valid plans can be guessed and verified in polynomial time by a non-deterministic algorithm, leading to the following corollary.

Corollary 12 *Membership in* **NP** *for* TRANSPORT *planning*

The plan existence and bounded plan existence problems for TRANSPORT_{ijk} *are in* **NP** *for all values of* i*,* j*,* k*.*

3.2 Plan Existence

We will now discuss the different TRANSPORT planning problems in detail, starting with plan existence. Our first result shows that the existence of many mobiles with different roadmaps and different carrying capacities does not make (non-optimal) transportation planning difficult.

Theorem 13 *Complexity of* $\text{PLANEX-TRANSPORT}_{*\infty*}$

The plan existence problem for $\text{TRANSPORT}_{*\infty*}$ *can be decided in polynomial time.*

PROOF. *Let* V *be the set of locations of the given* TRANSPORT *task and road be the roadmap function. For each mobile* m *with non-zero carrying capacity, we perform a breadth-first search on* $(V, road(m))$*, starting at the initial location of* m*. All roads that are reached by one of these breadth-first explorations can be used by a mobile carrying a portable. In the absence of fuel constraints, all such roads can be used any number of times.*

Consequently, the task can be solved if and only if for each portable, its goal location can be reached from its initial location using only these roads. This can easily be decided in polynomial time, and in fact the actual plans can easily be generated. □

The previous result addresses the most general domain in the family which features unrestricted fuel. The remaining domains, those involving fuel constraints, are all generalizations of TRANSPORT_{111} or $\text{TRANSPORT}_{\infty 11}$. We will now show that for these two, and consequently for all domains with restricted fuel, deciding plan existence is **NP**-complete.

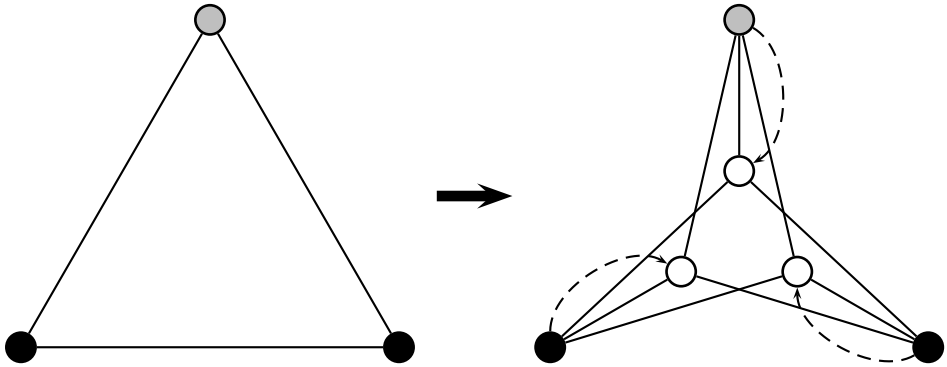

Fig. 4. Graph and corresponding TRANSPORT$_{111}$ task.

Theorem 14 *Complexity of* PLANEX-TRANSPORT$_{111}$

The plan existence problem for TRANSPORT$_{111}$ *is* **NP***-complete.*

PROOF. *Membership in* **NP** *was shown in Corollary 12. We prove* **NP***-hardness by reduction from the* **NP***-complete problem of finding a Hamiltonian path with a fixed start vertex [11, Problem GT39].*

Let (V, E) *be a graph and* $v_1 \in V$*. Then* (V, E) *contains a Hamiltonian path starting at* v_1 *if and only if there is a solution for the* TRANSPORT$_{111}$ *task defined as follows: For each* $v \in V$*, there are two distinct locations* v *(called an* entrance*) and* v^* *(called an* exit*), with one unit of fuel each. At each entrance, there is a portable to be moved to the corresponding exit. There is only one mobile, of capacity one, starting at the entrance* v_1*. There are roads between* v *and* v^* *for all* $v \in V$ *and between* u^* *and* v *for all* $(u, v) \in E$*.*

This mapping is illustrated in Figure 4. Entrance locations are marked by black, exits by white nodes, and gray nodes indicate the vertex (and entrance location) v_1*. Portables are pictured by dashed lines pointing from their respective initial to their respective final locations.*

Under this mapping, if there is a Hamiltonian path in (V, E) *starting at* v_1*, say* $[v_1, \ldots, v_n]$*, then there is a solution for the planning task where the movement path of the mobile is* $[v_1, v_1^*, \ldots, v_n, v_n^*]$ *and portables are picked up and dropped in the obvious way.*

Now consider there is a solution to the planning task. Whenever a portable is picked up (at an entrance), the only reasonable thing to do is to move to its destination (the corresponding exit) and drop it, because there is no point in deferring that movement when the carrying capacity is exhausted. The mobile must then proceed to the next entrance, which is only possible in the ways defined by the edges in the original graph. Thus, the plan corresponds to a path in the original graph that visits every vertex. To prove that this corresponds

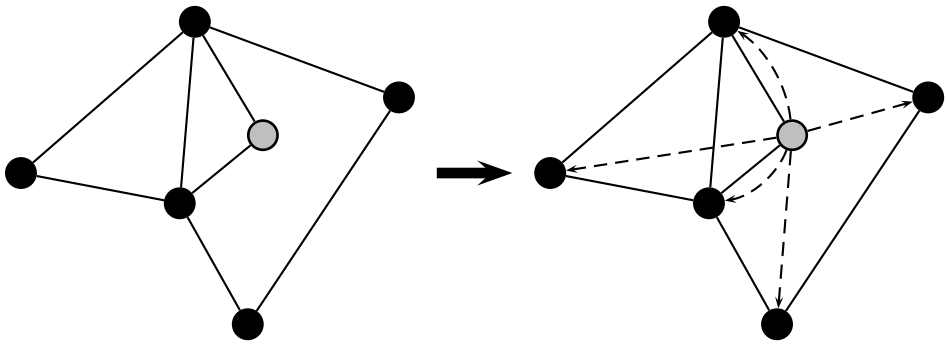

Fig. 5. Graph and corresponding TRANSPORT$_{\infty 11}$ task.

to a Hamiltonian path, we need to check that no entrance location is ever visited twice. We can safely assume this: Whenever a location is visited for the second time, it can never be left again because of fuel constraints. In a successful plan, this can only happen after each portable has been dropped off at its final destination, and hence that movement can safely be omitted. Thus, if the planning task has a solution, there is a Hamiltonian path in the original graph. □

The same reduction (except for the carrying capacity of the mobile) could be used for proving **NP**-completeness of PLANEX-TRANSPORT$_{\infty 11}$. However, we give another proof for this, which is more straight-forward and also shows that the problem is already hard if the roadmap of the mobile is restricted to be a planar graph.

Theorem 15 *Complexity of* PLANEX-TRANSPORT$_{\infty 11}$

The plan existence problem for TRANSPORT$_{\infty 11}$ *is* **NP***-complete, even if the roadmap of the only mobile is restricted to be a planar graph.*

PROOF. *Membership in* **NP** *was shown in Corollary 12. For hardness, we reduce from the Hamiltonian Path problem with a fixed start vertex in a planar graph [7]. Let* (V, E) *be the graph and* $v_1 \in V$*. Then* (V, E) *contains a Hamiltonian path starting at* v_1 *if and only if there is a solution for the* TRANSPORT$_{\infty 11}$ *task defined as follows: The set of locations is* V*, and there is a single mobile, of unlimited capacity, with roadmap* (V, E) *and initial location* v_1*. Each location provides one unit of fuel, and there is one portable to be delivered from* v_1 *to each location from* $V \setminus \{v_1\}$*. This is illustrated in Figure 5, where again gray nodes stand for the start vertex and initial location of the mobile and dashed arrows indicate portables.*

Clearly, this problem is solvable if and only if there is a Hamiltonian Path in (V, E) *starting at* v_1*, with Hamiltonian Paths corresponding to movement*

paths of the mobile. □

This concludes our analysis of the PLANEX-TRANSPORT$_{ijk}$ decision problems. Putting the previous three results together, we conclude that they can be solved in polynomial time if $j = \infty$ and are **NP**-complete otherwise.

Corollary 16 *Summary for* PLANEX-TRANSPORT

The plan existence problem for TRANSPORT$_{ijk}$ *can be decided in polynomial time if* $j = \infty$ *and is* **NP***-complete otherwise.*

3.3 Bounded Plan Existence

In this section, we investigate the bounded plan existence problem for transportation domains without fuel constraints. We do not have to discuss the restricted fuel case: **NP**-completeness for these problems is already entailed by the corresponding results for PLANEX, applying Corollaries 12 and 16.

In fact, the proofs of Theorems 14 and 15 can be adjusted to prove **NP**-completeness of PLANLEN-TRANSPORT$_{1\infty1}$ and PLANLEN-TRANSPORT$_{\infty\infty1}$ by removing the fuel restrictions and introducing plan length bounds of $4|V|-1$ and $3|V|-3$, respectively. However, these results require allowing for arbitrary (or arbitrary planar) roadmaps, and thus do not apply to planning domains such as LOGISTICS or GRID. For that reason, we will prove some stronger results now.

Our first result applies to *grid* roadmaps, i. e. graphs with vertex set $\{0, \ldots, w-1\} \times \{0, \ldots, h-1\}$ for some $w, h \in \mathbb{N}$ (called *width* and *height* of the grid, respectively), where vertices (a, b) and (a', b') are connected by an edge if and only if $|a - a'| + |b - b'| = 1$. Note that grids are always planar graphs. The proof is based on the TRAVELING SALESMAN $\mathcal{L}_1$ METRIC decision problem, which is the special case of the TSP problem where all sites are points in $\mathbb{N}^2$ and the distance between sites (x, y) and (x', y') is given by their $\mathcal{L}_1$ or *Manhattan* distance $d_1((x, y), (x', y')) = |x' - x| + |y' - y|$. This problem is known to be **NP**-complete in the strong sense [11, Problem ND23]. We can thus assume that encoding sizes of site coordinates are linear (rather than logarithmic) in their magnitudes; otherwise the following transformation would not be polynomial.

Theorem 17 *Complexity of* PLANLEN-TRANSPORT$_{1\infty1/\infty\infty1}$

The bounded plan existence problems for TRANSPORT$_{1\infty1}$ *and* TRANSPORT$_{\infty\infty1}$ *are* **NP***-complete, even if the roadmap of the only mobile is restricted to be a*

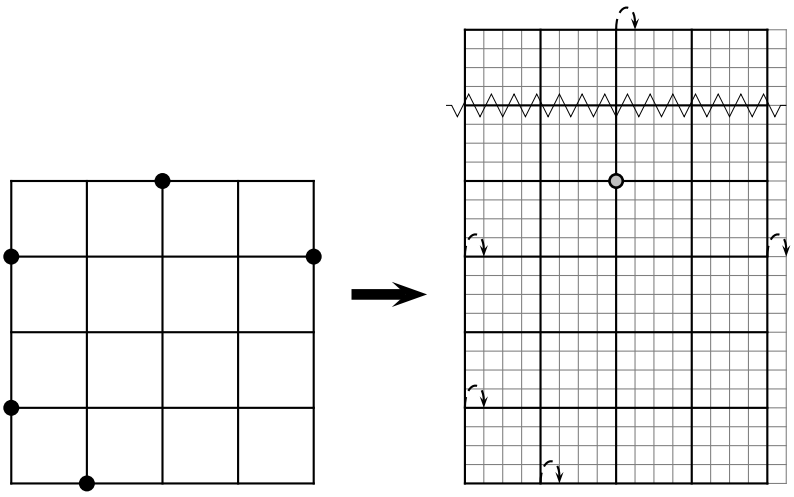

Fig. 6. TRAVELING SALESMAN $\mathcal{L}_1$ METRIC sites and corresponding TRANSPORT task.

grid and all portables only need to be moved to adjacent locations.

PROOF. *Membership in* **NP** *was shown in Corollary 12.*

Assume we are given a TRAVELING SALESMAN $\mathcal{L}_1$ METRIC *instance with site set S of cardinality $n > 0$ and tour length bound B. Let $x_{\max}$ and $y_{\max}$ be the maximum x and y coordinates in S, respectively, and let $s^* = (x^*, y^*) \in S$ be a site that maximizes the y coordinate, i. e. a northmost site. Furthermore, we set $k = 4n$ and $D = kn \cdot (x_{\max} + y_{\max}) + 4n$.*

We map this TSP instance to a TRANSPORT *task with the following properties: There is a single mobile of capacity one (for obtaining a* $\text{TRANSPORT}_{1\infty1}$ *task) or unbounded capacity (for obtaining a* $\text{TRANSPORT}_{\infty\infty1}$ *task). Its roadmap is a grid of width $kx_{\max}+2$ and height $ky_{\max}+D+1$. There are no fuel restrictions.*

For each site $(x, y) \in S \setminus \{s^*\}$, there is a portable which is initially located at (kx, ky) and needs to be moved to $(kx+1, ky)$. The mobile is initially located at (kx^*, ky^*), and an additional portable, called the remote portable, *is initially located at $(kx^*, ky^* + D)$ and needs to be moved to $(kx^* + 1, ky^* + D)$. The bound on plan length is defined as $K = kB + D + 4n - 1$.*

We restate this transformation in words to make it more understandable. First, we scale all site coordinates by a factor k. Then, we place a portable at each site excluding the northmost one, where we place the mobile. We then place the remote portable far up north (D units after scaling), on the same column as the mobile. The goal is to move all the portables one unit to the right. Figure 6 gives an example of the mapping. For reasons of clearness, the figure is not to the scale: The scaling factor k and the distance between the initial locations of the mobile and the remote portable should be much higher. Again, the initial

location of the mobile is painted gray, and portables are depicted by dashed arrows.

We will now argue that there is a traveling salesman tour of length at most B in the original TSP instance if and only if the corresponding planning task has a solution of length at most K. First assume that there is such a tour. It is easy to see that there is a strong connection between $\mathcal{L}_1$ distances and movements in the planning task: The shortest movement sequence to get from p to q on a grid roadmap consists of $d_1(p, q)$ steps.

Thus, taking the scaling constant k into account, there is a sequence of at most kB movements that passes through the initial locations of all portables except the remote one and then returns to the location of origin. With another D movements, the mobile can then reach the remote portable. All that remains to be done now is to insert four actions whenever a location containing a portable is encountered: pick up that portable, move east, drop it, move west again. The last movement is not needed for the remote portable. This leads to a plan of length at most $kB + D + 4n - 1 = K$, as required.

Now assume that there is no *traveling salesman tour of length B or less, i. e. the shortest tour has length $B+1$ or more. We will show that all plans consist of more than K steps. First note that the distance between any two sites is at most $x_{\max} + y_{\max}$, where the maximum is assumed in the most extreme case of the sites being located in opposite corners of the grid. Because a tour is the sum of n distances, there is* always *a traveling salesman tour of length at most $n(x_{\max} + y_{\max})$, so B cannot be greater than this value. Using the definition of D, this implies that $D \geq kB + 4n$.*

First consider any plan where the remote portable is picked up for the first time at some point before all the other portables have been moved to their goal locations. In this case the mobile will at some point have to move from its initial row $ky_{\max}$ to row $ky_{\max} + D$ to pick up the remote portable and later get back to row $ky_{\max}$ or below to drop other portables. This will involve at least $2D$ movements, thus plan length will be at least $2D \geq kB + D + 4n > K$, exceeding the boundary.

Therefore we only need to consider the case where the remote portable is picked up for the first time after all the other portables have been dropped at their goal location. We can safely assume that the movement between the last but one portable to be dropped and the remote portable passes through the initial location of the mobile. If it does not, the movement path can be adjusted to achieve this without increasing plan length by first moving eastwards or westwards until the column of the remote portable has been reached, then moving northwards.

For each portable, the mobile must at some point move to the initial location of that portable, and we just argued that it will return to its own initial location

at some point. The number of movement actions to achieve this cannot be less than the length of the shortest traveling salesman tour for the set of sites obtained by scaling each site in S by a factor of k. Thus, at least $k(B+1)$ move actions will be needed for this, plus D move actions for getting from the initial location to the location of the remote portable.

This adds up to a lower bound of $k(B+1)+D = kB+k+D = kB+D+4n > K$ actions. Thus, no plan of length at most K exists. □

For the unrestricted capacity case, there is another interesting special case, relevant to the LOGISTICS domain, for which we can prove **NP**-completeness. The proof utilizes the FEEDBACK VERTEX SET problem [11, Problem GT7], which is defined as follows. Given a digraph $G = (V, A)$ and a natural number K, does G have a feedback vertex set of cardinality at most K? A feedback vertex set is a subset $V' \subseteq V$ such that the subgraph induced by $V \setminus V'$ is acyclic (and hence, no longer contains any "feedback").

Theorem 18 *Complexity of* PLANLEN-TRANSPORT$_{\infty\infty1}$

The bounded plan existence problem for TRANSPORT$_{\infty\infty1}$ *is* **NP***-complete, even if the roadmap is restricted to be a complete graph.*

PROOF. *Membership in* **NP** *was shown in Corollary 12. For hardness, we reduce from the Feedback Vertex Set problem [11, Problem GT7]. Let (V, A) be a digraph and K be a natural number. Then (V, A) contains a feedback vertex set of size at most K if and only if there is a solution of length at most $3|V| + 2|A| + K$ for the* TRANSPORT$_{\infty\infty1}$ *task defined as follows. There is a single mobile of unbounded capacity. Its roadmap is a complete graph over the locations V and an additional location v_0, which is the initial location of the mobile. There is one portable to be moved from v_0 to each other location and one portable to be moved from u to v for each $(u, v) \in A$. There are no fuel constraints.*

Figure 7 illustrates the reduction. As usual, the initial location is depicted in gray and portables correspond to dashed arrows. A minimal feedback vertex set (and the corresponding locations of the TRANSPORT *task) are highlighted with white nodes.*

Note that if (V, A) contains an arc from some vertex u to a vertex v, then the mobile has to visit location u before location v at least once, to pick up a portable that must be moved from u to v. This means that for each cycle in the digraph, the mobile must visit some location that corresponds to a vertex in the cycle at least twice.

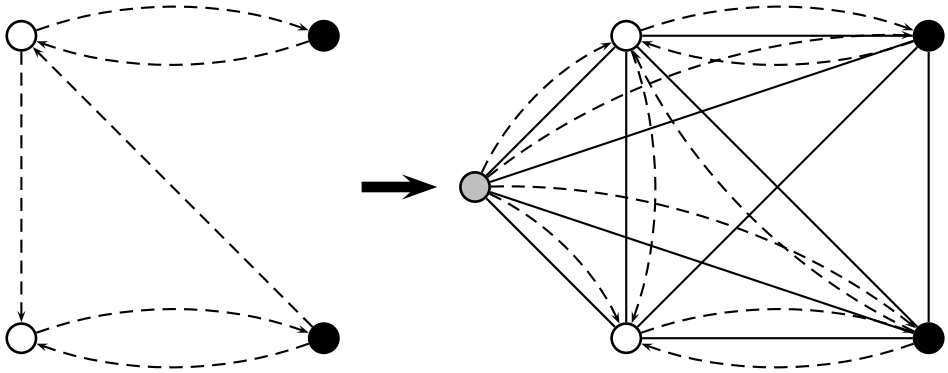

Fig. 7. Directed graph and corresponding TRANSPORT$_{\infty\infty 1}$ task.

We observe that for each feedback vertex set $V' \subseteq V$, the planning task can be solved by moving the mobile to the vertices from V' in any order, then to the vertices from $V \setminus V'$ in an order which corresponds to a topological sort of the subgraph induced by $V \setminus V'$ (which must be acyclic because V' is a feedback vertex set), and finally to the vertices from V' again, in any order, picking up and dropping portables in the obvious way. This requires $|A| + |V|$ pickup and drop actions each and $|V| + |V'|$ movements, totaling a number of actions bounded by $3|V| + 2|A| + K$ if $|V'| \leq K$.

On the other hand, any plan must contain at least one pickup and drop action for each portable and visit each location at least once, totaling $3|V| + 2|A|$ actions. Consequently, if a plan does not exceed the given length bound, no more than K locations can be visited more than once. These locations must form a feedback vertex set: If there were cycles that touched no such location, some other location would have to be revisited, too. □

The previous result shows that it is not (only) the route planning aspect of the transportation tasks that makes them hard to solve optimally, as there is no real route planning involved when the roadmap is a complete graph. The source of difficulty rather seems to be the interaction between the individual subgoals, i. e. goal portables.

We end the discussion of the bounded plan existence problem for the TRANSPORT domain family with the following summary.

Corollary 19 *Summary for* PLANLEN-TRANSPORT

The bounded plan existence problem for TRANSPORT$_{ijk}$ *is* **NP***-complete for all values of i, j, k.*

In AI planning, sequential length is not the only wide-spread quality criterion for a plan. People are also interested in short *parallel* plans, where several actions can be applied at the same time, provided they do not interfere. To make this precise, we would need to formally define some notion of *interference*. Three different ways of approaching this task come to mind in the TRANSPORT framework:

- Use **Graphplan**-style [12] parallelism.
- Different mobiles can act at the same time, but no two transitions that affect the same mobile (picking up or dropping a portable or moving with that mobile) can occur simultaneously.
- Different mobiles can act at the same time, and the same mobile can perform multiple actions simultaneously if it does not move.

A drawback of **Graphplan**-style parallelism is that it critically depends on the way the planning problems are formalized in the PDDL language. This has the peculiar effect that in some planning domains like LOGISTICS and GRIPPER, **Graphplan** allows for multiple portables to be picked up simultaneously by the same mobile, whereas this is not possible in others, such as MYSTERY. More severely, in the MYSTERY domain **Graphplan** does not even allow for multiple mobiles to leave the same location simultaneously, which is hard to justify.

Does this imply that in order to discuss the parallel planning problem in a satisfying way, we need to conduct a separate analysis for all of the three different kinds of parallelism we mentioned? Fortunately, this is not the case. First of all, all our hardness results were obtained for special cases where there is only one mobile, and with that restriction, **Graphplan**-style parallelism in the competition domains is in all cases equivalent to one of the other two forms of parallelism. What is more, in this case, with only one mobile, the second kind of parallelism does not allow for any concurrent activity at all and hence the hardness results for bounded (sequential) plan existence immediately apply.

Hence we only need to discuss the third notion of parallelism: Multiple mobiles can act simultaneously, and the same mobile can pick up and/or drop multiple portables at the same time. Fortunately, membership in **NP** can be established just as easily in this case, and the previous proofs of hardness still apply with some minor adjustments. The proof of Theorem 17 carries over verbatim, and for Theorem 18, we only need to change the bound on plan length from $2|A|+3|V|+K$ to $2|V|+2K+1$, counting $|V|+K$ plan steps for the movements of the mobile and $|V|+K+1$ plan steps to do all the necessary pickup and drop actions, one at the beginning and one after each movement.

Thus, similar to Corollary 19 we observe the following.

Remark 20 PLANLEN-TRANSPORT*, parallel plans*

The bounded parallel plan existence problem for TRANSPORT$_{ijk}$ *is* **NP***-complete for all values of i, j, k.*

We call this a "remark" rather than a theorem because we have not formalized a notion of "parallel plans", but for all formalizations mentioned above, we have argued that this is a valid theorem.

3.5 MYSTERY, MYSTERY', LOGISTICS *and* GRIPPER

Having completed the analysis of the TRANSPORT domain family, we can now apply the results to the transportation benchmarks from the AIPS competitions, beginning with the MYSTERY and MYSTERY' domains. All bounded plan existence results for the transportation benchmarks extend to the parallel case, for reasons detailed in the previous discussion.

Definition 21 MYSTERY *domain and* MYSTERY' *domain*

The MYSTERY *planning domain is the restriction of* TRANSPORT$_{**+}$ *to those planning tasks where the roadmaps are planar graphs and the amount of fuel at each location is finite.*

The MYSTERY' *planning domain is identical to* MYSTERY *except that for all locations v, v' such that $v \neq v'$, it introduces an additional action* squirt$_{v,v'}$*.*

Action squirt$_{v,v'}$ *is defined in all states $(l, fuel)$ with $fuel(v) \geq 2$, mapping the state to $(l, fuel \oplus (v, fuel(v) - 1) \oplus (v', fuel(v') + 1))$.*

According to this definition, MYSTERY is a planning domain that falls into the TRANSPORT hierarchy, and MYSTERY' is a closely related planning domain with additional actions that move fuel between locations. Our previous results can be applied to these two benchmarks immediately.

Theorem 22 *Complexity of* MYSTERY *and* MYSTERY' *planning*

The plan existence and bounded plan existence problems for MYSTERY *and* MYSTERY' *are* **NP***-complete, even if there is only a single mobile, there are no capacity constraints, and there is exactly one unit of fuel at each location.*

PROOF. *For* MYSTERY*, hardness of plan existence (and thus also of bounded*

plan existence) follows immediately from Theorem 15. Membership in **NP** *follows from Corollary 12.*

For MYSTERY', *the hardness results are equally applicable, because for the restricted class of tasks considered (those with one unit of fuel per location), the two planning domains are identical.*

For membership in **NP**, *we can again argue that there is a polynomial bound on the length of shortest plans for solvable tasks: The bound for the number of movements, pick-up and drop actions still holds, and minimal plans contain no more movements of fuel ("squirt" actions) than movements of mobiles.* □

Next, we discuss the well-known LOGISTICS domain, which featured prominently in both AIPS competitions.

Definition 23 LOGISTICS *domain*

The LOGISTICS *planning domain is the restriction of* $\text{TRANSPORT}_{\infty\infty*}$ *to those planning tasks where there exists a partition* $\mathcal{C}$ *of the location set (called the set of* ***cities****) and a set of locations* A *(called* ***airports****), such that for each mobile* m, *either* $road(m) = \{ (v, v') \mid v, v' \in A, v \neq v' \}$, *in which case* m *is called an* ***airplane****, or* $road(m) = \bigcup_{C \in \mathcal{C}} \{ (v, v') \mid v, v' \in C, v \neq v' \}$, *in which case* m *is called a* ***truck****.*

Again, the complexity of planning in the LOGISTICS domain follows immediately from our previous results.

Theorem 24 *Complexity of* LOGISTICS *planning*

The plan existence problem for LOGISTICS *can be decided in polynomial time. The bounded plan existence problem for* LOGISTICS *is* **NP***-complete, even if there is only a single city, a single truck, and no airplane (or equivalently, no trucks and a single airplane).*

PROOF. *The plan existence result follows from Theorem 13. The bounded plan existence result follows from Corollary 12 and Theorem 18.* □

The last benchmark that completely fits into the TRANSPORT framework, GRIPPER, is somewhat simplistic, as the following definition shows.

Definition 25 GRIPPER *domain*

The GRIPPER *planning domain is the restriction of* $\text{TRANSPORT}_{*\infty 1}$ *to those planning tasks where there are exactly two locations* a *and* b, *the mobile and*

all portables are initially located at a, all portables must be moved to b, and the mobile has a carrying capacity of two.

For reasons of completeness, we record the following result.

Theorem 26 *Complexity of* GRIPPER *planning*

The plan existence and bounded plan existence problems for GRIPPER *can be decided in polynomial time.*

PROOF. *Gripper tasks are always solvable, so deciding plan existence is trivial. For deciding bounded plan existence, an optimal plan can be generated to compare its length to the specified bound.*

The following strategy generates a plan of minimal length: If there are at least two portables at the initial location, choose any two of them and pick them up. If there is only one portable, pick it up. If there is no portable, nothing needs to be done. If it was necessary to pick something up, move to the other location and drop the load. If this does not result in a goal state, move back and iterate. Clearly this leads to an optimal plan and can be done in time polynomial in the number of portables. □

Of course, it would also be easy (and faster) to solve the bounded plan existence problem for GRIPPER without actually generating a plan, but we prefer to provide actual planning algorithms for the problems that we can decide in polynomial time.

3.6 GRID

The next domain we want to discuss, called GRID, is different to the previously discussed benchmarks in that it features a *dynamic roadmap*: Locations can be initially *locked* (and hence inaccessible) and later become accessible through the invocation of *open* actions, if the (single) mobile carries an appropriate portable (key). Because of these additional properties, GRID tasks are not identical to TRANSPORT tasks, and we first need to define them.

Definition 27 GRID *task*

A GRID ***task*** *is a 3-tuple* (T, L, U)*, where*

- T *is a* $\text{TRANSPORT}_{1\infty1}$ *task with a grid roadmap with location set* V *and portable set* P*,*

- *$L \subseteq V$ is the set of initially* ***locked locations****, not containing the initial location of the mobile, and*
- *$U \subseteq P \times L$ is the* ***unlock relation****, satisfying the criterion that for any two portables the sets of related locations are either equal or disjoint.*

Portables in GRID *are called* ***keys****.*

The planning task state model that is obtained by applying the TRANSPORT *domain to T is called* the TRANSPORT model *of the task.*

Effectively, a GRID task is a special TRANSPORT task with two additional properties: There is a set of locations that are initially inaccessible, and there is a relation that specifies which keys can be used to gain access to which locations. The constraints on the unlock relation imply that two keys are either functionally equivalent or open disjoint sets of locations. In the original PDDL description, this is formalized with different "shapes" for keys and locks such that each key and lock has a specific shape and keys only fit into locks with matching shapes.

Because GRID tasks are extensions of TRANSPORT tasks, we define the GRID domain by reusing the definition of TRANSPORT.

Definition 28 GRID *domain*

The GRID *planning domain maps* GRID *tasks to planning task state models as follows.*

The set of states consists of all pairs (s, L) such that s is a state of the TRANSPORT *model and L is a set of locations. We say that a location is* locked *in a given state if it is an element of L.*

The initial state is defined by the initial state of the TRANSPORT *model and the set of initially locked locations. A state (s, L) is a goal state if and only if s is a goal state of the* TRANSPORT *model.*

The set of actions consists of the actions of the TRANSPORT *model, which are extended to* GRID *states as follows: Action a is defined in (s, L) if and only if it is defined in s, in which case it maps (s, L) to $(a(s), L)$. The only exception to this is that movements to locked locations are* not *defined.*

Additionally, there are swap *actions $swap_{p,p'}$ for all keys p, p' such that $p \neq p'$ and* unlock *actions $open_v$ for all locations v.*

The additional swap actions are compositions of drop and pickup actions, i. e. $swap_{p,p'} = pick_p \circ drop_{p'}$.

The action $open_v$ is defined in all states (s, L) where v is locked, the current

location of the mobile m is adjacent to v in the grid, and there is a key k with current location m such that (k, v) is in the unlock relation. Applying the action to the state results in $(s, L \setminus \{v\})$.

As seen in the definition, states in GRID consist of the usual parts of a TRANSPORT state as well as the set of locations that are currently locked. The only adjustment to the usual actions of TRANSPORT task state models is the restriction of movement actions to only move to unlocked destinations. The new "swap" action is a combined drop and pickup action, and the new "open" actions unlocks a locked location adjacent to the mobile, provided it is currently carrying a matching key.

We will now discuss the complexity of planning in the GRID domain.

Theorem 29 *Complexity of* GRID *planning*

The plan existence problem for GRID *can be decided in polynomial time. The bounded plan existence problem for* GRID *is* **NP***-complete, even if all locations are initially unlocked and all keys only need to be moved to adjacent locations.*

PROOF. *For plan existence, we devise a polynomial algorithm that generates a plan if one exists or else reports that none exists. First, calculate the* reachable area*, i. e. the set of locations that can be reached by the mobile from the initial location by a sequence of movement actions (without opening locations). If any key located within the reachable area can open a locked location adjacent to the reachable area, move to such a key, pick it up, move to some location it can unlock, open that location, and drop the key. Recalculate the reachable area and iterate until no further locations can be unlocked.*

Because the number of locked locations and the number of actions required to open a locked location are each bounded by a polynomial in the input size, this is a polynomial algorithm. The resulting situation can clearly be solved if and only if the initial situation could be solved, and its solvability can be checked with a decision procedure that decides plan existence for $\text{TRANSPORT}_{1\infty1}$ *tasks, such as the one from Theorem 13.*

This also shows the existence of polynomial bounds on the length of shortest plans in the GRID *domain, implying membership in* **NP** *for the bounded plan existence problem. Hardness, even under the restrictions cited above, follows from Theorem 17. Although this result is not immediately applicable, because* TRANSPORT *has no combined pickup and drop actions, the reasoning in the proof is still valid.* □

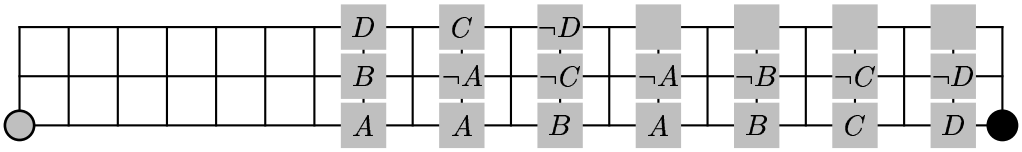

Fig. 8. GRID task corresponding to $(A \vee B \vee D) \wedge (A \vee \neg A \vee C) \wedge (B \vee \neg C \vee \neg D)$.

Concluding our discussion of GRID, we give another proof of **NP**-hardness for the bounded plan existence problem. Unlike the previous result, this proof does not emphasize the difficulty of goal ordering. We prove **NP**-completeness for the case of a single goal key. The hardness of the generated tasks lies in deciding which locations to open in order to get access to other parts of the map, and the decision problem used in the transformation is not related to "route planning" in any obvious way. In fact, the route planning part of the generated planning tasks is trivial.

Theorem 30 *Complexity of* GRID *planning with one goal key*

The bounded plan existence problem for GRID *is* **NP***-complete, even if there is only one goal key, all keys and the mobile start at the same location and the height of the grid graph is fixed to three.*

PROOF. *Membership in* **NP** *follows from the previous theorem. Proof of hardness is by reduction from* 3SAT*. Let* (V, C) *be a* 3SAT *instance, where* $V = \{v_1, \ldots, v_n\}$ *is a set of variables, and* C *is a set of* m *clauses over* V*, each clause consisting of exactly three literals.*

Let $W = (2n+1)(2m+2n)+4(n+m)^2+11(n+m)+6$ *and* $w = W+2m+2n$*. The roadmap grid of the corresponding* GRID *task has width* $w+1$ *and height* 3*, and the initial location of the mobile and all keys is* $(0, 0)$*. For each literal over* V*, there is a corresponding key. Additionally, there is a single goal key, to be moved to* $(w, 0)$*.*

For $i \in \{0, \ldots, w\}$*, we call the set of locations* $\{(i, 0), (i, 1), (i, 2)\}$ *the* i*-th* column *of the grid. The set of initially locked locations consists of the columns* $\{W+1, W+3, \ldots, W+2m-1\}$*, called the* clause barriers*, and the columns* $\{W+2m+1, W+2m+3, \ldots, W+2m+2n-1$*, called the* variable barriers*.*

Each clause barrier corresponds to a different clause of the 3SAT *instance, and each location of the barrier corresponds to a different literal of that clause: It can only be opened by the corresponding key.*

Similarly, each variable barrier corresponds to a different variable v *of the* 3SAT *instance. One of its locations can only be opened by the key correspond-*

ing to v, one only by the key corresponding to $\neg v$, and one has no matching key.

For an example, see Figure 8, where the gray node indicates the initial location of the mobile and keys and the black node indicates the destination of the goal key. Locked locations are marked as boxes labeled with the corresponding key. The empty corridor to the left of the clause barriers in the figure is not to the scale and should be far wider.

Clearly, this mapping can be performed in time which is polynomial in the size of a unary encoding of the Traveling Salesman $\mathcal{L}_1$ Metric *instance and results in a valid* Grid *task. We claim that the task has a solution of length at most $(2n+2)W$ if and only if the* 3SAT *instance has a solution.*

The intuition behind the mapping is the following: Because the mobile can only carry one key at a given time, and because there is a wide corridor to bridge between the initial location and the locked locations, only a limited number of keys can be used for opening the various locks. We say that a literal is "selected" if the corresponding key is used for opening a location.

Specifically, assume that a plan of length at most $(2n+2)W$ exists. In such a plan, at most $n+1$ keys can be carried across the empty corridor, as using $n+2$ keys would require at least $(2n+3)W$ movements. Because one of those keys must be the goal key, at most n literals are selected. Because of the way the variable barriers are organized, this implies that exactly *one literal for each variable is selected, and hence the selection of literals defines a truth assignment. It is a satisfying assignment, because it contains at least one literal from each clause – otherwise the mobile could not get past the clause barriers.*

Now assume there is a satisfying truth assignment for the propositional formula. We must show that there is a plan of length at most $(2n+2)W$. We mark the keys that correspond to true literals of the truth assignment as "selected" and adopt the following strategy:

(1) For each of the $n+m$ barriers, from left to right:
   (a) Randomly choose a selected key that can open one of the locations of the barrier.
   (b) Move to the current location of that key on a shortest path.
   (c) Pick up the key, move to the location left of the one to be opened on a shortest path, open the location and drop the key.
(2) Move to location $(0,0)$, pick up the goal key, move to $(w,0)$ and drop the key.

This clearly solves the task. We now have to bound the number of actions required. There are exactly $n+m+1$ pickup and drop actions each, and $n+m$ open actions, totaling $3(n+m)+2$ non-movement actions.

The mobile never moves to the left while carrying something and always carries a key while moving to the right. However, each key can be moved to the right at most w times. Hence, the mobile moves to the right at most $(n+1) \cdot w$ times. As its final destination is w units to the right of its start destination, we can deduce that it performs at most $n \cdot w$ movements to the left. The total number of horizontal movements is therefore bounded by $(2n+1)w = (2n+1)W + (2n+1)(2m+2n)$.

Vertical moves are only necessary before passing barriers ($n+m$ barriers times $2(n+m)+2$ planned paths), for accessing the correct location to be opened ($n+m$ times) and immediately before picking up a key or dropping the goal key ($n+m+2$ times). In each of these cases, no more than two moves upwards or downwards are needed. This leads to an upper bound on the total number of vertical movements equal to $2((n+m)(2(n+m)+2)+(n+m)+(n+m+2)) = 4(n+m)^2+8(n+m)+4$.

The total number of actions is thus no bigger than the sum of these three bounds, which is $3(n+m)+2+(2n+1)W+(2n+1)(2m+2n)+4(n+m)^2+8(n+m)+4 = (2n+2)W$, as required. □

This concludes our discussion of the GRID domain.

3.7 MICONIC-10

For the last transportation benchmark, the MICONIC-10 elevator domain, introduced by Köhler and Schuster [13], nomenclature is a bit complicated. The AIPS 2000 competition actually featured three different domains under that name, so it is more appropriate to speak of the MICONIC-10 domain *family*. However, we will shortly see that only the "full" MICONIC-10 domain needs in-depth discussion. The simplest variant, called MICONIC-10-STRIPS, defines tasks very similar to LOGISTICS with one mobile, or $\text{TRANSPORT}_{\infty\infty1}$ with complete graph roadmaps.

Definition 31 MICONIC-10-STRIPS *task*

A MICONIC-10-STRIPS ***task*** *is a* $\text{TRANSPORT}_{\infty\infty1}$ *task with a complete graph roadmap such that all portables are goal portables and for each portable, initial and goal location are different.*

The mobile of a MICONIC-10-STRIPS *task is called the* ***elevator****, the portables are called* ***passengers****, and the locations are called* ***floors****.*

The only difference between plans in the MICONIC-10-STRIPS domain and plans in TRANSPORT lies in the restriction that passengers can only be dropped off at their destination floor, not at other floors, and cannot be picked up by the elevator again, once they have been moved to their destination. This is formalized in the following domain definition.

Definition 32 MICONIC-10-STRIPS *domain*

The MICONIC-10-STRIPS *planning domain is equal to the restriction of the* TRANSPORT *domain to the set of* MICONIC-10-STRIPS *planning tasks, except for the following differences.*

For all passengers p, the action $pick_{m,p}$ is only defined in states where the current location of m equals the initial location of p.

For all passengers p, the action $drop_p$ is only defined in states where the current location of m equals the goal location of p.

This definition is not completely in accordance with the PDDL definition that was used in the AIPS 2000 competition: In the competition version, it is possible to pick up the same passenger multiple times at his initial location, even if he is already inside the elevator or at his goal location. In the latter case, he can also be dropped there another time. This is a harmless modeling flaw, which does not affect the following complexity result.

Theorem 33 *Complexity of* MICONIC-10-STRIPS *planning*

The plan existence problem for MICONIC-10-STRIPS *can be decided in polynomial time, and the bounded plan existence problem for* MICONIC-10-STRIPS *is* **NP***-complete.*

PROOF. *For plan existence, the algorithm from Theorem 13 can be used.*

For bounded plan existence, Theorem 18 applies. It is easy to verify that the minor differences between MICONIC-10-STRIPS *and* $\text{TRANSPORT}_{\infty\infty 1}$ *with complete graph roadmaps does not invalidate the proof of that theorem.* □

The real MICONIC-10 domain, however, is somewhat more complicated, introducing many additional constraints on elevator movement caused by special service requirements. MICONIC-10 tasks are defined as follows; we refer to the literature [13] for a motivation of the different features.

Definition 34 MICONIC-10 *task*

A MICONIC-10 ***task*** *is a 14-tuple* $(F, \preceq, f_0, P, l_0, l_G, A, P_V, P_N, P_D, P_S, P_A, P_1, P_2)$*, where*

- F *is a finite set of* ***floors*** *with a total order* $\preceq$ *and* ***initial elevator floor*** $f_0 \in F$*,*
- P *is a finite set of* ***passengers****,*
- $l_0, l_G : P \to F$ *are the* ***initial floor*** *and* ***destination floor*** *functions, respectively, satisfying* $l_0(p) \neq l_G(p)$ *for all passengers* p*,*
- $A \subseteq P \times F$ *is the* ***access relation****, where we say that passenger* p ***has access to*** *floor* f *if and only if* $(p, f) \in A$*, and*
- $P_V, P_N, P_D, P_S, P_A, P_1, P_2 \subseteq P$ *are the sets of* ***VIP****,* ***non-stop****,* ***direct travel****,* ***supervised****,* ***attendant****,* ***group one*** *and* ***group two*** *passengers, respectively.*

A MICONIC-10 *task is called* ***simple*** *if all passengers have access to all floors and all special passenger sets (*P_V*,* P_N*,* P_D*,* P_S*,* P_A*,* P_1*,* P_2*) are empty.*

Compared to the STRIPS variant, MICONIC-10 replaces individual pickup or drop actions for passengers by a single "stop" action which causes all passengers that have reached their destination floor to debark and all passengers waiting at that floor to enter automatically.

It also introduces passengers that impose movement restrictions on the elevator. The elevator may only stop at floors to which all passengers inside the elevator have access. VIP passengers must be served first, and if a non-stop passenger is inside the cabin, the elevator may only stop at his destination floor. If a direct travel passenger wants to travel up, the elevator must not move down, and vice versa. Supervised passengers may only be in the cabin while attendants are also present, and passengers from the two groups may not travel together. The following definition formalizes these constraints.

Definition 35 MICONIC-10 *domain*

The MICONIC-10 *planning domain maps* MICONIC-10 *tasks with passenger set* P *and floor set* F*, ordered by* $\preceq$*, to planning task state models as follows.*

The set of states consists of all triples (f_E, P_M, P_S)*, where* $f_E \in F$ *and* $P_M, P_S \subseteq P$*. The location of the elevator is denoted by* f_E*, called the* current floor*. A passenger is called* moving *(in the elevator) if he is in* P_M*,* served *if he is in* P_S *and* waiting *if he is neither moving nor served.*

The initial state is $(f_0, \emptyset, \emptyset)$*, where* f_0 *is the initial elevator floor. A state is a goal state if all passengers in* P *are served.*

The set of actions consists of a movement action $move_f$ *for each floor* f *and a single stop action.*

The movement action $move_f$ is defined in all states (f_E, P_M, P_S) such that $f_E \neq f$ and for all destination floors f_G of moving direct travel passengers, either $f_E \preceq f \preceq f_G$ or $f_G \preceq f \preceq f_E$. The resulting state is then (f, P_M, P_S).

If the stop action stop is defined in a state (f_E, P_M, P_S), then the resulting state is $(f_E, P_M \setminus P_{debark} \cup P_{enter}, P_S \cup P_{debark})$, where P_{debark} contains all moving passengers with destination floor f_E, and P_{enter} contains all waiting passengers with initial floor f_E.

The action is defined in all states satisfying the following conditions:

- ***Access:*** *All moving passengers have access to the current floor.*
- ***VIP:*** *The current floor is the initial floor of a waiting VIP passenger or the destination floor of a moving VIP passenger, or all VIP passengers are served.*
- ***Non-stop:*** *The current floor is the destination floor of a moving non-stop passenger, or there are no moving non-stop passengers.*
- ***Supervising:*** *If the* resulting state *contains a supervised moving passenger, then it also contains a moving attendant.*
- ***Groups:*** *The* resulting state *does not both contain moving passengers of groups one and two.*

The Miconic-10-Simple *domain is the restriction of* Miconic-10 *to the set of simple* Miconic-10 *tasks.*

This definition is in accordance with the original description of the Miconic-10 domain by Köhler and Schuster. It repairs two flaws of the PDDL definition, which does not implement the VIP service and non-stop travel constraints correctly. In the original definition, the elevator is allowed to travel to the initial or destination floor of a VIP passenger even if the passenger has boarded/debarked already and other VIP passengers should take priority. Additionally, the elevator cannot stop at any floor if several non-stop travel passengers with conflicting destinations have boarded. Both differences to the PDDL specification are harmless regarding our results, however. We will not make use of nonstop passengers in our proofs, and there will only be one VIP passenger per task, in which case the problem does not arise. Before proving results for the full Miconic-10 domain, we briefly discuss the simple variant.

Theorem 36 *Complexity of* Miconic-10-Simple *planning*

The plan existence problem for Miconic-10-Simple *can be decided in polynomial time, and the bounded plan existence problem for* Miconic-10-Simple *is* **NP***-complete.*

PROOF. *Plan existence is trivial, since plans always exist. A trivial plan*

that solves a MICONIC-10-SIMPLE *task stops at each floor in some arbitrary order, after which no more passengers are waiting, then stops at each floor a second time, at which point all passengers have been served.*

For bounded plan existence, note that this is equivalent to bounded parallel plan existence in $\text{TRANSPORT}_{\infty\infty 1}$ *with complete graphs for the variant of parallel planning where multiple portables can be picked up and dropped at the same time. As discussed in Section 3.4, this is an* **NP***-complete problem.* □

Of course, this result implies that bounded plan existence in the full MICONIC-10 domain is also **NP**-hard. However, the following result shows that the additional constraints do lead to different complexity results.

Note that the previous proof of hardness critically relies on the fact that the roadmap of a MICONIC-10 task is a complete graph, which is a somewhat unreasonable assumption. In order to take into account the real costs of moving elevators between distant floors, it would also be interesting to investigate a variant of the MICONIC-10 domain where floors are only connected to the floors *directly* above and below. For the following result these different ways of modelling do not make a difference, because in the full MICONIC-10 domain, plan existence is already **NP**-complete.

Theorem 37 *Complexity of* MICONIC-10 *planning*

The plan existence and bounded plan existence problems for MICONIC-10 *are* **NP***-complete, even if the only special passengers are attendants, supervised passengers and a single VIP passenger.*

PROOF. *Membership in* **NP** *for both problems follows from a polynomial plan length argument: The number of stop actions in a reasonable plan can be bounded by twice the total number of passengers because a stop action can be omitted if no passenger enters or leaves the elevator. The number of move actions can be bounded by the number of stop actions, one movement before each stop.*

Hardness for the bounded plan existence problem follows from the previous result. For plan existence, we reduce from the problem of finding a Hamiltonian path with a fixed start vertex v_1 *in a digraph* (V, A) *[11, Problem GT39].*

The corresponding MICONIC-10 *task has the following floors: the initial elevator floor* f_0*, a* final floor f_∞*, for each vertex* u *a* vertex start floor f_u *and* vertex end floor f_u^**, and for each arc* (u, v) *an* arc floor $f_{u,v}$*. F is the set of all these floors. For each vertex* u*,* F_u *is the set containing* f_u*,* f_u^* *and the*

arc floors for outgoing arcs of u. Because we do not make use of direct travel passengers, the total order on the floors is irrelevant.

These are the passengers to be served:

Passenger	*From*	*To*	*Access to...*	*Special*
p_0	f_0	f_{v_1}	$\{f_0, f_{v_1}\}$	*VIP, attendant*
$\forall u \in V$: p_u	f_0	f_u	$F \setminus \{f_\infty\}$	*supervised*
$\forall u \in V$: p_u^*	f_u	f_u^*	$F_u \cup \{f_\infty\}$	*attendant*
$\forall u \in V$: p_u^∞	f_u^*	f_∞	$F \setminus \{f_u\}$	*none*
$\forall (u,v) \in A$: $p_{u,v}$	$f_{u,v}$	f_v	$\{f_{u,v}, f_u^*, f_v\}$	*attendant*

Assume that it is possible to solve the task. Because p_0 is a VIP, the first stops must be at f_0 and f_{v_1}, picking up all the supervised passengers and $p_{v_1}^*$. Because of the access restrictions of that passenger, the journey can only proceed to floors from F_{v_1}, and $f_{v_1}^*$ is not an option because going there would lead to the only attendant leaving. Thus, the elevator must go to f_{v_1,v_2} (for some vertex v_2 that is adjacent to v_1) and can then only proceed to $f_{v_1}^*$ and then f_{v_2}, picking up $p_{v_1}^\infty$ and $p_{v_2}^*$.

We are now in a similar situation as upon arrival at f_{v_1}, and again, the elevator will eventually go to some floor f_{v_3}, then f_{v_4}, following the arcs of the digraph (V, A) in a path $[v_1, \ldots, v_n]$ until all vertices have been visited once. No vertex can be visited twice because of the access restrictions for passengers of type p_u^∞. So plan existence implies a Hamiltonian path starting at v_1 in the digraph.

On the other hand, the previous discussion shows that if a Hamiltonian path exists, there is a sequence of elevator movements and stops such that all supervised passengers have been served and the elevator is located at some floor f_u for $u \in V$. No longer requiring attendants, it can then immediately proceed to f_u^*, then f_∞ and finally serve the remaining passengers of type $f_{u,v}$ (for arcs (u,v) not part of the Hamiltonian path), one after the other, completing the plan. □

3.8 Summary

This concludes our analysis of transportation planning. For a fairly general family of transportation domains, we have shown **NP**-completeness of non-optimal planning in the restricted fuel case and **NP**-completeness of optimal planning in all cases. Just finding some plan in tasks where fuel is abundant

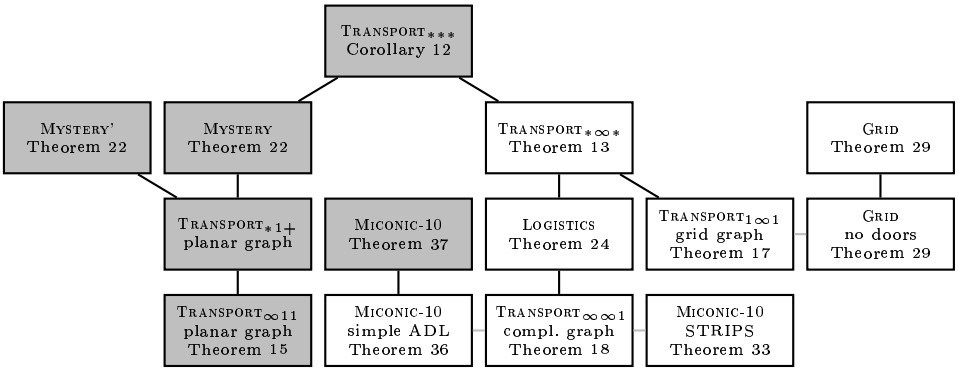

Fig. 9. Complexity of plan existence for transportation benchmarks.

was shown to be a polynomial problem in the TRANSPORT hierarchy.

These results extend to the transportation benchmarks from the AIPS competitions: Where fuel constraints were present, namely in the MYSTERY and MYSTERY' domains, both decision problems under investigation are **NP**-complete. Without fuel constraints, such as in the domains LOGISTICS, GRID, MICONIC-10-STRIPS and MICONIC-10-SIMPLE, non-optimal plans can be generated in polynomial time, but bounded plan existence is still **NP**-complete. Two domains exhibit slightly different behavior: In the full MICONIC-10 domain, despite unlimited fuel, plan existence is already **NP**-complete because of the variety of different constraints on the movement of the elevator, and in the GRIPPER domain, planning is so easy that even optimal plans can be generated in polynomial time by a trivial algorithm.

The results were obtained by exploiting the generalization/specialization hierarchy of the planning domains under investigation. The hierarchy is illustrated in Figure 9. Black lines indicate special cases, gray lines strong similarities of domains. Deciding plan existence is **NP**-complete for domains with gray boxes, and plans can be generated in polynomial time for domains in white boxes. The bounded plan existence problem is **NP**-complete for all domains in the figure; the trivial GRIPPER domain is not shown.

Some benchmarks are still **NP**-hard if some severe restrictions are made. For instance, in the GRID domain, no locked locations are needed for proving **NP**-hardness. All hard multi-agent benchmarks are already hard if there is only a single agent. Mainly for this reason, we could also show that for all transportation domains investigated, sequential and parallel planning are of identical complexity.

4 Non-transportation domains

In this section, we will discuss the remaining benchmark domains from the AIPS 1998 and AIPS 2000 planning competitions, namely MOVIE, ASSEMBLY, BLOCKSWORLD, SCHEDULE and FREECELL. For the first three, our discussion will be brief.

4.1 MOVIE, ASSEMBLY *and* BLOCKSWORLD

McDermott described the MOVIE domain as follows [8]:

> MOVIE: In this domain, the goal is always the same (to have lots of snacks to watch a movie). There are seven actions, including rewind-movie and get-chips, but the number of constants increases with the problem number.

The purpose of the MOVIE domain in the 1998 competition was to check if the performance of the competing planning systems degrades in the face of many objects in the task definitions. It was found that this was not the case, as all planners could quickly solve all MOVIE tasks. From a complexity point of view, this benchmark is not interesting. All tasks are solved by the same plan, and hence non-optimal, optimal sequential and optimal parallel planning are all constant time problems in this domain. Therefore, we restrict our discussion of MOVIE to the following remark.

Remark 38 *Complexity of* MOVIE *planning*

The plan existence and bounded plan existence problems for MOVIE *can be decided in polynomial time.*

The next benchmark under discussion, ASSEMBLY, was introduced by McDermott as follows [8]:

> ASSEMBLY: The goal is to assemble a complex object made of subassemblies. There are four actions: (1) commit resource assembly, (2) release resource assembly, (3) assemble part assembly, and (4) remove part assembly. The sequence of steps must obey a given partial order. In addition, through poor engineering design, many subassemblies must be installed temporarily in one assembly, then removed and given a permanent home in another.

A discussion of the complexity of ASSEMBLY faces some problems. First of all, although the domain was part of the ADL track of the 1998 competition, neither of the two participating ADL planning systems was able to solve a single competition task, so results we could provide for this domain are probably

less valuable because of the lack of empirical data for comparison.

More importantly, some of the competition tasks are specified in incorrect PDDL, making it impossible to feed them to current ADL-capable planners such as **FF** [14], which can solve all remaining ASSEMBLY tasks. Specifically, the competition tasks 7, 12, 13, 14, 19 and 27 are flawed.

Last and most importantly, there seem to be some severe errors in the PDDL definition of the domain itself, namely in the *remove* operator, which in its current state allows completely disassembling a composite object into its atomic parts without making it unavailable or incomplete. For example, consider the (impossible) task of creating a rectangular table and an oval table out of four legs, a rectangular board and an oval board. In the domain as specified for the competition, it is possible to build the rectangular table out of the board and the legs, remove the legs (which leaves the rectangular table in its *complete* state) and then use the legs and the oval board to build the oval table, creating two tables.

There is another minor oddity with regard to resource allocation, which allows to allocate resources such as a voltmeter to an object, incorporate this object into other objects while the resource stays allocated and then deallocate the resource although the object it has been allocated to is buried deep inside another object.

An in-depth treatment of the ASSEMBLY domain would require us to look both at the domain "as is", in its flawed state, to be able to judge the performance of planning systems that solve ASSEMBLY tasks, and to look at the corrected version. We have performed this analysis [7] and do not want to repeat the whole discussion here, as we consider it of little significance considering the amount of space it would require. It turns out that in the "corrected" ASSEMBLY domain, there is an infinite sequence of scaling tasks for which shortest plan lengths grow exponentially in the problem size, whereas plan lengths in the competition version are polynomially bounded and plans can be generated in polynomial time. We will quote the first result here and point at the reference for the proof [7].

Theorem 39 *Complexity of* ASSEMBLY *planning*

Shortest plan sizes in the (corrected) ASSEMBLY *domain can grow exponentially in the encoding length of the task, and hence optimal and non-optimal planning require exponential time.*

Of course, it is possible that the decision problems corresponding to optimal and non-optimal plan generation are polynomial-time problems despite the fact that plan lengths cannot be polynomially bounded. This is the case for the towers of Hanoi problem, which is not entirely unrelated to ASSEMBLY.

However, since our motivation for analyzing PLANEX and PLANLEN ultimately lies in obtaining insights into the complexity of plan generation, we consider such results less useful and do not analyze the ASSEMBLY domain further.

Instead, we turn to BLOCKSWORLD, probably *the* best-known planning domain. It was part of the AIPS 2000 planning competition and probably needs no further explanation. The complexity of BLOCKSWORLD planning has been investigated by a number of researchers. The following result [4] is well-known.

Theorem 40 *Complexity of* BLOCKSWORLD *planning*

The plan existence problem for BLOCKSWORLD *can be decided in polynomial time. The bounded plan existence problem for* BLOCKSWORLD *is* **NP***-complete.*

For a proof of this theorem, we refer to the work by Gupta and Nau [4]. Other interesting results for BLOCKSWORLD, including optimal planning algorithms and near-optimal planning algorithms that run in linear time have been detailed by Slaney and Thiébaux [6].

Considering the as of yet unaddressed problem of optimal *parallel* planning in the BLOCKSWORLD domain, where multiple blocks can be moved simultaneously as long as all movement involves different towers, it is not hard to see that plans with a minimal number of time steps can be found in polynomial time. It is sufficient to replace the choice point in the optimal planning algorithm by Slaney and Thiébaux with a planning step that performs all the considered moves simultaneously. Building on the existing results, it is easy to verify that this does not lead to multiple moves affecting the same tower, and that the resulting parallel plan is of minimal length.

4.2 SCHEDULE

We now turn our attention to the SCHEDULE domain from the AIPS 2000 competition, which can be described as follows:

> SCHEDULE: A set of physical objects must be processed by various machines to change their physical properties, such as color, shape and surface condition to match the goal requirements. The available equipment consists of a polisher, a roller, a lathe, a grinder, a punch, a drill press, a spray painter and an immersion painter.

In order to prove complexity results, we have to formalize this description. The following definition of SCHEDULE tasks is motivated by the PDDL definitions

used in the planning competition.

Definition 41 SCHEDULE *task*

The sets of SCHEDULE ***temperatures, surface conditions, shapes, colors*** *and* ***holes*** *and the set of* SCHEDULE ***object states*** *are defined as follows:*

$$O_{\mathrm{T}} = \{cold, hot\}$$
$$O_{\mathrm{SC}} = \{rough, smooth, polished, none\}$$
$$O_{\mathrm{S}} = \{cylindrical, circular, oblong\}$$
$$O_{\mathrm{C}} = \{blue, yellow, red, black, none\}$$
$$O_{\mathrm{H}} = \{front1, front2, front3, back1, back2, back3\}$$
$$O = O_{\mathrm{T}} \times O_{\mathrm{SC}} \times O_{\mathrm{S}} \times O_{\mathrm{C}} \times \mathbb{P}(O_{\mathrm{H}})$$

A SCHEDULE ***task*** *is a finite sequence of pairs from* $O \times \mathbb{P}(O)$.

According to this definition, an object in the SCHEDULE domain is characterized by its temperature, surface condition, shape, color and set of holes, for each of which there is a finite range of possible values. A SCHEDULE task specifies a sequence of such object states, describing the initial states of a set of objects, each coupled with a set of object states that specify possible goal states for these objects. In the SCHEDULE domain as used in the competition, there are some further restrictions on these sets. For example, specifications like "This object must become smooth and oblong" are allowed, but specifications like "This object must become red and hot or green and smooth" are not. By allowing more complex goal specifications, it seems that we make the planning problem harder, but we will shortly see that this does not affect our results. We can now define the SCHEDULE domain.

Definition 42 SCHEDULE *domain*

The set of SCHEDULE ***machinery*** *is defined as*
$M = \{roller, lathe, grinder, polisher, punch, drill, spray, immersion\}$.

Each machine $m \in M$ *has an associated* ***transformation*** t_{m}, *which is a partial function on object states. The exact definition of these transformations is not important to our analysis.*

The SCHEDULE *planning domain maps* SCHEDULE *tasks* $((i_1, G_1), \ldots, (i_n, G_n))$ *to planning task state models* $\mathcal{M}$ *as follows.*

The set of states is $O^n \times \mathbb{P}(M) \times \mathbb{P}(\{1, \ldots, n\})$. *For a state* $((o_1, \ldots, o_n), M_{\mathrm{B}}, O_{\mathrm{B}})$, o_i *is called the* current state of the i-th object, *a machine is said to be* busy

if it is in M_B, and the i-th object is said to be busy *if $i \in O_B$.*

The initial state is $((i_1, \ldots, i_n), \emptyset, \emptyset)$, and the set of goal states consists of those states where for all i, the current state of the i-th object is in G_i.

The set of actions is $\{ process_{m,i} \mid m \in M, i \in \{1, \ldots, n\} \} \cup \{timestep\}$.

The action $process_{m,i}$ is defined in all states $((o_1, \ldots, o_n), M_B, O_B)$ such that t_m is defined for o_i and neither m nor the i-th object are busy. The resulting state is then $((o_1, \ldots, o_{i-1}, t_m(o_i), o_{i+1}, \ldots, o_n), M_B \cup \{m\}, O_B \cup \{i\})$.

The action timestep is defined in all states $((o_1, \ldots, o_n), M_B, O_B)$ where at least one machine is busy and leads to state $((o_1, \ldots, o_n), \emptyset, \emptyset)$.

Plans in the SCHEDULE domain naturally fall into parts, separated by *timestep* actions. During each of these parts, each machine can only be used at most once, and hence each part consists of at most $|M| = 8$ actions. In this domain, it is more natural to optimize the number of *parts* than the number of actions. Therefore, our analysis addresses both optimization criteria.

Theorem 43 *Complexity of* SCHEDULE *planning*

The plan existence and bounded plan existence problems for SCHEDULE *can be decided in polynomial time. The latter result also holds if only the number of timestep actions, not the total number of actions, is taken into account for calculating plan length.*

PROOF. *We only prove the bounded plan existence results, which imply the plan existence result. To do so, we first devise an optimal planning algorithm for the "usual" bounded plan existence case and then show how it can be modified if only the number of timestep actions shall be counted.*

First, observe that objects with the same current state and goal specification need not be distinguished. As there is only a fixed number K of possible combinations of current state and goal specification for an object, a SCHEDULE *state (apart from the sets of busy machines and objects) can be described in an abstract way by a K-tuple of natural numbers, specifying* how many *objects of each kind are present. This will be called an* abstract state *of the task. Abstract states where each entry greater than zero relates to a state/goal pair where the state matches the goal description are called* abstract goal states.

The same applies to process actions: Rather than specifying that the i-th object is being processed by machine m, it suffices to say that some object *which matches the current state and goal description of the i-th object is being transformed by m. Thus, the process actions can be reformulated to work on ab-*

stract states. Converting an abstract plan of that kind into an actual plan is sufficiently easy to not require further discussion.

To avoid having to worry about busy machines and objects, the abstract actions can be replaced by abstract macro actions*, sequences of abstract actions containing exactly one timestep action, which is the last one in the sequence. Each plan can be partitioned into macro actions of that kind, assuming that it ends in a timestep action (which does not make sense for shortest plans, but we can require this property and then remove the last action after the plan has been generated). The advantage of this view is that before and after each abstract macro action, no machine and no object are busy.*

Because there are only eight machines, each abstract macro action can consist of no more than nine actions, including the terminating timestep. Thus, the number of possible macro actions is constant.

What is gained by recasting the problem in that way? As was said before, abstract states can be represented by a K-tuple of natural numbers, where $K = |O \times \mathbb{P}(O)|$ is fixed, and each natural number in this tuple can be bounded by the total number of objects in the task, which is its encoding length. Thus, the number of abstract states is polynomial *in the encoding length.*

This means that explicit graph-search techniques can be used to find a shortest plan consisting of abstract macro actions. Because different macro actions can comprise a different number of actions (between one and nine), the corresponding arcs must be weighted by that number. Still, the one-to-all shortest path problem in a weighted digraph can be solved in polynomial time, and it is then easy to pick the abstract goal state with shortest weighted distance from the abstract initial state (if any reachable goal state exists), extract the abstract plan, expand the macro actions and assign actual objects to the abstract actions to compute an optimal sequential plan in polynomial time.

If only timestep actions are to be counted, a non-weighted digraph should be used, because then all macro actions have a uniform cost of one. □

The polynomial time property of the algorithm critically relies on the fact that the number of machines and object states is constant. If the object states, machines and transformations are given as part of the individual planning tasks, then the bounded plan existence becomes **NP**-complete, at least in the more realistic case where only *timestep* actions are counted. This is already true if there are only three machines [11, Problem SS18]. Of course, the plan existence problem stays easy because objects can still be dealt with one at a time, by searching the transition graph that is defined by the set of object transformations.

Note that the execution time of the described algorithm, depending on the graph search algorithm used, grows at least as quickly as n^K, where n is the input size and $K = |O \times \mathbb{P}(O)| = 7680 \cdot 2^{7680}$, which means that this is not a practical algorithm. As was mentioned earlier, the real AIPS domain does not allow for arbitrary goal descriptions, which reduces the 2^{7680} factor to 80. Still, an $O(n^{7680 \cdot 80}) = O(n^{614400})$ algorithm is not tractable in practice. Although further optimizations can be used to decrease the complexity significantly, it is not obvious what a really tractable algorithm could look like.

4.3 FreeCell

The final benchmark we want to discuss is the FreeCell domain, based on the popular solitaire card game. The original card game can be described as follows:

> FreeCell: The game is played with a standard deck of 52 cards, initially arranged into eight *tableau piles* of six or seven cards each. Cards can be moved between these eight tableau piles, four *free cells* and four *foundation piles* according to the following rules:
> - Cards may only be picked up if they occupy a free cell or if they are the top card of a tableau pile. No more than one card can be picked up at the same time.
> - Cards may only be dropped in a free cell if it does not currently hold any other card.
> - Cards may only be added to a tableau pile (as its new top card) if that pile is empty, or the value of the card is one less than the value of the top card of the pile and it is of a different color (e. g., the four of spades can only be added to tableau piles with the five of diamonds or hearts as their top card).
> - Aces may be added to an empty foundation pile. Other cards may only be added to a foundation pile if their value is one higher than the value of the top card of the pile and they are of the same suit.
>
> The objective of the game is to move all cards to foundations.

Using a fixed deck size, the standard FreeCell game only allows for a constant number of different initial configurations. This is not very interesting from a complexity theory point of view, because problems with a finite number of instances can trivially be decided in polynomial time. Thus, it is necessary to allow varying deck sizes, either by adding new suits or by adding cards to the existing suits.

Of these, we choose the latter, because it seems the more natural choice. In fact, the AIPS 2000 competition tasks use the same scaling parameter,

although for these, deck sizes never exceed 52 cards and in most cases, less cards are used. We also allow for a varying number of tableau piles and free cells, but we will see that our hardness results already hold if the number of free cells is a fixed constant.

Definition 44 FreeCell *task*

For a natural number $n \in \mathbb{N}$, $C_n = \{\diamondsuit, \heartsuit, \clubsuit, \spadesuit\} \times \{1, \ldots, n\}$ is called the ***n-deck****. Its elements are called* ***cards****. For a card (s, v), s is called its* ***suit*** *and v is called its* ***value****. A card is called* ***red*** *if its suit is $\diamondsuit$ or $\heartsuit$, and* ***black*** *otherwise.*

An ***n-w-tableau*** *is a set of at most w non-empty sequences over C_n such that each card appears in at most one such sequence. The individual sequences are called* ***tableau piles****. The last card of the sequence is called its* ***top*** *card, the subsequence that is obtained by removing the top card is called the* ***buried part*** *of the pile. A card c* ***matches*** *a tableau pile if and only if c and the top card of the pile are of different color and the value of c is one less than the value of the top card.*

A FreeCell ***task*** *is a 4-tuple (n, w, c, T), where*

- *$n \in \mathbb{N}$ is called the* ***suit length****,*
- *$w \in \mathbb{N}$ is called the* ***tableau width****,*
- *$c \in \mathbb{N}$ is called the* ***free cell count****, and*
- *T is an n-w-tableau such that each card in C_n appears in exactly one tableau pile. It is called the* ***initial tableau****.*

To readers acquainted with FreeCell, the previous definition should go without much explanation. One thing we would like to point out, however, is that FreeCell tasks as defined above do not necessarily contain tableau piles of (roughly) *equal size*, as is usually assumed. This is not an overgeneralization, because the height of tableau piles can be equalized by adding additional cards of lowest value to the smaller piles. These will be moved to foundations immediately in any reasonable plan, resulting in the original uneven tableau. Moves of this kind are never bad and are hence performed automatically by many FreeCell computer programs.

We can now formally define the semantics of legal FreeCell moves by defining the domain.

Definition 45 FreeCell *domain*

The FreeCell *planning domain maps* FreeCell *tasks of suit length n and tableau width w to planning task state models as follows.*

The set of states consists of all pairs (T, F) such that T is an n-w-tableau and F is a set of cards which are not part of any tableau pile such that $|F|$ is bounded by the free cell count of the task. T and F are called the current tableau *and* free cell cards*, respectively. In any state, it is assumed that cards that are neither in T nor in F have been moved to foundations.*

In the initial state, the current tableau is the initial tableau and the set of free cell cards is empty. There is a single goal state, where both the current tableau and set of free cell cards are empty.

To define the set of actions, we introduce pickup *and* drop activities*, which are partial functions on the state set that are not themselves actions of the planing task state model but helpful for defining them.*

There are three kinds of pickup activities for a card c. The first, pickup from free cell*, is defined in all states (T, F) in which c is a free cell card. It maps to the state $(T, F \setminus \{c\})$. The second,* pickup from tableau*, is defined in all states (T, F) where c is the* only *card of some tableau pile $p \in T$ and maps to $(T \setminus \{p\}, F)$. The last,* pickup from tableau pile*, is defined in all states (T, F) containing some tableau pile p with top card c and non-empty buried part p' and maps to $(T \setminus \{p\} \cup \{p'\}, F)$.*

Additionally, there are four kinds of drop activities for card c. Drop in free cell *and* drop on tableau *are the inverse functions of the first two kinds of pickup activities just described. Note that the definition of the state space ensures that the maximum number of free cells and tableau piles is never exceeded. The third,* drop on tableau pile *is the inverse function of* drop from tableau pile*, but with the additional restriction that the dropped card must match the pile it is dropped on. The last,* drop at foundations*, is defined in states where no tableau or free cell card of the same suit as c has a lower value. Dropping at foundations leaves tableau and free cell cards unchanged.*

Having defined pickup and drop activities, the set of actions of the planning task state models consists of all compositions drop$\circ$pick, where pick is a pickup activity and drop is a drop activity for the same card.

The following theorem contains our last technical result.

Theorem 46 *Complexity of* FreeCell *planning*

The plan existence and bounded plan existence problems for FreeCell *are* **NP***-complete, even for an arbitrary fixed number of free cells.*

PROOF. *Membership in* **NP** *for both problems follows from a polynomial plan length argument. This part of the proof is quite technical and we will not*

provide it in full detail.

Note that the only kinds of actions that cannot be undone immediately (by applying an inverse action) are movements to foundations and movements of cards from tableau piles where the top card does not match the buried part of the pile.

Cards in foundations are out of play, so the number of actions of the first kind in any plan is bounded by the total number of cards. As no new mismatch in tableau piles is ever introduced in the course of plan execution, the number of actions of the second kind is bounded by the number of mismatches in the initial tableau.

Hence, there is a polynomial bound on the number of non-undoable movements, and it suffices to argue that in between *non-undoable movements, the number of actions in shortest plans are polynomially bounded. This part of the proof is spelled out in the reference [7].*

For hardness, we only need to discuss the plan existence problem; bounded plan existence follows.

The proof is by reduction from 3SAT. *Let* (V, C) *be a* 3SAT *instance, where* $V = \{v_1, \ldots, v_n\}$ *is a set of variables and* C *is a set of clauses over* V *containing exactly three literals each. We write* $l_{i,j}$ *for the* j*-th literal of the* i*-th clause. The corresponding* FreeCell *task is rather intricate and we encourage the reader to look at Figure 10 during the following discussion to get some intuition of how propositional formula and planning task are interrelated.*

We need some ordering for the literals over V*, so we call* v_i *the* $(2i-1)$-th literal *and* $\neg v_i$ *the* $2i-th$ *literal and write* l_k *for the* k*-th literal. We define the* number of occurrences *of* l_k *as the number of pairs* (i, j) *such that* $l_{i,j} = l_k$*, and the corresponding pairs are called the* first, second, ... occurrence *of* l_k*. In which order the occurrences are numbered is of no importance. Additionally, we define* o_k*, the* cumulated number of occurrences *up to* l_k*, as the sum of the number of occurrences of* $l_{k'}$ *for all* $k' \leq k$*.*

Furthermore, we define the selection value val_{S} *as* $|C| + 2n + 2$*, the* literal value *of* l_k *as* $val_k = val_{\mathrm{S}} + 2k + 2o_k$*, the* clause value val_{C} *as* $val_{2n} + 2$ *and the* bottom value val_{B} *as* $val_{\mathrm{C}} + 6|C|$*. In the example of Figure 10,* $val_{\mathrm{S}} = 11$*,* $val_1 = 15$ *(for literal* v_1*),* $val_2 = 21$ *(for* $\neg v_1$*),* $val_3 = 27$ *(for* v_2*),* $val_4 = 31$ *(for* $\neg v_2$*),* $val_5 = 37$ *(for* v_2*),* $val_6 = 41$ *(for* $\neg v_3$*),* $val_{\mathrm{C}} = 43$ *and* $val_{\mathrm{B}} = 61$*.*

The FreeCell *task has a suit length of* $val_{\mathrm{B}} + 4|C| - 2$*, a tableau width of* $6|C| + 2|V| + 2$ *and a free cell count of 0. The initial tableau is arranged as follows.*

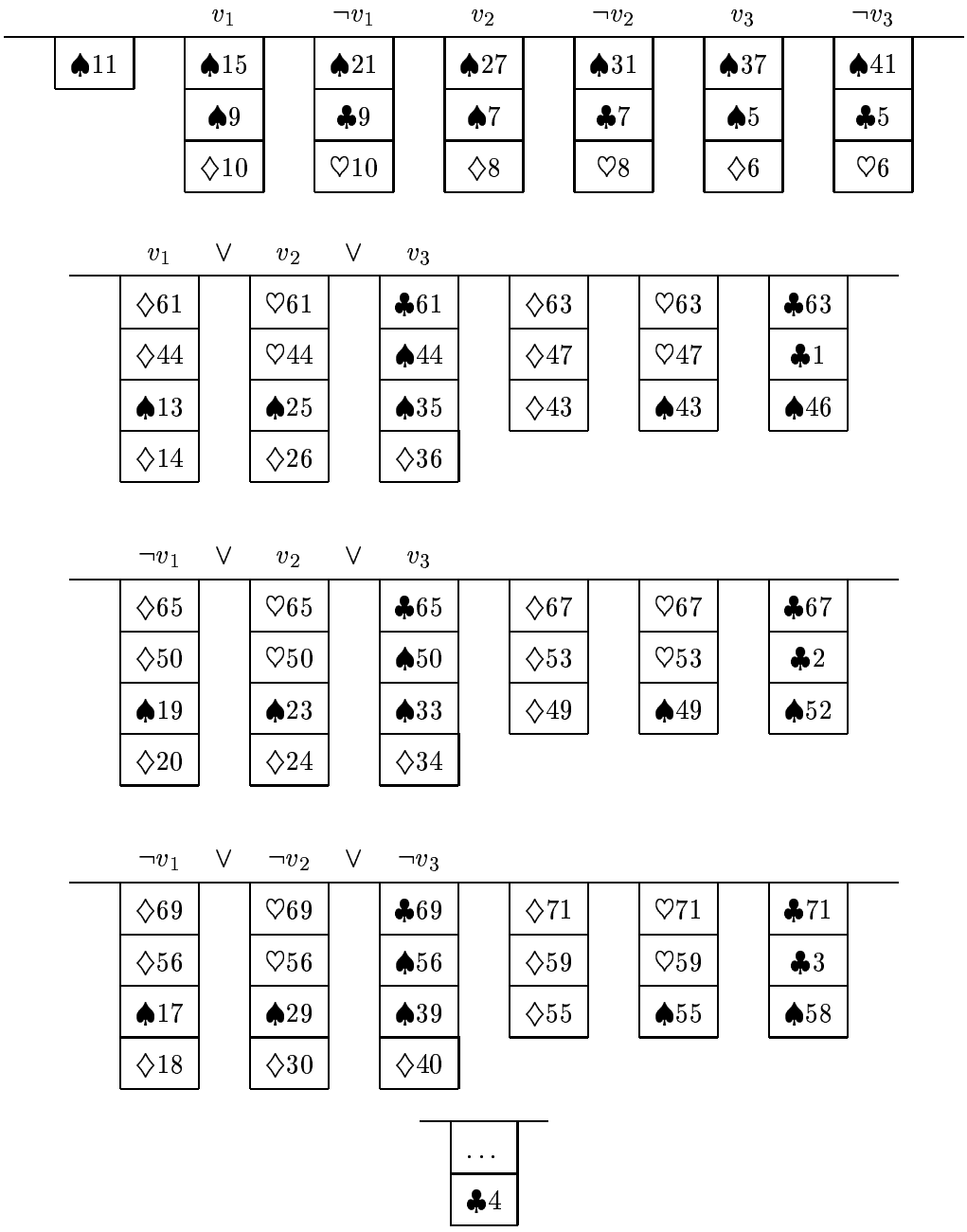

Fig. 10. FreeCell task corresponding to $(v_1 \vee v_2 \vee v_3) \wedge (\neg v_1 \vee v_2 \vee v_3) \wedge (\neg v_1 \vee \neg v_2 \vee \neg v_3)$.

The piles of the initial tableau, depicted in Figure 10, fall into three groups. The first $2|V| + 1$ *piles are called the* literal selection piles*, depicted at the top of the figure. The next* $6|C|$ *piles are called the* clause piles*, organized into subgroups of six piles that each relate to a specific clause, called* clause groups*. The last pile, holding most of the cards, is called the* big pile*, at the bottom of the figure. Note that cards at the* top *of a pile are shown near the* bottom *of the picture, following the usual convention of* FreeCell *implementations on computers.*

Literal selection piles: *The first of these only contains the card* $(\spadesuit, val_{\mathrm{S}})$. *The other piles contain three cards each. Each of them corresponds to a literal, the pile for* l_k *being defined as* $(\spadesuit, val_k)(\spadesuit, val_{\mathrm{S}} - k - 1)(\diamondsuit, val_{\mathrm{S}} - k)$ *if* k *is odd and as* $(\spadesuit, val_k)(\clubsuit, val_{\mathrm{S}} - k)(\heartsuit, val_{\mathrm{S}} - k + 1)$ *if* k *is even.*

Clause groups: *Each group is organized as follows. There are six piles corresponding to the* i*-th clause. We set the* bottom *value for the group as* $bottom = val_{\mathrm{B}} + 4(i-1)$ *and the* base *value for the group as* $base = val_{\mathrm{C}} + 6(i-1)$.

The first three piles contain four cards each. The first and second of these are of value bottom and base$+1$*, respectively, suit does not matter. The remaining cards depend on the literals in the clause: For* $1 \leq j \leq 3$*, the third and fourth cards of the* j*-th pile are* $(\spadesuit, val_k - 2m)$ *and* $(\diamondsuit, val_k - 2m + 1)$*, where* k *and* m *are calculated such that* (i, j) *is the* m*-th occurrence of* l_k.

The other three piles contain three cards each.
The fourth pile is defined as $(\diamondsuit, bottom + 2)(\diamondsuit, base + 4)(\diamondsuit, base)$.
The fifth pile is defined as $(\heartsuit, bottom + 2)(\heartsuit, base + 4)(\spadesuit, base)$.
The sixth pile is defined as $(\clubsuit, bottom + 2)(\clubsuit, i)(\spadesuit, base + 3)$.

Big pile: *The top card of this pile is* $(\clubsuit, |C| + 1)$*, and below this are all remaining cards, ordered such that cards of lower value are closer to the top.*

We now show that this FreeCell *task can be solved if and only if there is a satisfying assignment to the variables of the logical formula. First assume there is such a satisfying assignment* $\alpha : V \to \{\top, \bot\}$*. The following strategy solves the task:*

For each $i \in \{1, \ldots, n\}$*, move the top two cards from the literal selection piles that correspond to literals which are true under* α *to the first literal selection pile. This releases the bottom cards of some literal selection piles, spades cards which can then be used to move cards from the clause piles. In the example of Figure 10, for the assignment* $\{(v_1, \top), (v_2, \top), (v_3, \bot)\}$ *these are the 15, 27 and 41 of spades. These are called the* literal choice cards.

The first three piles of each clause group relate to the literals in that clause. The top two cards of such a pile can be moved to the literal selection piles if and only if the literal choice card of the corresponding literal has been revealed.

Because we have a satisfying truth assignment, a literal is satisfied in each clause, and thus it is possible to remove the top two cards of one of the first three piles of each clause group. If the new top card is black, the top card of the fourth pile of the clause group can be moved on top of it; if it is red, the top card of the fifth pile can be moved. Thus a red card is revealed in the fourth or fifth pile, and the top card of the sixth pile can be moved on top of that.

After this has been done for all clauses, the first $|C|$ cards of clubs are available in the sixth piles of the clause groups and can be moved to foundations, allowing to move the top card of the big pile to foundations. This reveals many low-valued cards, and it is not hard to see that all cards of values up to val_S can be moved to foundations immediately. This reveals the literal choice cards of all literals that are false under the chosen assignment, allowing to move the top two cards of the clause group piles relating to unsatisfied literals to the literal selection piles as well.

After this has been done, all piles are ordered by value, with cards of lower value closer to the top, allowing to move all remaining cards to foundations, solving the task.

Now assume that the FreeCell *task is solvable. It is not possible to move the bottom card of any tableau pile within the tableau before the top card of the big pile is moved, because all cards that they could be moved on top of are buried in the big pile. Before the top card of the big pile is moved, it is not possible to move the bottom card of any pile to foundations either. This implies that the first movement of the top card of the big pile cannot go to an empty tableau position.*

On the other hand, it can not be moved on top of any other card as its first movement, because all possible destination cards are buried under it. Together, this implies that its first (and thus only) movement must be directly to foundations.

This in turn requires all lower-valued clubs cards to be moved to foundations first, requiring movements within the clause piles. For each clause group, the top card of the sixth pile must be moved, and it can only be moved to the second card of the fourth or fifth pile, requiring the top card of either of these piles to be moved. These in turn can only be moved on top of the second (counting from the bottom) card from any of the first three piles of that clause group. Thus, in each clause group, the top two cards of one of the first three piles must be moved somewhere else for the task to be solvable.

The only way this can be done is by uncovering the literal choice cards of corresponding literals in the way explained in the other direction of the proof. As it is not possible to uncover the literal choice card for v_i and $\neg v_i$ at the same time (for any i), this requires the existence of a satisfying assignments to the truth variables, completing the proof. □

If there are more than zero free cells, a very similar reduction can be used by ensuring that all free cells must become occupied right at the start of any plan and cannot be cleared before the top card of the big pile is moved to foundations. Again, we refer to the literature for details [7].

This result concludes our discussion of FreeCell. As the proof shows, the hardness of planning in this domain is not (or at least not exclusively) due to the difficulty in allocating free cells or empty tableau positions, but rather due to the choice of *which* card to move on top of a tableau pile when there are two possible choices.

5 Discussion

In this article, we analyzed the computational complexity of deciding plan existence and bounded plan existence for the planning domains used in the AIPS 1998 and 2000 competitions.

We discovered that the majority of benchmarks fits into a hierarchy of *transportation* domains, which we introduced and analyzed. One of our intentions in looking at a hierarchy of planning domains rather than isolated decision problems was to find the boundary between easy and hard problems within the general domains. For Transport, this boundary is clearly defined: For domains with restricted fuel, it is hard to generate plans, whereas for domains with unlimited fuel, solving this task is easy. One possible explanation why fuel restrictions make these problems harder is that, by bounding the number of movements that can be performed, they effectively limit the length of the generated plans, and hence the problem of just finding *any* plan becomes akin to the problem of finding an *optimal* plan.

This latter problem is hard to solve for all but the most trivial transportation domains we investigated (i. e., for all except Gripper). The main source of hardness that we could identify is a difficulty in *ordering*: While it is usually evident how individual portables should be transported to their destinations, the interactions between different portables make optimally solving the overall task a hard problem.

The Miconic-10 elevator domain, which is the only transportation domain we analyzed where fuel is not restricted and yet generating plans is hard, shows how side effects of actions can make a greedy planning strategy inappropriate. Specifically, the idea of picking up one person after the other and moving them to their destination floor does not work because it is not possible to keep other people (who we do not intend to move to their destination yet) from boarding the elevator, and their boarding can render the task of moving the other passengers to their destinations more difficult or impossible because of access restrictions or required attendance.

For three of the non-transportation benchmarks, we did not go into detail. Planning in Movie is trivial, planning in Blocksworld is a well-studied

problem, and the ASSEMBLY domain contains several flaws. In the two remaining planning domains, we found no difference in complexity between optimal and non-optimal planning. In SCHEDULE, both problems can be solved in polynomial time, and in FREECELL, the corresponding decision problems are both **NP**-complete. However, it should be noted that for SCHEDULE, our optimal planning algorithm is of very high complexity (though still polynomial), whereas it is not hard to devise linear-time algorithms for non-optimal SCHEDULE planning.

It is interesting to observe that in many benchmark domains, there is a complexity gap between plan existence and bounded plan existence. This is a point which must not be neglected when evaluating planning algorithms. Optimal and non-optimal planning systems cannot be easily compared to one another in terms of performance in a meaningful way, because they are solving different problems. While this fact is by no means new, it is interesting to note that it actually applies to many of the benchmark domains that are routinely used for evaluating planning systems. There has been significant recent progress on non-optimal planning, but in our opinion, optimal planners tend to get less attention than they deserve, maybe due to the fact that they are often compared to their non-optimal counterparts in terms of the size of problems they can handle. This kind of comparison is hardly fair.

The higher complexity of optimal planning helps explain why planning systems based on local search, most notably the **FF** system, have lately performed so much better than ones based on **Graphplan** or satisfiability techniques. Because our hardness proofs carry over to bounded parallel plan existence, they imply that in these domains planners like **Graphplan** [12] or **IPP** [15] try to solve provably hard subproblems that local search planners do not have to care about. When optimal plans are not required, local search has a conceptual advantage, and we cannot hope for similar performance from any planner striving for optimality.

A common criticism of complexity results is that they do not apply to "real" problem instances because these typically do not exhibit the arcane features of the instances used for reductions. Addressing this point, it is worth pointing out that our plan existence results coincide well with results by Hoffmann, who has investigated the search space topologies of the various benchmark domains [16].

Using a heuristic similar to the one of **FF** as the foundation of the analysis, Hoffmann defined a taxonomy of planning domains, distinguishing different kinds of dead-ends and local minima in the search space. The set of domains for which we proved plan existence to be **NP**-complete is precisely the one that forms the hardest category under Hoffmann's taxonomy ("unrecognized dead ends", top right in Figure 4 of the reference). Because Hoffmann's re-

sults were obtained in an empirical analysis, investigating randomly generated planning tasks, this shows that hard planning tasks in these domains do occur in practice.

A last result we want to point out is that all discussed decision problems are in **NP**, although **PSPACE**-complete planning domains can be expressed in the underlying representation language. While it might be interesting to have a greater variety in the hardness of benchmarks, it is evident that membership in **NP** is guaranteed as soon as there are polynomial bounds on plan lengths, which is a reasonable requirement from a plan execution point of view.

With this remark, we want to conclude our discussion of results. Of course, open issues remain. For the **NP**-complete decision problems, it would be good to be able to characterize "hard" tasks by identifying a *phase transition* between (usually easy) under-constrained and (usually easy) over-constrained tasks. This would greatly increase the benefit of those domains for benchmarking purposes.

As another issue, the distinction between non-optimal and optimal planning is quite coarse, and for those domains which exhibit differences in complexity for these problems, the question arises if it is possible to find *good* (if not optimal) plans in polynomial time, for example plans that are guaranteed not to exceed the length of optimal ones by more than a constant factor. It is evident that such performance guarantees are not hard to give in LOGISTICS or BLOCKSWORLD, but what about GRID?

We close with the following brief summary of results, ordered alphabetically by domain name.

- ASSEMBLY: The domain definition has several flaws. In the corrected version, shortest plan sizes can grow exponentially with the input size and hence plan generation needs exponential time.
- BLOCKSWORLD: Plans can be generated in low-order polynomial time. Bounded plan existence is **NP**-complete.
- FREECELL: Plan existence and bounded plan existence are **NP**-complete, even if the number of free cells is fixed to an arbitrary natural number.
- GRID: Plans can be generated in low-order polynomial time. Bounded plan existence is **NP**-complete, even if there are no locked locations, or if there is only one key to be moved to some goal location.
- GRIPPER: Optimal plans can be generated in low-order polynomial time.
- LOGISTICS: Plans can be generated in low-order polynomial time. Bounded plan existence is **NP**-complete, even if there is only a single truck and no airplane, or vice versa.
- MICONIC-10: In the two easier versions, plans can be generated in low-order polynomial time, and bounded plan existence is **NP**-complete. In the full

version, plan existence and bounded plan existence are **NP**-complete, even if the only special passengers are attendants, supervised passengers and a single VIP.
- MOVIE: Optimal plans can be generated in low-order polynomial time.
- MYSTERY and MYSTERY': Plan existence and bounded plan existence are **NP**-complete, even if there is only a single mobile, there are no capacity constraints, and there is exactly one unit of fuel at each location.
- SCHEDULE: Optimal plans can be generated in polynomial time. However, the running time of our algorithm is $O(n^{614400})$.

References

[1] T. Bylander, The computational complexity of propositional STRIPS planning, Artificial Intelligence 69 (1–2) (1994) 165–204.

[2] K. Erol, D. S. Nau, V. S. Subrahmanian, Complexity, decidability and undecidability results for domain-independent planning., Artificial Intelligence 76 (1–2) (1995) 65–88.

[3] C. Bäckström, B. Nebel, Complexity results for SAS^+ planning, Computational Intelligence 11 (4) (1995) 625–655.

[4] N. Gupta, D. S. Nau, On the complexity of blocks-world planning, Artificial Intelligence 56 (2–3) (1992) 223–254.

[5] B. Selman, Near-optimal plans, tractability, and reactivity, in: J. Doyle, E. Sandewall, P. Torasso (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR'94), Morgan Kaufmann, 1994, pp. 521–529.

[6] J. Slaney, S. Thiébaux, Blocks world revisited, Artificial Intelligence 125 (2001) 119–153.

[7] M. Helmert, On the complexity of planning in transportation and manipulation domains, Master's thesis, Albert-Ludwigs-Universität Freiburg, postscript available at http://www.informatik.uni-freiburg.de/~ki/theses.html (2001).

[8] D. McDermott, The 1998 AI Planning Systems competition, AI Magazine 21 (2) (2000) 35–55.

[9] F. Bacchus, The AIPS'00 planning competition, AI Magazine 22 (3) (2001) 47–56.

[10] D. Long, M. Fox, Automatic synthesis and use of generic types in planning, in: Chien et al. [17], pp. 196–205.

[11] M. R. Garey, D. S. Johnson, Computers and Intractability — A Guide to the Theory of NP-Completeness, Freeman, 1979.

[12] A. Blum, M. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1–2) (1997) 281–300.

[13] J. Köhler, K. Schuster, Elevator control as a planning problem, in: Chien et al. [17], pp. 331–338.

[14] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, Journal of Artificial Intelligence Research 14 (2001) 253–302.

[15] J. Köhler, B. Nebel, J. Hoffmann, Y. Dimopoulos, Extending planning graphs to an ADL subset, in: S. Steel, R. Alami (Eds.), Recent Advances in AI Planning. 4th European Conference on Planning (ECP'97), Vol. 1348 of Lecture Notes in Artifical Intelligence, Springer-Verlag, New York, 1997, pp. 273–285.

[16] J. Hoffmann, Local search topology in planning benchmarks: An empirical analysis, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01), 2001.

[17] S. Chien, S. Kambhampati, C. A. Knoblock (Eds.), Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000), Breckenridge, 2000.