Local Search Topology in Planning Benchmarks: A Theoretical Analysis

Jörg Hoffmann

Institute for Computer Science
Albert Ludwigs University
Georges-Köhler-Allee, Geb. 52
79110 Freiburg, Germany
hoffmann@informatik.uni-freiburg.de

Abstract. Many state-of-the-art heuristic planners derive their heuristic function by relaxing the planning task at hand, where the relaxation is to assume that all delete lists are empty. The success of such planners on many of the current benchmarks suggests that in those task's state spaces relaxed goal distances yield a heuristic function of high quality. Recent work has revealed empirical evidence confirming this intuition, stating several hypotheses about the local search topology of the current benchmarks, concerning the non-existence of dead ends and of local minima, as well as a limited maximal distance to exits on benches.
Investigating a large range of planning domains, we prove that the above hypotheses do in fact hold true for the majority of the current benchmarks. This explains the recent success of heuristic planners. Specifically, it follows that FF's search algorithm, using an idealized heuristic function, is a polynomial solving mechanism in (at least) eight commonly used benchmark domains. Our proof methods shed light on what the structural reasons are behind the topological phenomena, giving hints on how these phenomena might be automatically recognizable.

Technical Report No. 165

Report Date November 16, 2001

1 Introduction

In the last three years, planning systems based on the idea of heuristic search have been very successful. At the AIPS-1998 planning systems competition, HSP1 compared well with the other systems [17], and at the AIPS-2000 competition, out of five awarded fully automatic planners, FF and HSP2 were based

on heuristic search, while another two, Mips and STAN, were hybrids that incorporated, amongst other things, heuristic search [2].

Interestingly, four of these five planners use the same base approach for deriving their heuristic functions: they relax the planning task description by ignoring all delete lists, and estimate, to each search state, the length of an optimal relaxed solution to that state. This general idea has first been proposed by Bonet et al. [4]. The length of an optimal relaxed solution would yield an admissible heuristic. However, as was proven by Bylander [5], computing the optimal relaxed solution length is still NP-hard. Therefore, Bonet et al. introduced a technique for approximating optimal relaxed solution length, which they use in both versions of HSP [3]. The heuristic engines in FF [14] and Mips [6] use different approximation techniques.

Three of the above planners, HSP1, FF, and Mips, use their heuristic estimates in variations of local search algorithms, where the search space to a task is the state space, i.e., the space of all states that are reachable from the initial state. Now, the behavior of local search depends crucially on the topology of the search space (as has been studied in the SAT community, for example by Frank et al. [8]). Thus, the success of these heuristic planners on many planning tasks suggests that in those task's state spaces relaxed goal distances yield a heuristic function of high quality. Recent work has revealed empirical evidence confirming this intuition. Computing the optimal relaxed solution length to reachable states in small planning tasks from the competition domains, and measuring parameters of the resulting local search topology, Hoffmann [13] found the following. In the majority of the domains, the investigated instances did not contain any dead ends (states from which the goal is unreachable), and neither did they contain any local minima (regions of the state space where all neighbours look worse). Sometimes all instances in a domain had the same constant maximal exit distance (roughly, the maximal distance to a state with better evaluation). Hoffmann hypothesizes that his observations on the *small* example instances carry over directly to *all* instances in the respective domains.

In the presented investigation, we consider 20 often used planning benchmark domains, including all competition examples. Based on formal definitions of the domains, we prove that the majority do in fact have the aforementioned topological properties, confirming all of Hoffmann's hypotheses except one. The results give a strong argument for interpreting the recent success of heuristic planners as utilizing the topology of the benchmarks, as suggested by Hoffmann [13]. Specifically, it follows that FF's search algorithm is a polynomial solving mechanism in eight of the domains, under the idealizing assumption that its heuristic identifies the optimal relaxed distances. What's more, our proof methods shed light on which structural properties—at the level of the planning task's definition—are responsible for the topological phenomena. As most of the structural properties we identify are of a syntactical nature, this knowledge gives hints as to how the topological phenomena might be automatically recognizable.

The paper is organized as follows. We start by giving a formal framework for STRIPS and ADL domains in Section 2, and overviewing our investigated

domains in Section 3. Section 4 contains our definitions of local search topology, Section 5 describes Hoffmann's empirical findings in the competition domains and the hypotheses he makes. In Section 6, we present the key lemmata underlying most of our proofs in the single domains. These proofs are given in the subsequent sections: for each domain in turn, we give a formal definition of the domain, and prove what the topology of the respective state spaces looks like. Section 23 gives the overall picture that our results determine. Section 24 concludes and gives an outlook on what still needs to be done.

Please note that this technical report is mainly intended as a source of details for people who have read the short summarizing paper [12]. The latter paper is likely much better to get a general understanding of the investigation at hand, so if you have not yet read that paper, then you are strongly encouraged to do so.

2 Planning Framework

To enable proofs to properties of planning *domains*, rather than single tasks, we need a formal definition of what a planning domain *is*. Our definitions formalize in a straightforward manner the way how domains are usually handled in the community. The only other work we know of that uses a formal notion of planning domains is recent work by Malte Helmert [11]. There, the semantics of different transportation domains are formalized in order to prove their computational complexity. In difference to that, our definitions are strictly syntax-oriented—after all, the heuristic function we consider is extracted directly from the syntax of the planning task.

2.1 Predicates and Facts

The base entity on which planning tasks and domains, as we view them in the following, are built, are logical facts. These are predicates that are instantiated with constants. We take on a convenient view of instantiated predicates as tuples of symbol sequences, i.e., strings, where we assume throughout our investigation an underlying alphabet $\Sigma \supseteq \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots\}$.

Definition 1. *A* predicate *is a pair* (prd, ar). $prd \in \Sigma^+$ *is the* name, $ar \in \mathbb{N}$ *the* arity *of the predicate. Given a set* $\mathcal{P}$ *of predicates, the set of* facts based on $\mathcal{P}$ *is* $\mathcal{F}^{\mathcal{P}} := \{(prd, cst_1, \ldots, cst_{ar}) \mid (prd, ar) \in \mathcal{P}, cst_i \in \Sigma^+\}$. *For a fixed set of constants* $C \subseteq \Sigma^+$, *the set of* facts based on $\mathcal{P}$ and C *is* $\mathcal{F}^{\mathcal{P},C} := \{(prd, cst_1, \ldots, cst_{ar}) \mid (prd, ar) \in \mathcal{P}, cst_i \in C\}$.

2.2 States, Actions, and Tasks

Definition 2. *Given a collection of predicates* $\mathcal{P}$, *the set of* states based on $\mathcal{P}$ *is*

$$\mathcal{S}^{\mathcal{P}} := \{s \mid s \subseteq \mathcal{F}^{\mathcal{P}}\}$$

Accordingly, we denote by $\mathcal{S}^{\mathcal{P},C} := \{s \mid s \subseteq \mathcal{F}^{\mathcal{P},C}\}$*, for a given set* C *of constants, the* states based on $\mathcal{P}$ and C.

Definition 3. *Given a collection of predicates* $\mathcal{P}$*, the set of* actions based on $\mathcal{P}$ *is*

$$\mathcal{A}^{\mathcal{P}} := \{(pre, add, del) \mid pre, add, del \subseteq \mathcal{F}^{\mathcal{P}}\}$$

For an action $a = (pre(a), add(a), del(a))$*,* $pre(a)$ *is the* precondition*,* $add(a)$ *is the* add list*, and* $del(a)$ *is the* delete list *of* a.

The semantics of actions will be defined later. For now, we introduce the formal notion of propositional planning tasks.

Definition 4. *Given a collection of predicates* $\mathcal{P}$*, a (propositional)* planning task based on $\mathcal{P}$ *is a triple* (A, I, G) *where* A *is a set of actions* $A \subseteq \mathcal{A}^{\mathcal{P}}$*, and* $I \in \mathcal{S}^{\mathcal{P}}$*, the* initial state*, and* $G \in \mathcal{S}^{\mathcal{P}}$*, the* goal state*, are states.*

2.3 Operators, Instances, and Domains

Definition 5. *Given a collection of predicates* $\mathcal{P}$*, a* planning operator with p parameters based on $\mathcal{P}$ *is a function* $o : (\Sigma^+)^p \mapsto \mathcal{A}^{\mathcal{P}}$ *from* p *symbol sequences to the set of actions based on* $\mathcal{P}$.

Applying operators to tuples of constants (symbol sequences) yields actions. Instances (yielding tasks when associated with an operator set) are triples of constant sets, initial, and goal states.

Definition 6. *Given a collection of predicates* $\mathcal{P}$*, a* planning instance based on $\mathcal{P}$ *is a triple* (C, I, G)*. The* constants C *are symbol sequences* $C \subseteq \Sigma^+$*. The* initial state I *and the* goal state G *are states based on* $\mathcal{P}$ *and* C*, i.e.,* $I, G \in \mathcal{S}^{\mathcal{P},C}$.

A planning domain constitutes the predicates used, the operators yielding the action sets, and the legal instances.

Definition 7. *A* planning domain *is a triple* $\mathcal{D} = (\mathcal{P}, \mathcal{O}, \mathcal{I})$*.* $\mathcal{P}$ *is a set of predicates.* $\mathcal{O}$ *is a set of planning operators based on* $\mathcal{P}$*, and* $\mathcal{I}$ *is a set of instances based on* $\mathcal{P}$*. For* $(C, I, G) \in \mathcal{I}$*, we say that* (C, I, G) *is an* instance of $\mathcal{D}$.

Instances of a domain are directly associated with planning tasks.

Definition 8. *Given a planning domain* $\mathcal{D} = (\mathcal{P}, \mathcal{O}, \mathcal{I})$*, and an instance* (C, I, G) *of* $\mathcal{D}$*. The* planning task to (C, I, G) *is* (A, I, G)*, with*

$$A := \{o(cst_1, \ldots, cst_p) \mid o \in \mathcal{O} \text{ with } p \text{ parameters }, cst_i \in C\}$$

In the following, we will often identify planning instances with the corresponding planning tasks. For an instance of a planning domain $\mathcal{D}$, we also say that the instance and the respective task *belong to* $\mathcal{D}$.

2.4 Plans

Sequences of actions can be applied to states by the following semantics.

Definition 9. *Given a collection of predicates $\mathcal{P}$, a state s based on $\mathcal{P}$ and an action a based on $\mathcal{P}$. The* result of applying *(the sequence consisting solely of) a to s is*

$$Result(s, \langle a \rangle) := \begin{cases} (s \cup add(a)) \setminus del(a) & \text{if } pre(a) \subseteq s \\ \text{undefined} & \text{otherwise} \end{cases}$$

In the first case, the action is said to be applicable *in s. The result of applying a sequence $\langle a_1, \ldots, a_n \rangle$ to a state is recursively defined as*

$$Result(s, \langle a_1, \ldots, a_n \rangle) := Result(Result(s, \langle a_1, \ldots, a_{n-1} \rangle), \langle a_n \rangle).$$

Applying the empty sequence does no changes, i.e., $Result(s, \langle\rangle) := s$. The length *of a sequence $P = \langle a_1, \ldots, a_n \rangle$ is n, denoted by $|P|$.*

Plans are simple action sequences in our framework.

Definition 10. *Given a planning task (A, I, G). A* plan, or solution, for (A, I, G) *is a sequence $P \in A^*$ of actions in A for which $G \subseteq Result(I, P)$ holds. The task (A, I, G) is* solvable *if there is at least one plan for it.*

At various points, we need the notion of optimality.

Definition 11. *Given a planning task (A, I, G). A plan is* optimal for (A, I, G)*, if it has minimal length amongst all plans for (A, I, G). A plan $\langle a_1, \ldots, a_n \rangle$ is* minimal for (A, I, G)*, if no single action can be left out, i.e., if*

$$\forall\, 1 \leq i \leq n : \langle a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n \rangle \text{ is not a plan for } (A, I, G)$$

Proposition 1. *Given a planning task (A, I, G). If P is an optimal plan for (A, I, G), then P is also a minimal plan for (A, I, G).*

2.5 Relaxation

The heuristic function we consider is derived by ignoring the delete lists of the actions.

Definition 12. *Given an action $a = (pre(a), add(a), del(a))$, the* relaxation a^+ of a *is defined as*

$$a^+ := (pre(a), add(a), \emptyset)$$

For a set A of actions, the relaxation A^+ of A *is $A^+ := \{a^+ \mid a \in A\}$. The* relaxation of a planning task (A, I, G) *is (A^+, I, G). An action sequence $P = \langle a_1, \ldots, a_n \rangle \in A^*$ is a* relaxed plan for (A, I, G) *if $\langle a_1^+, \ldots, a_n^+ \rangle$ is a plan for the relaxation of (A, I, G).*

2.6 Extension to ADL

We have so far been restricted to STRIPS. In ADL, actions are more expressive because they can use complex preconditions and conditional effects. We extend our definitions accordingly. By $L_0^{\mathcal{P}}$, we denote all closed first-order logic formulae (no free variables) that use only the predicates in $\mathcal{P}$. We say, in line with our terminology, that such formulae are *based on* $\mathcal{P}$. Given a set of constants C, we denote by $L_0^{\mathcal{P},C}$ those formulae based on $\mathcal{P}$ were the only used constants are taken from C.

Definition 13. *Given a collection of predicates $\mathcal{P}$, the set of* conditional effects based on $\mathcal{P}$ *is*

$$\mathcal{E} := \{(con, add, del) \mid con \in L_0^{\mathcal{P}}, add, del \subseteq \mathcal{F}^{\mathcal{P}}\}$$

For a conditional effect $e = (con(e), add(e), del(e))$, $con(e)$, $add(e)$, and $del(e)$ are the effect condition, add- *and* delete *list of e, respectively. The set of* ADL actions based on $\mathcal{P}$ *is*

$$\mathcal{A}_{adl}{}^{\mathcal{P}} := \{(pre, E) \mid pre \in L_0^{\mathcal{P}}, E \subseteq \mathcal{E}\}$$

For an ADL action $a = (pre(a), E(a))$, $pre(a)$ is the precondition, *and $E(a)$ are the* effects *of a.*

We will soon see what the semantics of an ADL action are. Let us first adjust our definitions of tasks, operators, instances, and domains. Little changes suffice in all cases.

Definition 14. *Given a collection of predicates $\mathcal{P}$, an* ADL planning task based on $\mathcal{P}$ *is a triple (A, I, G) where A is a set of ADL actions $A \subseteq \mathcal{A}_{adl}{}^{\mathcal{P}}$, the* initial state $I \in \mathcal{S}^{\mathcal{P}}$ *is a state, and $G \in L_0^{\mathcal{P}}$ is a formula based on $\mathcal{P}$.*

ADL operators can make use of all-quantified effects, which are applied to all constants for which the effect condition holds. Such effects have free variables in the condition, add- or delete-list. For convenience, we assume such effects are compiled by the operator function into several single effects that are fully instantiated. This can be done by replacing the free variables with all relevant constants. For doing this, the operator function needs to take as an additional parameter the set of these constants.

Definition 15. *Given a collection of predicates $\mathcal{P}$, an* ADL planning operator with p parameters based on $\mathcal{P}$ *is a function $o : (\Sigma^+)^p \times 2^{\Sigma} \mapsto \mathcal{A}_{adl}{}^{\mathcal{P}}$ from p symbol sequences to the set of ADL actions based on $\mathcal{P}$.*

Definition 16. *Given a collection of predicates $\mathcal{P}$, an* ADL planning instance based on $\mathcal{P}$ *is a triple (C, I, G). The* constants C *are symbol sequences $C \subseteq \Sigma^+$. The* initial state $I \in \mathcal{S}^{\mathcal{P},C}$ *is a state, the* goal condition $G \in L_0^{\mathcal{P},C}$ *is a formula based on $\mathcal{P}$ and C.*

For the sake of notational simplicity, we allow ADL domains to also comprise simple STRIPS operators.

Definition 17. *An* ADL planning domain *is a triple $\mathcal{D} = (\mathcal{P}, \mathcal{O}, \mathcal{I})$. $\mathcal{P}$ is a set of predicates. $\mathcal{O}$ is a set of ADL or STRIPS planning operators based on $\mathcal{P}$, and $\mathcal{I}$ is a set of ADL instances based on $\mathcal{P}$. For $(C, I, G) \in \mathcal{I}$, we say that (C, I, G) is an* instance of $\mathcal{D}$.

Definition 18. *Given an ADL planning domain $\mathcal{D} = (\mathcal{P}, \mathcal{O}, \mathcal{I})$, and an instance (C, I, G) of $\mathcal{D}$. The* planning task to (C, I, G) *is (A, I, G), with*

$$A := \{o(cst_1, \ldots, cst_p) \mid o \in \mathcal{O} \text{ } p\text{-parameter STRIPS operator}, cst_i \in C\}$$
$$\cup \ \{o(cst_1, \ldots, cst_p, C) \mid o \in \mathcal{O} \text{ } p\text{-parameter ADL operator}, cst_i \in C\}$$

We overload the *Result* function to handle both STRIPS and ADL actions.

Definition 19. *Given a collection of predicates $\mathcal{P}$, a state s based on $\mathcal{P}$ and an ADL action a based on $\mathcal{P}$. The* result of applying *(the sequence consisting solely of) a to s is*

$$Result(s, \langle a \rangle) := \begin{cases} (s \cup A(s,a)) \setminus D(s,a)) & \text{if } s \models pre(a) \\ undefined & otherwise \end{cases}$$

with $A(s,a) := \bigcup_{e \in E(a), s \models con(e)} add(e)$ and $D(s,a) := \bigcup_{e \in E(a), s \models con(e)} del(e)$

In words, we apply all effects of an ADL action that have their effect condition satisfied in the state at hand. We are conservative in the sense that we first make all add effects true, and then remove the union of all deletes. The definitions of *Result* for the other cases (than that of a singleton sequence) remain unchanged.

Definition 20. *Given an ADL planning task (A, I, G). A* plan, or solution, for *(A, I, G) is a sequence $P \in A^*$ of actions in A for which $Result(I, P) \models G$ holds.*

The notions of optimal and minimal plans remain unchanged. The relaxation translates as follows.

Definition 21. *Given an ADL action $a = (pre(a), E(a))$, the* relaxation a^+ of *a is defined as*

$$a^+ := (pre(a), \{(con, add, \emptyset) \mid (con, add, del) \in E(a)\})$$

The definitions for sets of actions, planning tasks, and plans remain the same. We remark that ignoring delete lists simplifies a task only if all formulae are negation-free. Within a domain, any task can be polynomially transformed (normalized) to have that property: compute the negation normal form to all formulae (negations only in front of facts), then introduce for each negated fact $\neg(\mathrm{prd}, \mathrm{cst}_1, \ldots, \mathrm{cst}_{ar})$ a new fact $(\text{not-prd}, \mathrm{cst}_1, \ldots, \mathrm{cst}_{ar})$ and make sure it is true in a state iff $(\mathrm{prd}, \mathrm{cst}_1, \ldots, \mathrm{cst}_{ar})$ is false. In this report, we define our domains such that formulae are negation-free from begin with. From now on, our definitions are valid for STRIPS and ADL alike.

3 Investigated Domains

The domains we investigate will be defined in detail in their respective sections. To give the reader some idea of what is to come, we briefly list the domains here.

As said, we focus on 20 different STRIPS and ADL benchmark domains, including all 13 domains that have been used in the AIPS-1998 and AIPS-2000 planning system competitions. The competition domains are *Assembly*, *Blocksworld-arm*, *Freecell*, *Grid*, *Gripper*, *Logistics*, *Miconic-ADL*, *Miconic-SIMPLE*, *Miconic-STRIPS*, *Movie*, *Mprime*, *Mystery*, and *Schedule*. We assume that the reader is familiar with these domains. Descriptions can be looked up in the articles on the competitions [17, 2]. Apart from the 13 competition domains, we investigate 7 more benchmark domains often used in the literature. These domains can be briefly described as follows.

1. *Blocksworld-no-arm*: unlike in the competition version, this encoding does not use an explicit robot arm; instead, blocks are moved around directly by operators moving them from one block to another block, or from the table to a block, or from a block to the table.
2. *Briefcaseworld*: transportation domain using conditional effects; objects can be put into or taken out of the (single) briefcase, and a move operator, which can be applied between any two locations, moves all objects along that are currently inside.
3. *Ferry*: also a transportation domain, with operators that board a car onto the (single) ferry, or debark a car from it; the ferry can only transport one car at a time, and a sail operator can be applied to move the ferry between any two locations.
4. *Fridge*: for a number of fridges, the broken compressors must be replaced. This involves unfastening a number of screws that hold the compressors, removing the old compressors and attaching the new ones, and fastening all screws again.
5. *Hanoi*: encoding of the classical Towers of Hanoi problem, using a move(x, y, z) operator to move a disc x from a disc y to a disc z (the pegs are encoded as discs that can not be moved).
6. *Simple-Tsp*: a trivial version of the TSP problem, where the goal is that all locations have been visited, and a move operator can be applied between any two locations (all action costs being equal, as usual in STRIPS).
7. *Tyreworld*: a number of flat tyres must be replaced, which involves inflating the spare tyres, loosening the nuts, and in turn jacking up the hubs, undoing the nuts, removing the flat tyre and putting on the spare one, doing up the nuts, and jacking down the hub again; finally, all nuts must be tightened, and the tools must be put away into the boot.

For formally defining a domain, one must amongst other things decide what exactly the instances are. For almost none of the investigated domains is there such a definition in the literature. The obvious approach we have taken is to abstract from the known example suits.

4 Local Search Topology

Following Hoffmann [13] we now give our base definitions of local search topology. Note that we consider only solvable tasks. While some of the definitions would make sense without that restriction, our main aim is after all to identify the reasons why local search can be so successful. On unsolvable tasks local search will be lost anyway.

The search space we are interested in is the following.

Definition 22. *Given a solvable (STRIPS or ADL) planning task* (A, I, G). *The set* S *of* reachable states *is*

$$S := \{s \mid \exists\ P \in A^* : Result(I, P) = s\}$$

The reachable states together with their accessibility relation define a directed graph: the state space.

Definition 23. *Given a solvable planning task* (A, I, G). *The* state space to (A, I, G) *is the directed graph* (S, E) *where the nodes are the reachable states, and the edge set is*

$$E := \{(s, s') \mid s, s' \in S,\ \exists\ a \in A : Result(s, \langle a \rangle) = s'\}$$

For any reachable state, we define the goal distance of the state: the length of an optimal plan achieving the goal from the state; and the relaxed distance of the state: the length of an optimal *relaxed* plan achieving the goal. The latter notion defines the heuristic function whose properties we will be investigating.

Definition 24. *Given the state space* (S, E) *to a solvable planning task. The* goal distance of $s \in S$ *is*

$$gd(s) := min\{|P| \mid P \in A^*, P \text{ is plan for } (A, s, G)\}$$

and the relaxed distance of s *is*

$$h^+(s) := min\{|P| \mid P \in A^*, P \text{ is relaxed plan for } (A, s, G)\}$$

We set the minimum over an empty set to ∞.

Definition 25. *Given the state space* (S, E) *to a solvable planning task. A* heuristic *is a function* $h : S \mapsto \mathbb{N}_0 \cup \{\infty\}$, *such that* $h(s) = 0 \Leftrightarrow gd(s) = 0$.

The h^+ function is a heuristic.

Definition 26. *Given the state space* (S, E) *to a solvable planning task. A state* $s \in S$ *is a* dead-end, *if* $gd(s) = \infty$. *A heuristic* h *is* completeness preserving, *if* $h(s) = \infty \Rightarrow gd(s) = \infty$.

The h^+ function is completeness preserving: if a task can not be solved even when ignoring the delete lists, then the task is unsolvable.

With respect to dead ends, any planning task falls into one of the following four classes.

Definition 27. *Given the state space (S, E) to a solvable planning task, and a completeness-preserving heuristic h. The state space is*

1. undirected, *if* $\forall (s, s') \in E : (s', s) \in E$
2. harmless, *if* $\exists (s, s') \in E : (s', s) \notin E$, *and* $\forall s \in S : gd(s) < \infty$
3. recognized, *if* $\exists s \in S : gd(s) = \infty$, *and* $\forall s \in S : gd(s) = \infty \Rightarrow h(s) = \infty$
4. unrecognized, *if* $\exists s \in S : gd(s) = \infty \wedge h(s) < \infty$

Note that an undirected state space can still contain dead ends—but only when it is unsolvable, in which case all states are dead ends.

For SAT problems, the topology of search spaces with respect to the behavior of local search has been investigated by Frank et al. [8]. As the basis of their work, Frank et al. formally define a partitioning of the search space into plateaus of different kinds. For our purposes, we extend their definitions to deal with search spaces where edges can be directed.

Definition 28. *Given the state space (S, E) to a solvable planning task, and a completeness-preserving heuristic h. For $l \in \mathbb{N}_0 \cup \{\infty\}$, a* plateau P of level l *is a maximal subset of S for which the induced subgraph in (S, E) is strongly connected, and $h(s) = l$ for each $s \in P$.*

Plateaus are regions that are equivalent under reachability aspects, and look the same from the point of view of the heuristic function. Obviously, each state s lies on exactly one plateau.

Definition 29. *Given the state space (S, E) to a solvable planning task, a completeness-preserving heuristic h, and a plateau P. A state $s \in P$ is an* exit of P, *if there is a state $s' \notin P$ such that $(s, s') \in E$ and $h(s') \leq h(s)$. s is an* improving *exit, if, for at least one such s', $h(s') < h(s)$.*

Exits are states from which one can leave a plateau without increasing the value of the heuristic function. In undirected graphs, like are considered by Frank et al. [8], leaving a plateau implies changing the value of the heuristic function, so all exits are improving there. According to the proportion of exits on a plateau, Frank et al. divide plateaus into four classes: local minima, benches, contours, and global minima. Taking account of directed edges, we have two types of exits, and define the following six different classes.

Definition 30. *Given the state space (S, E) to a solvable planning task, and a completeness-preserving heuristic h.*

1. *A* recognized dead end *is a plateau P of level $h = \infty$.*
2. *A* local minimum *is a plateau P of level $0 < h < \infty$ that has no exits.*

3. *A* plain *is a plateau P of level $0 < h < \infty$ that has at least one exit, but no improving ones.*
4. *A* bench *is a plateau P of level $0 < h < \infty$ that has at least one improving exit, and at least one state that is not an improving exit.*
5. *A* contour *is a plateau P of level $0 < h < \infty$ that consists entirely of improving exits.*
6. *A* global minimum *is a plateau P of level 0.*

Each plateau belongs to exactly one of the above classes. In our solvable instances, global minima are exactly the plateaus of level 0. With a completeness-preserving heuristic, recognized dead ends are irrelevant, and can be ignored. From local minima, there is no direct way of getting closer to the goal. From benches, there is. From contours, one can get closer immediately. Plains behave as a kind of entrance to either local minima or benches.

Definition 31. *Given the state space (S, E) to a solvable planning task, and a completeness-preserving heuristic h. A* flat path *is a path where all states on the path have the same heuristic value. For a plateau P, the* flat region $FR(P)$ from P *is the set of all plateaus P' such that there is a flat path from some $s \in P$ to some $s' \in P'$.*

Definition 32. *Given the state space (S, E) to a solvable planning task, and a completeness-preserving heuristic h. A plain P* leads to benches *if $FR(P)$ contains some bench or contour. Otherwise, P* leads to local minima.

We remark here that at this point the definitions in our short paper [12] are somewhat simplified: there, plains that lead to local minima are local minima per definition already, same with benches; this is done by allowing as an exit of a plateau any state—not necessarily on the plateau itself—that can be reached on a flat path, and that has a better evaluated neighbour. While our definition here—taken from Hoffmann [13]—captures the state space topology in more detail, both definitions are equally appropriate for the subsequent investigation.

Apart from dead ends and local minima, we will be concerned with the maximal exit distance. For benches, one can directly improve on the heuristic value. The hard bit for local search there can be to find the exits. One can assess the difficulty of finding the exits by a number of criteria, like size of the bench or exit percentage. The criterion we consider is especially relevant for FF's heuristic function, as the next section will explain.

Definition 33. *Given the state space (S, E) to a solvable planning task, and a completeness-preserving heuristic h. For a state s the* exit distance $ed(s)$ *is the length of a shortest flat path from s to some s' such that there is some s'' with $(s', s'') \in E, h(s'') < h(s')$. The maximal exit distance in the state space is the maximum over the exit distances of all states s with $h(s) < \infty$.*

At this point, we differ somewhat from Hoffmann's original definition: while we do not maximize the exit distance over recognized dead ends—as those can

be ignored anyway—we *do* include states on local minima. Note that states on local minima have infinite exit distance so in the presence of these the maximal exit distance will be infinite. In difference to that, Hoffmann [13] maximizes the exit distances only over the states on benches. With our definition here we want to take account of the fact that in the presence of local minima leaving a bench through an exit might not always be the right thing to do: like when all exits lead into a local minimum. The presence of local minima dominates, in that sense, the difficulty of getting off benches. We will see that many of our domains do in fact not contain any local minima at all.

5 Hoffmann's Hypotheses

Hoffmann [13] empirically investigates topological properties in the competition domains, with respect to h^+; the approach being to compute h^+ for the states in example state spaces and measure parameters of the resulting local search topology. Because computing h^+ is NP-hard, the investigation is restricted to rather small example instances. Hoffmann measures (amongst other things) the dead-end class of the instances, the number of states on local minima, and the maximal exit distance. He summarizes his observations in the table shown in Figure 1.

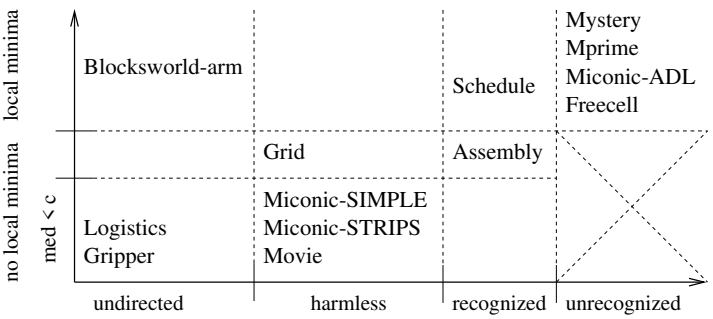

Fig. 1. Hoffmann's overview of his observations in the competition domains.

The x-axis in Figure 1 shows the different dead-end classes by increasing difficulty (for the domains in a dead-end class i, all investigated instances belong to a class $j \leq i$, and at least one instance belongs to class i). The y-axis combines local minima percentage with maximal exit distance: in the uppermost part of the table, the domains are shown were Hoffmann *did* find any local minima; below that, domains are shown were there were no local minima at all in Hoffmann's instances (unrecognized dead ends imply the existence of local minima, so that part of the table is crossed out); in the lowermost part of the table domains are shown were all instances had the same constant maximal exit distance.

Hoffmann names the table in Figure 1 the *planning domain taxonomy*, and hypothesises that his observations hold true for *all* instances in the respective domains: that all instances of a domain shown in dead-end class i are in a class $j \leq i$; that no instance of a domain shown in the lower parts of the table contains any local minima; that for a domain shown in the lowest part there is a constant $c \in \mathbb{N}$ such that the maximal exit distance in all instances is at most c. The pratical relevance of this, as Hoffmann argues, is the following. The domains on the left bottom side of the taxonomy are "simple" for heuristic planners like HSP and FF because there the h^+ function, which those planners approximate, is a heuristic function of high quality. The majority of the competition domains seem to lie on the simple side. So, if these observations carry over to all instances in the respective domains, then the taxonomy can serve as an explanation for the good behaviour of the aforementioned planners.

For domains in the lowermost part of the taxonomy, FF's search algorithm is in fact polynomial in the sense that (assuming the heuristic function is given) it looks at polynomially many states before reaching the goal [13]. FF's search algorithm is the following.

$s := I$
while $h^+(s) \neq 0$ **do**
    do breadth first search for s', $h^+(s') < h^+(s)$
    $s := s'$
endwhile

Without local minima, each iteration of this algorithm crosses a bench, so breadth first search finds a better state at maximal depth $c + 1$; the branching factor is limited by the number of actions in the task; each iteration improves the heuristic value by at least one, so after at most $h^+(I)$ iterations, a goal state is reached.

In the subsequent investigation, we verify how much truth there is in Hoffmann's hypotheses. Going beyond that, we look at a larger number of domains, and determine exactly which part of Hoffmann's taxonomy they belong to. Obviously, we need only prove positive results (like the non-existence of local minima) for those domains were Hoffmann did not find a negative example: there is naught to prove about the *Freecell*, *Miconic-ADL*, *Mprime* or *Mystery* domains, and for *Blocksworld-arm* and *Schedule* we already know that there are local minima; likewise, any domain that was observed in a dead end class i can only end up in a class $j \geq i$.

As it turns out, all of Hoffmann's hypotheses are true except those about the *Assembly* domain. Like in Hoffmann's investigation, our main aim is to explain the good performance of local search planners, so we focus on *solvable* instances only—on unsolvable instances, local search is lost anyway. The subsequent sections, each considering a single domain, are arranged in an order such that the proofs move along the line from easy knockovers to rather involved theorems. Before we start, we present the core lemmata that underly most of the (easier) proofs.

6 A Theoretical Core

In this section, we present some theory determining, given a propositional planning task, sufficient conditions for (some of) the properties hypothesized above. The lemmata identify the core common features that we found while proving the hypotheses for each single domain. Thereby, this section

1. simplifies and shortens the proofs in the subsequent investigation;
2. provides insights into the common features of the instances in different domains that lead to similar behaviour with respect to the h^+ function, different as the domains may look at first sight;
3. gives hints as to how some of the hypothesized properties might be automatically recognized or at least approximated.

The first point is obvious; as for the second and third points, we add some discussion in this section, where appropriate. The third point is important because the respective decision problems are hard: the decision about dead ends has been proven to be PSPACE-hard by Hoffmann and Nebel [14], and little effort is involved in showing that deciding about the existence of local minima under h^+ is likewise PSPACE-hard—in an arbitrary task, "flatten" h^+ by introducing an artificial action with empty preconditions that achieves the original goal but deletes some other newly introduced goal; the modified task exhibits local minima iff the original task is unsolvable.

We first look at the reasons behind the dead end class of many domains. We will need the formal notion of inconsistency at some points.

Definition 34. *Given a planning task (A, I, G). Two facts are* inconsistent *if there is no reachable state $s \in S$ that contains both of them. Two* sets *of facts $\mathcal{F}$ and $\mathcal{F}'$ are* inconsistent *if each fact in $\mathcal{F}$ is inconsistent with at least one fact in $\mathcal{F}'$.*

6.1 Dead Ends

Earlier work by Koehler and Hoffmann [15] has identified a simple sufficient condition for STRIPS planning tasks having an undirected state space. The observation is that in the instances of many benchmark domains there is always an inverse action that undoes exactly what one has just done.

Definition 35. *Given a STRIPS planning task (A, I, G). An action $a \in A$ is* invertible, *if*

1. *$add(a)$ is inconsistent wth $pre(a)$,*
2. *$del(a) \subseteq pre(a)$, and*
3. *there is an action $\overline{a} \in A$ such that*
   (a) *$pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$,*
   (b) *$add(\overline{a}) = del(a)$, and*
   (c) *$del(\overline{a}) = add(a)$.*

The first two requirements here make sure that a's effects have really taken place, i.e., that the add effects were not already true, and that the delete effects were not false anyway; then, $\overline{a}$ can be applied to undo exactly those effects.

Lemma 1. *Given a STRIPS planning task (A, I, G). If all actions $a \in A$ are invertible, then the state space to the task is undirected.*

Proof. Say we have a reachable state $s \in S$, and an action a that is applicable, $pre(a) \subseteq s$. We need to show that there is an action $\overline{a}$ such that $Result(Result(s, \langle a \rangle), \langle \overline{a} \rangle) = s$. By prerequisite, we can choose an action $\overline{a}$ that inverts a in the sense of Definition 35. Applying $\overline{a}$ in $Result(s, \langle a \rangle) =: s'$ (note here that $pre(\overline{a}) \subseteq s'$ because $s' \supseteq (pre(a) \cup add(a)) \setminus del(a)$) leads us back to s:

$$\begin{aligned} Result(s', \langle \overline{a} \rangle) &= (s' \cup add(\overline{a})) \setminus del(\overline{a}) \\ &= (((s \cup add(a)) \setminus del(a)) \cup del(a)) \setminus add(a) \\ &= (s \cup add(a)) \setminus add(a) \quad \text{as } del(a) \subseteq pre(a) \subseteq s \\ &= s \quad \text{as } add(a) \cap s = \emptyset \end{aligned}$$

For the last equation, $add(a) \cap s = \emptyset$ holds because $add(a)$ is inconsistent wth $pre(a) \subseteq s$.

All parts of Definition 35 are purely syntactical, and can therefore be verified in polynomial time, except the first requirement, inconsistency of the add list with the preconditions. Now, while inconsistency is as hard to decide as planning itself, there are several polynomial, and empirically fast, approximation techniques known from the literature [7, 9, 18].

We next identify a sufficient condition for a STRIPS task containing no dead ends, i.e., being at most harmless. One key notion is a weaker version of invertibility, requiring only the possibility to get back to a state that contains *at least* as many facts as the state where one started from.

Definition 36. *Given a STRIPS planning task (A, I, G). An action $a \in A$ is* at least invertible, *if there is an action $\overline{a} \in A$ such that*

1. $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$,
2. $add(\overline{a}) \supseteq del(a)$, *and*
3. $del(\overline{a})$ *is inconsistent with* $pre(a)$.

The first two conditions of the definition are obvious. As for the last point, note that we do not refer to a's add list at all; inconsistency of $\overline{a}$'s deletes with a's preconditions implies that $\overline{a}$ deletes only facts that have not been true in s anyway (but might have been made true by a's add list). Note also that Definition 35 is stronger than Definition 36: if $del(\overline{a}) = add(a)$, and $add(a)$ is inconsistent with $pre(a)$, then, of course, $del(\overline{a})$ is inconsistent with $pre(a)$.

Another reason why an action can be harmless is that its delete effects are irrelevant—they are not important for anything in the task but the action itself—in combination with its add effects being static, that is, not deleted by any action:

by the latter property, applying the action once is enough, and by the former property, applying the action does not rule out any possibilities except applying it a second time.

Definition 37. *Given a STRIPS planning task (A, I, G). An action $a \in A$* has irrelevant delete effects*, if*

$$del(a) \cap (G \cup \bigcup_{a \neq a' \in A} pre(a')) = \emptyset$$

An action has static add effects*, if*

$$add(a) \cap \bigcup_{a' \in A} del(a') = \emptyset$$

If all actions in a task are either at least invertible or have irrelevant delete- and static add-effects, then the state space is at most harmless.

Lemma 2. *Given a solvable STRIPS planning task (A, I, G). If it holds for all actions $a \in A$ that either*

1. *a is at least invertible, or*
2. *a has irrelevant delete effects and static add effects,*

then there are no dead ends in the state space to the task, i.e., $gd(s) < \infty$ for all $s \in S$.

Proof. To any reachable state $s = Result(I, \langle a_1, \ldots, a_n \rangle) \in S$, we need to identify a solution P for (A, s, G). Let $\langle p_1, \ldots, p_m \rangle \in A^*$ be a solution for (A, I, G) (which exists as (A, I, G) is solvable by prerequisite). We construct P as follows.

$M := \emptyset$
for $i := n \ldots 1$ **do**
  if a_i is at least invertible by $\overline{a_i}$ **then**
    if $\overline{a_i} \notin M$ apply $\overline{a_i}$ **endif**
  else $M := M \cup \{a_i\}$
  endif
endfor
for $i := 1 \ldots m$ **do**
  if $p_i \notin M$ **then** apply p_i **endif**
endfor

M serves as a kind of memory set for the actions that could not be inverted. We need to prove that the preconditions of all applied actions are fulfilled in the state where they are applied, and that the goals are true upon termination.

Let us start with the first loop. We denote by $s_i := result(I, \langle a_1, \ldots, a_i \rangle)$ the state after executing the ith action on the path to s, and by s'_i the state before the first loop starts with value i. We prove (for use below) the stronger property

$$s'_i \supseteq (s_i \cap (G \cup \bigcup_{a \in A \setminus M_i} pre(a))) \cup \bigcup_{a \in M_i} add(a)$$

M_i here denotes the current state of the set. Applicability of actions is part of the proof. We proceed by backward induction over i. If $i = n$, we got $s'_i = s_i$ and $M_i = \emptyset$, so the equation is trivially true. Now say the equation is true for $i \geq 1$. We prove that the equation holds for $i-1$. If a_i is not at least invertible, then no action is applied, $s'_{i-1} = s'_i$, and $M_{i-1} = M_i \cup \{a_i\}$. Concerning the left hand side of the equation, we observe that a_i does by prerequisite not delete any fact from $G \cup \bigcup_{a \in A \setminus M_{i-1}} pre(a)$ (M_{i-1} contains a_i), so all relevant facts from s_{i-1} have already been true in s'_i; concerning the right hand side of the equation, we observe that the facts in $add(a_i)$ are never deleted by prerequisite, so $\bigcup_{a \in M_{i-1}} add(a)$ is contained in s'_i. Now say that a_i is at least invertible by $\overline{a_i}$. We got $M_{i-1} = M_i$. Say $\overline{a_i}$ is applied. It is applicable because its preconditions are contained in s_i, and it is not an element of M_i. For the resulting state s'_{i-1}, all facts that a_i has deleted from s_{i-1} are added, and only facts are deleted that have not been true in s_{i-1} anyway; also, none of the add effects of actions in M_i is deleted, so the equation is fulfilled. Finally, if $\overline{a_i}$ is not applied, then its add effects are already contained in s'_i, subsuming the delete effects of a_i.

Inserting $i = 0$ in the equation we have just proven, we get

$$s_0 \supseteq (I \cap (G \cup \bigcup_{a \in A \setminus M_0} pre(a))) \cup \bigcup_{a \in M_0} add(a)$$

s_0 is the state from which the second loop is started. So we start a solution plan, excluding the actions in a set M_0, from a state including all initial facts that are contained in the goal or in the precondition of any action not in M_0. As the state additionally contains all add effects of all actions in M_0, and those add effects are not deleted by any action, it is clear that we can simply skip the actions in M_0 and achieve the goal.

Definition 37 is purely syntactical, Definition 36 is purely syntactical except its last part, requiring computation of inconsistency. As said above, several good approximation techniques for inconsistency are known from the literature.

The two properties handled above, undirected and harmless state spaces, are obviously properties of the planning tasks themselves, independent of the h^+ function. Different from that, the third dead end class, recognized dead ends, *does* depend on the heuristic function. We did, however, not find a bridge from syntax to semantics for this case. Anyway, only two of our domains belong to that class according to Hoffmann's hypotheses, and, as it will turn out, for one of these domains the hypothesis is wrong.

6.2 Local Minima

We now prove some theory towards sufficient conditions for the non-existence of local minima under h^+. We first prove a technical lemma that will be used in this subsection, and in some of the more complicated proofs in the theory below. In the following, we denote $A(a,s) := add(a)$ if a is a STRIPS action, and $A(a,s) := \bigcup_{e \in E(a), s \models con(e)} add(e)$ if a is an ADL action; likewise, $D(a,s) := del(a)$ or $D(a,s) := \bigcup_{e \in E(a), s \models con(e)} del(e)$.

Lemma 3. *Given a STRIPS or ADL planning task (A, I, G), a state s, and an action a that is applicable in s yielding the state s'. If there is a relaxed plan P^+ for (A, s, G) that starts with a, and an action $\overline{a} \in A$ that is applicable in s' and for which $A(\overline{a}, s') \supseteq D(a, s)$ holds, then a relaxed plan for (A, s', G) can be constructed by removing a from P^+, and adding $\langle \overline{a} \rangle$ as a prefix.*

Proof. After executing the relaxed version of $\overline{a}$ in s', all facts are true that were true in s, plus the facts that were made true by a. Thus, P^+ without a is a relaxed plan from there.

The importance of Lemma 3 is that, with the Lemma's terminology, there is a relaxed plan starting in s' that *has the same length* as the one for s. So if the relaxed plan for s was optimal, we can conclude that $h^+(s') \leq h^+(s)$ holds.

Examples of STRIPS actions that obviously qualify for the role of $\overline{o}$ in Lemma 3 are the inverse, and at least inverse, actions in the sense of Definitions 35 and 36. In fact, a slightly weaker requirement is already sufficient.

Definition 38. *Given a STRIPS planning task (A, I, G). An action $a \in A$ is* at least relaxed invertible*, if there is an action $\overline{a} \in A$ such that*

1. $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$,
2. *and* $add(\overline{a}) \supseteq del(a)$.

The delete effects need not be mentioned in the above definition because $\overline{a}$, in the context of Lemma 3, is only used within the relaxation anyway. Note that both invertible and at least invertible actions are also at least relaxed invertible.

The key property behind the lack of local minima under h^+ is, in most of our domains, that the relaxation respects the actions of the real task in the following sense.

Definition 39. *Given a STRIPS or ADL planning task (A, I, G). An action $a \in A$* is respected by the relaxation *if, for any reachable state $s \in S$ such that a starts an optimal plan for (A, s, G), there is an optimal relaxed plan for (A, s, G) that also starts with a.*

As a simple example for an action that is respected by the relaxation, consider loading or unloading a truck in *Logistics*: the only way of transporting a package within its initial or destination city is by using the local truck, so *all* plans, real or relaxed, must use the respective action. We will say some more on that topic below.

We now put together the respect of the relaxation for the real actions, Definition 39, the existence of actions that at least invert an action's effects in the relaxation, Definition 38, and irrelevant delete effects, Definition 37, to obtain our main tool for identifying STRIPS planning tasks where h^+ does not yield any local minima.

Lemma 4. *Given a STRIPS planning task (A, I, G), such that the state space (S, E) does not contain unrecognized dead ends. If each action $a \in A$ either*

1. *is respected by the relaxation and at least relaxed invertible, or*
2. *has irrelevant delete effects,*

then there are no local minima in (S, E) under evaluation with h^+.

Proof. The states with $gd(s) = \infty$ are not on a local minimum by prerequisite, with $h^+(s) = \infty$. We will prove that to each reachable state s with $0 < gd(s) < \infty$ there is an action a starting an optimal plan for (A, s, G) such that $h^+(Result(s, \langle a \rangle)) \leq h^+(s)$. This proves our lemma: iterating the argument, we obtain a path from s to a goal state on which the value of h^+ decreases monotonically. Therefore, there is a flat path from s to some state s' that is not on the same plateau as s (at the latest, the goal state is not on the same plateau as s), so s can not lie on a local minimum.

Our choice of a is any action that starts an optimal plan for (A, s, G). We denote $s' := Result(s, \langle a \rangle)$. First case, a is respected by the relaxation and at least relaxed invertible. So a also starts an optimal relaxed plan P^+ for (A, s, G), and there is an action $\overline{a} \in A$ with $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$ and $add(\overline{a}) \supseteq del(a)$. We can apply Lemma 3 to obtain a relaxed plan for (A, s', G) that has the same length as P^+, yielding $h^+(s') \leq h^+(s)$. Second case, a has irrelevant delete effects, i.e., $del(a) \cap (G \cup \bigcup_{a \neq a' \in A} pre(a')) = \emptyset$. Let P^+ be an optimal relaxed plan for (A, s, G). If P^+ contains a, then we can assume it starts with a; the facts that a deletes are not needed by any other action, nor by the goal, so removing a from P^+ yields a relaxed plan for s', giving us $h^+(s') \leq h^+(s) - 1$.[1] If P^+ does not contain a, then P^+ is also a relaxed plan for s', because a's delete effects are needed for nothing except a itself. So then, again, $h^+(s') \leq h^+(s)$, concluding our argumenation.

Planning tasks where there are unrecognized dead ends do contain local minima (within those unrecognized dead ends, namely) anyway, so we must postulate that this is not the case, like when the task at hand is solvable and undirected, or harmless.

The second prerequisite of Lemma 4, and the second half of the first prerequisite, are purely syntactical, and are fulfilled by planning tasks qualifying for the undirectedness or harmlessness criteria we have seen above. The semantics come in by the first half of the first prerequisite, postulating that an action, if it does have relevant delete effects, must be respected by the relaxation. The reason why this is the case in many of our domains is that, intuitively, if an action a is needed for an optimal solution, then this action either

- achieves some fact that is needed by *all* solutions from the current state, no matter if relaxed or not, and that can not be achieved in a better way (there is no action achieving strictly more facts than a), or
- a achieves some fact representing one way to get closer to the goals, where no other way is strictly better, relaxed or not.

[1] As a matter of fact, $h^+(s') = h^+(s) - 1$ as the value of h^+ can decrease by at most 1 within a single action application.

A typical example for an action of the first kind is the *Logistics* example mentioned above, loading or unloading a truck. A typical example for an action of the second kind is picking up a ball from room A in *Gripper*: if both hands are empty, it does not matter which one picks up the ball.

Our proofs for ADL tasks will generally be along the lines of Lemma 4. We do not write up a separate core lemma, because: only 3 of our domains without local minima are ADL domains anyway (one of them, $Fridge$, using no conditional effects); the reasons for the lack of local minima in each of these domains are somewhat varied; in any case, though the intuitions behind an ADL action being (at least relaxed) invertible, or having irrelevant delete effects, are quite similar to the intuitions for STRIPS actions, writing up those intuitions syntactically is rather awkward.

6.3 Maximal Exit Distance

With respect to the maximal exit distance in a state space, we merely mention a simple technical property underlying the technique we use for proving the hypotheses.

In planning tasks where the actions are either respected by the relaxation and at least relaxed invertible, or have irrelevant delete effects—in planning tasks that fulfill the requirements for Lemma 4—optimal solution paths are flat in the sense that h^+ does not increase when one moves along them. So our proof technique will be to move along an optimal solution path from any reachable state s, and identify at which point, at the latest, a strictly shorter relaxed plan than the optimal one for s can be constructed. This is the case, for example, when one applies an action that has irrelevant delete effects.[2]

Proposition 2. *Given a STRIPS planning task (A, I, G), a reachable state s, and an action a with irrelevant delete effects, such that a starts an optimal relaxed plan P^+ for (A, s, G). Then, $h^+(Result(s, \langle a \rangle)) < h^+(s)$, i.e., s is an exit state under h^+.*

Proof. Removing a from P^+ yields a relaxed plan for $(A, Result(s, \langle a \rangle), G)$: in the successor state, all add effects of a are true, and only irrelevant facts, not needed by any action or for the goal, have been removed. So the remainder of P^+ is enough to achieve the goals from there.

7 Tsp

The *Tsp* domain we consider is a trivial variant where locations are fully connected with equal cost. It is defined as follows.

[2] The following argument is in fact part of the proof to Lemma 4; we write the implication up separately for the sake of readability.

Definition 40. *The Tsp domain is* $Tsp = (\mathcal{P}, \mathcal{O}, \mathcal{I})$*, where the set of predicates considered is*

$$\mathcal{P} = \{(\textit{not-eq}, 2), (at, 1), (\textit{visited}, 1)\}$$

$\mathcal{O}$ *consists of the single operator*

- *move:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\textit{move}(x, y) := (\{(\textit{not-eq}, x, y), (at, x)\}, \{(at, y), (\textit{visited}, y)\}, \{(at, x)\})$$

and the set of instances is

$$\begin{aligned}\mathcal{I} = \{\ & (L, \\ & \{(\textit{not-eq}, x, y) \mid x, y \in L, x \neq y\} \cup \{(at, l_0)\}, \\ & \{(\textit{visited}, x) \mid x \in L\}) \\ & \mid L \subseteq \Sigma^+, |L| \geq 2, l_0 \in L\}\end{aligned}$$

The non-equality predicate is *not* contained in the domain file usually used in the community. However the obvious intention behind the move action is that it can only be applied between different locations. Some researchers have restricted operator applications such that all parameters must be pairwise different [16]. In some cases, like here, this is adequate; but in other cases it is not. We have chosen to include non-equality of parameters into the operator definition itself, in those cases where the intuitive meaning of an operator suggests to do so, and non-equality does not follow anyway. This is the case for move operators. Restricting the two locations in a move to be different does not change the semantics of a domain: the action that moves from A to A is useless anyway. But skipping such pointless actions does improve readability.[3]

Obviously, all instances of *Tsp* are solvable.

Corollary 1. *Given an instance* (C, I, G) *of the Tsp domain, the corresponding state space is harmless.*

Proof. Obviously, the state space is not undirected, as visiting a location can not be undone. Furthermore, the actions can be transformed such that they are at least invertible in the sense of Definition 36: move(x, x) actions will never become applicable so these can be skipped; for $x \neq y$, the static non-equality precondition of move(x, y) can be removed; in the transformed set, move(x, y) is at least inverted by move(y, x) in the sense of Definition 36. With Lemma 2, we are finished.

The transformation described in the proof above utilizes, and completely removes, the non-equality constraint that we have introduced artificially. We will apply likewise arguments at all points where we introduce non-equality to restrict operator parameters, and at other points where static predicates can be

[3] *Tsp* is the only of our domains where moving from A to A could make sense: namely when A is the starting location for which (visited, A) has not yet been achieved; nevertheless, it is counter-intuitive to allow such an action.

used to simplify a task description (for example, often operators include static predicates to specify the *types* of the arguments that are allowed). Note that some planners, like IPP and FF, apply a pre-process that detects static predicates, performing the transformation fully automatically [16, 14].

Theorem 1. *Given an instance (C, I, G) of the Tsp domain, the corresponding state space does not contain any local minima under h^+.*

Proof. As all actions are at least invertible, with Lemma 4 it suffices to show that all actions are respected by the relaxation. Now if a move(x, y) starts an optimal plan, then y has not yet been visited, so one optimal relaxed plan starts with the same action.

Theorem 2. *Given an instance (C, I, G) of the Tsp domain, the maximal exit distance in the corresponding state space is 0 under h^+.*

Proof. By the argumentation in the proof to Theorem 1, the instance fulfills the requirements for Lemma 4, so h^+ never increases on optimal solution paths. In any non-goal state s, an optimal plan starts by visiting one yet unvisited location, move(x, y). Say that action starts the optimal relaxed plan P^+ for s. If one applies move(x, y), a relaxed plan for the resulting state s' can be constructed by removing move(x, y) from P^+, and replacing all actions of the form move(x, y') in P^+ with move(y, y'). Thus, $h^+(s') < h^+(s)$, and all non-goal states are exits.

8 Movie

In the *Movie* domain, solving an instance requires getting a number of snacks, rewinding the movie, and resetting the counter.

Definition 41. *The Movie domain is $Movie = (\mathcal{P}, \mathcal{O}, \mathcal{I})$, where the set of predicates considered is*

$\mathcal{P} = \{$ (*movie-rewound*, 0),
(*counter-at-two-hours*, 0),
(*counter-at-other-than-two-hours*, 0),
(*counter-at-zero*, 0),
(*have-chips*, 0),
(*have-dip*, 0),
(*have-pop*, 0),
(*have-cheese*, 0),
(*have-crackers*, 0),
(*chips*, 1),
(*dip*, 1),
(*pop*, 1),
(*cheese*, 1),
(*crackers*, 1) $\}$

$\mathcal{O}$ consists of the operators

– *rewind-movie-2:* $\Sigma^0 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\textit{rewind-movie-2}() := (\{(\textit{counter-at-two-hours})\}, \{(\textit{movie-rewound})\}, \emptyset)$$

– *rewind-movie:* $\Sigma^0 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\textit{rewind-movie}() := (\ & \{(\textit{counter-at-other-than-two-hours})\},\\ & \{(\textit{movie-rewound})\},\\ & \{(\textit{counter-at-zero})\}\ )\end{aligned}$$

– *reset-counter:* $\Sigma^0 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\textit{reset-counter}() := (\emptyset, \{(\textit{counter-at-zero})\}, \emptyset)$$

– *get-chips:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\textit{get-chips}(x) := (\{(\textit{chips}, x)\}, \{(\textit{have-chips})\}, \emptyset)$$

– *get-dip:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\textit{get-dip}(x) := (\{(\textit{dip}, x)\}, \{(\textit{have-dip})\}, \emptyset)$$

– *get-pop:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\textit{get-pop}(x) := (\{(\textit{pop}, x)\}, \{(\textit{have-pop})\}, \emptyset)$$

– *get-cheese:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\textit{get-cheese}(x) := (\{(\textit{cheese}, x)\}, \{(\textit{have-cheese})\}, \emptyset)$$

– *get-crackers:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\textit{get-crackers}(x) := (\{(\textit{crackers}, x)\}, \{(\textit{have-crackers})\}, \emptyset)$$

and the set of instances is

$$\begin{aligned}\mathcal{I} = \{\ & (\ C \cup D \cup P \cup Z \cup K,\\ & \quad \{(\textit{chips}, x) \mid x \in C\} \cup\\ & \quad \{(\textit{dip}, x) \mid x \in D\} \cup\\ & \quad \{(\textit{pop}, x) \mid x \in P\} \cup\\ & \quad \{(\textit{cheese}, x) \mid x \in Z\} \cup\\ & \quad \{(\textit{crackers}, x) \mid x \in K\} \cup\\ & \quad \{(\textit{counter-at-other-than-two-hours})\},\\ & \{\ (\textit{movie-rewound}),\\ & \quad (\textit{counter-at-zero}),\\ & \quad (\textit{have-chips}),\\ & \quad (\textit{have-dip}),\\ & \quad (\textit{have-pop}),\\ & \quad (\textit{have-cheese}),\\ & \quad (\textit{have-crackers})\ \}\ )\\ & \mid\ (C \cup D \cup P \cup Z \cup K) \subseteq \Sigma^+, |C|, |D|, |P|, |Z|, |K| \geq 1\}\end{aligned}$$

A solution plan here simply selects one snack of each type, rewinds the movie, and resets the counter. Note that rewind-movie-2() is not applicable in any reachable state of any instance. All *Movie* instances are solvable.

Corollary 2. *Given an instance (C, I, G) of the Movie domain, the corresponding state space is harmless.*

Proof. The state space is not undirected because getting some snack, for example, can not be undone. All actions produced by one of the getting snacks operators have irrelevant delete effects (none at all) and static add effects in the sense of Definition 37. The rewind-movie() and reset-counter() actions are both at least invertible by reset-counter() in the sense of Definition 36. By applying Lemma 2, we are finished.

Theorem 3. *Given an instance (C, I, G) of the Movie domain, the corresponding state space does not contain any local minima under h^+.*

Proof. All actions produced by one of the getting snacks operators, and the reset-counter() action, have no, and therefore irrelevant, delete effects. The rewind-movie() action is at least relaxed invertible, so with Lemma 4 it suffices to show that rewind-movie() is respected by the relaxation. Now, if rewind-movie() starts an optimal plan, then (movie-rewound) is not yet true, so that action is part of all plans, relaxed or real.

Theorem 4. *Given an instance (C, I, G) of the Movie domain, the maximal exit distance in the corresponding state space is 1 under h^+.*

Proof. By the argumentation in the proof to Theorem 3, the instance fulfills the requirements for Lemma 4, so h^+ never increases on optimal solution paths. All actions except rewind-movie() have no—and therefore irrelevant—deletes, so states s where they start an optimal plan are exits by Proposition 2. Say the only action starting an optimal plan for the state s is rewind-movie(), meaning that one has all snacks already. Then, applying rewind-movie() to s yields a successor state s' where reset-counter() starts an optimal plan (in fact, reset-counter() achieves the goal from s'), so an upper limit to the exit distance is 1. If (counter-at-zero) $\in s$, then no action except rewind-movie() can change s at all, and $h^+(s) = h^+(s') = 1$, so then the exit distance is exactly 1.

9 Gripper

The *Gripper* domain is about transporting a number of balls with two gripper hands from a room A to another room B.

Definition 42. *The Gripper domain is Gripper $= (\mathcal{P}, \mathcal{O}, \mathcal{I})$, where the set of predicates considered is*

$$\mathcal{P} = \{\ (\textit{not-eq}, 2),\ (\textit{room}, 1),\ (\textit{ball}, 1),\ (\textit{gripper}, 1),\ (\textit{at-robby}, 1),\ (\textit{at}, 2),\ (\textit{free}, 1),\ (\textit{carry}, 2)\ \}$$

$\mathcal{O}$ consists of the operators

- *move:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\textit{move}(x, y) := (\ \{(\textit{not-eq}, x, y), (\textit{room}, x), (\textit{room}, y), (\textit{at-robby}, x)\},\ \{(\textit{at-robby}, y)\},\ \{(\textit{at-robby}, x)\}\ )$

- *pick:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\textit{pick}(x, y, z) := (\ \{\ (\textit{ball}, x), (\textit{room}, y), (\textit{gripper}, z),\ (\textit{at}, x, y), (\textit{at-robby}, y), (\textit{free}, z)\ \},\ \{\ (\textit{carry}, x, z)\ \},\ \{\ (\textit{at}, x, y), (\textit{free}, z)\ \}\ )$

- *drop:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\textit{drop}(x, y, z) := (\ \{\ (\textit{ball}, x), (\textit{room}, y), (\textit{gripper}, z),\ (\textit{carry}, x, z), (\textit{at-robby}, y)\ \},\ \{\ (\textit{at}, x, y), (\textit{free}, z)\ \},\ \{\ (\textit{carry}, x, z)\ \}\ )$

and the set of instances is

$$\begin{aligned}\mathcal{I} = \{\ (\ & \{a, b, l, r\} \cup B,\\ & \{(\textit{not-eq}, a, b), (\textit{not-eq}, b, a)\} \cup\\ & \{(\textit{room}, a), (\textit{room}, b), (\textit{gripper}, l), (\textit{gripper}, r)\} \cup\\ & \{(\textit{ball}, x) \mid x \in B\} \cup\\ & \{(\textit{free}, l), (\textit{free}, r)\} \cup\\ & \{(\textit{at}, x, a) \mid x \in B\} \cup\\ & \{(\textit{at-robby}, a)\},\\ & \{(\textit{at}, x, b) \mid x \in B\}\ )\\ \mid\ & (\{a, b, l, r\} \cup B) \subseteq \Sigma^+, |B| \geq 1\ \}\end{aligned}$$

Obviously, all instances of the *Gripper* domain are solvable.

Corollary 3. *Given an instance (C, I, G) of the Gripper domain, the corresponding state space is undirected.*

Proof. After transformation with respect to static predicates, all actions are invertible in the sense of Definition 35: $\text{pick}(x, y, z)$ is inverted by $\text{drop}(x, y, z)$ and vice versa, and $\text{move}(x, y)$ is inverted by $\text{move}(y, x)$. By applying Lemma 1, we are finished.

Theorem 5. *Given an instance (C, I, G) of the Gripper domain, the corresponding state space does not contain any local minima under h^+.*

Proof. As shown in the proof to Corollary 3 above, all actions are invertible, so with Lemma 4 it suffices to show that all actions are respected by the relaxation. If the optimal starting action in a reachable state s is a $\text{pick}(x, y, z)$ or a $\text{drop}(x, y, z)$, this is obvious because then the ball x is not yet in room b, and must be transported. If the optimal starting action is a $\text{move}(x, y)$, then there are either balls left to pick in $y = a$, or balls left to drop in $y = b$, so a relaxed plan from s has no choice but applying the respective move action.

Theorem 6. *Given an instance (C, I, G) of the Gripper domain, the maximal exit distance in the corresponding state space is at most 1 under h^+.*

Proof. By the argumentation in the proof to Theorem 5, the instance fulfills the requirements for Lemma 4, so h^+ never increases on optimal solution paths. Say we have a reachable state s and an optimal starting action *act* (we use that name to avoid confusion with the *room a* used in the instance), yielding the successor state s'. Say also that *act* starts the optimal relaxed plan P^+ for s. If $act = \text{drop}(x, y, z)$ then $y = b$, and removing *act* from P^+ yields a relaxed plan for s' as x is in its goal position and (carry, x, z) is not needed anymore; thus in that case s is an exit. If $act = \text{pick}(x, y, z)$ then $y = a$, and P^+ also contains the actions $\text{move}(a, b)$ and $\text{drop}(x, b, z)$; a relaxed plan for s' can be constructed by removing *act* from P^+ and placing the two other actions up front: that way, (free, z) is re-achieved before any other action is applied; the deletion of (at, x, a) does not matter as a is picked up only once due to the optimality of P^+. So in that case, s is also an exit. Finally, consider the case where $act = \text{move}(x, y)$. Then, the successor state s' falls into one of the two above cases, so an upper limit to the exit distance is 1.

If, in the situation described at the end of the proof to Theorem 6, picks or drops are left to do in room x, then the relaxed plan from s' must contain the inverse action $\text{move}(y, x)$, so then the exit distance is exactly 1.

10 Ferry

In *Ferry*, cars are transported via a ferry that can only carry one car at a time.

Definition 43. *The Ferry domain is $Ferry = (\mathcal{P}, \mathcal{O}, \mathcal{I})$, where the set of predicates considered is*

$\mathcal{P} = \{$ $(\textit{not-eq}, 2)$,
$(\textit{car}, 1)$,
$(\textit{location}, 1)$,
$(\textit{at-ferry}, 1)$,
$(\textit{at}, 2)$,
$(\textit{empty-ferry}, 0)$,
$(\textit{on}, 1)$ $\}$

$\mathcal{O}$ *consists of the operators*

- *sail:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \textit{sail}(x,y) := (\ & \{(\textit{not-eq},x,y),(\textit{location},x),(\textit{location},y),(\textit{at-ferry},x)\},\\ & \{(\textit{at-ferry},y)\},\\ & \{(\textit{at-ferry},x)\}\ ) \end{aligned}$$

- *board:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \textit{board}(x,y) := (\ & \{\ (\textit{car},x),(\textit{location},y),\\ & \quad (\textit{at},x,y),(\textit{at-ferry},y),(\textit{empty-ferry})\ \},\\ & \{(\textit{on},x)\},\\ & \{(\textit{at},x,y),(\textit{empty-ferry})\}\ ) \end{aligned}$$

- *debark:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \textit{debark}(x,y) := (\ & \{\ (\textit{car},x),(\textit{location},y),\\ & \quad (\textit{on},x),(\textit{at-ferry},y)\ \},\\ & \{(\textit{at},x,y),(\textit{empty-ferry})\},\\ & \{(\textit{on},x)\}\ ) \end{aligned}$$

and the set of instances is

$$\begin{aligned} \mathcal{I} = \{\ & (\ C \cup L,\\ & \quad \{(\textit{not-eq},x,y) \mid x,y \in L, x \neq y\} \cup\\ & \quad \{(\textit{car},x) \mid x \in C\} \cup\\ & \quad \{(\textit{location},x) \mid x \in L\} \cup\\ & \quad \{(\textit{at-ferry},il)\} \cup \{(\textit{at},x,\textit{cil}(x)) \mid x \in C\},\\ & \quad \{(\textit{at},x,\textit{cgl}(x)) \mid x \in C'\}\ )\\ & \mid\ (C \cup L) \subseteq \Sigma^+, |L| \geq 1,\\ & \quad il \in L, \textit{cil} : C \mapsto L,\\ & \quad C' \subseteq C, \textit{cgl} : C' \mapsto L\ \} \end{aligned}$$

In the instances, *il* is the initial location of the ferry, and *cil* is the car initial location function maping the cars to their respective initial locations. Likewise, *cgl* maps cars to their goal locations, where only the subset C' of all cars is required to be at a specific location to fulfill the goal. Obviously, all *Ferry* instances are solvable. Proving the hypothesized properties proceeds in almost exactly the same way as in the *Gripper* domain.

Corollary 4. *Given an instance* (C, I, G) *of the* $Ferry$ *domain, the corresponding state space is undirected.*

Proof. After transformation with respect to static predicates, all actions are invertible in the sense of Definition 35: board(x, y) is inverted by debark(x, y) and vice versa, and sail(x, y) is inverted by sail(y, x). By applying Lemma 1, we are finished.

Theorem 7. *Given an instance (C, I, G) of the Ferry domain, the corresponding state space does not contain any local minima under h^+.*

Proof. As shown in the proof to Corollary 4 above, all actions are invertible, so with Lemma 4 it suffices to show that all actions are respected by the relaxation. If the optimal starting action in a reachable state s is a board(x, y) or a debark(x, y), this is obvious because then the car x is not yet at its goal location, and must be transported. If the optimal starting action is a sail(x, y), then at location y there is either a car to board, or a car to debark, so a relaxed plan from s will have to visit that location; when this is done does not matter in the relaxation—the (at-ferry, y) fact remains true throughout relaxed plan execution—so one can start an optimal relaxed plan by sailing the ferry to y.

Theorem 8. *Given an instance (C, I, G) of the Ferry domain, the maximal exit distance in the corresponding state space is at most 1 under h^+.*

Proof. By the argumentation in the proof to Theorem 7, the instance fulfills the requirements for Lemma 4, so h^+ never increases on optimal solution paths. Say we have a reachable state s and an optimal starting action a, yielding the successor state s'. Say also that a starts the optimal relaxed plan P^+ for s. If $a =$ debark(x, y) then either y is the goal location of x, or x has no goal location ($x \notin C'$), so removing a from P^+ yields a relaxed plan for s' as (on, x) is not needed anymore; thus in that case s is an exit. If $a =$ board(x, y) then x must be transported, and P^+ also contains the actions sail(y, y') and debark(x, y'), where y' is the goal location of x; a relaxed plan for s' can be constructed by removing a from P^+ and placing the two other actions up front: that way, (empty-ferry) is re-achieved before any other action is applied; the deletion of (at, x, y) does not matter as a is boarded only once due to the optimality of P^+. So in that case, s is also an exit. Finally, consider the case where the optimal starting action is $a =$ sail(x, y). Then, the successor state s' falls into one of the two above cases—some car is boarded or debarked—so an upper limit to the exit distance is 1.

If, in the situation described at the end of the proof to Theorem 8, a car is left at x which must be boarded later on, or some other car must still be transported to x, then the relaxed plan from s' must contain the inverse action sail(y, x), so then the exit distance is exactly 1.

11 Logistics

In *Logistics*, objects must be transported within cities using trucks, and between cities using airplanes. No fuel restrictions apply, and road maps are fully connected.

Definition 44. *The Logistics domain is* $Logistics = (\mathcal{P}, \mathcal{O}, \mathcal{I})$, *where the set of predicates considered is*

$$\begin{aligned}\mathcal{P} = \{\ & (\textit{not-eq}, 2),\\ & (\textit{obj}, 1),\\ & (\textit{truck}, 1),\\ & (\textit{location}, 1),\\ & (\textit{airplane}, 1),\\ & (\textit{city}, 1),\\ & (\textit{airport}, 1),\\ & (\textit{at}, 2),\\ & (\textit{in}, 2),\\ & (\textit{in-city}, 2)\ \}\end{aligned}$$

$\mathcal{O}$ *consists of the operators*

- *drive-truck:* $\Sigma^4 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\textit{drive-truck}(x, y, z, v) := (\ & \{\ (\textit{truck}, x), (\textit{not-eq}, y, z), (\textit{location}, y), (\textit{location}, z), (\textit{city}, v),\\ & \quad (\textit{in-city}, y, v), (\textit{in-city}, z, v), (\textit{at}, x, y)\ \},\\ & \{\ (\textit{at}, x, z)\ \},\\ & \{\ (\textit{at}, x, y)\ \}\ )\end{aligned}$$

- *fly-airplane:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\textit{fly-airplane}(x, y, z) := (\ & \{\ (\textit{airplane}, x), (\textit{not-eq}, y, z), (\textit{airport}, y), (\textit{airport}, z),\\ & \quad (\textit{at}, x, y)\ \},\\ & \{\ (\textit{at}, x, z)\ \},\\ & \{\ (\textit{at}, x, y)\ \}\ )\end{aligned}$$

- *load-truck:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\textit{load-truck}(x, y, z) := (\ & \{\ (\textit{obj}, x), (\textit{truck}, y), (\textit{location}, z),\\ & \quad (\textit{at}, x, z), (\textit{at}, y, z)\ \},\\ & \{\ (\textit{in}, x, y)\ \},\\ & \{\ (\textit{at}, x, z)\ \}\ )\end{aligned}$$

- *load-airplane:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\textit{load-airplane}(x, y, z) := (\ & \{\ (\textit{obj}, x), (\textit{airplane}, y), (\textit{location}, z),\\ & \quad (\textit{at}, x, z), (\textit{at}, y, z)\ \},\\ & \{\ (\textit{in}, x, y)\ \},\\ & \{\ (\textit{at}, x, z)\ \}\ )\end{aligned}$$

- *unload-truck:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\textit{unload-truck}(x, y, z) := (\ & \{\ (\textit{obj}, x), (\textit{truck}, y), (\textit{location}, z),\\ & \quad (\textit{in}, x, y), (\textit{at}, y, z)\ \},\\ & \{\ (\textit{at}, x, z)\ \},\\ & \{\ (\textit{in}, x, y)\ \}\ )\end{aligned}$$

– *unload-airplane:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \textit{unload-airplane}(x,y,z) := (\ &\{\ (\textit{obj},x),(\textit{airplane},y),(\textit{location},z),\\ &\quad (\textit{in},x,y),(\textit{at},y,z)\ \},\\ &\{\ (\textit{at},x,z)\ \},\\ &\{\ (\textit{in},x,y)\ \}\ ) \end{aligned}$$

and the set of instances is

$$\begin{aligned} \mathcal{I} = \{\ (\ & O \cup T \cup A \cup C \cup L_1 \cup \ldots \cup L_n, \textit{where } C = \{c_1,\ldots,c_n\},\\ & \{(\textit{not-eq},x,y) \mid x,y \in L_1 \cup \ldots \cup L_n, x \neq y\} \cup\\ & \{(\textit{obj},x) \mid x \in O\} \cup\\ & \{(\textit{truck},x) \mid x \in T\} \cup\\ & \{(\textit{airplane},x) \mid x \in A\} \cup\\ & \{(\textit{city},x) \mid x \in C\} \cup\\ & \{(\textit{location},x) \mid x \in L_1 \cup \ldots \cup L_n\} \cup\\ & \{(\textit{in-city},x,c_i) \mid x \in L_i\} \cup\\ & \{(\textit{airport},l_1),\ldots,(\textit{airport},l_n)\} \cup\\ & \{(\textit{at},x,\textit{il}(x)) \mid x \in O \cup T \cup A\},\\ & \{(\textit{at},x,\textit{gl}(x)) \mid x \in O'\}\ )\\ \mid\ & (O \cup T \cup A \cup C \cup L_1 \cup \ldots \cup L_n) \subseteq \Sigma^+,\\ & |C| \geq 1, |L_i| \geq 1, |T| \leq |C|, l_i \in L_i,\\ & \textit{il} : O \cup T \cup A \mapsto L_1 \cup \ldots \cup L_n,\\ & \forall\ x \neq x' \in T : \textit{il}(x) \in L_i \Rightarrow \textit{il}(x') \notin L_i, \forall\ x \in A : \textit{il}(x) \in \{l_1,\ldots,l_n\},\\ & O' \subseteq O, \textit{gl} : O' \mapsto L_1 \cup \ldots \cup L_n\ \} \end{aligned}$$

In the definition of the instances, *il* and *gl* are the initial location and goal location functions, respectively. In words, there is at least one city, each city has at least one location, there is at most one truck in each city, airplanes are only allowed at airports (of which each city has exactly one), and only a subset of the transportable objects need to have goal locations specified. A *Logistics* instance can be unsolvable when a package can not be transported to its goal location; this can happen when in one of the relevant cities there is no truck, or when there is no airplane.

Corollary 5. *Given an instance* (C,I,G) *of the Logistics domain, the corresponding state space is undirected.*

Proof. After transformation with respect to static predicates, all actions are invertible in the sense of Definition 35: load-truck(x,y,z) actions are undone by unload-truck(x,y,z), symmetrically for other loading or unloading actions; drive-truck(x,y,z,v) is undone by drive-truck(x,z,y,v), symmetrically for fly-airplane actions. By applying Lemma 1, we are finished.

Theorem 9. *Given a solvable instance* (C,I,G) *of the Logistics domain, the corresponding state space does not contain any local minima under* h^+*.*

Proof. As seen in the proof to Corollary 5 above, all actions are invertible, so with solvability (implying the non-existence of dead ends) and Lemma 4 it suffices

to show that all actions are respected by the relaxation. If the optimal starting action a in a reachable state s is any loading or unloading action, then a is respected by the relaxation because in that case the respective package is not yet at its goal location, so it must be transported, and it is best—for relaxed as well as for real plans—to do that when an adequate vehicle is already in position. If the optimal starting action a is a driving or flying action to some location z, then some package must either be loaded or unloaded at z, so a relaxed plan from s has no choice but applying some action moving a transportation vehicle (of a's kind) there. All vehicles are equally good, except when there is a clever choice, i.e., a vehicle that already carries packages to be unloaded at z; but then, a will move one of those vehicles just like an optimal relaxed plan will, and all such vehicles are equally good in the relaxation.

Theorem 10. *Given a solvable instance (C, I, G) of the Logistics domain, the maximal exit distance in the corresponding state space is at most 1 under h^+.*

Proof. By the argumentation in the proof to Theorem 9, the instance fulfills the requirements for Lemma 4, so h^+ never increases on optimal solution paths. Say we have a reachable state s and an optimal starting action a, yielding the successor state s'. Say also that a starts the optimal relaxed plan P^+ for s. If a is a loading (unloading) action, its only delete is the at (in) relation of the transported object; as the object is loaded from the respective location (unloaded from the respective vehicle) only once in P^+, removing a from P^+ yields a relaxed plan for s', so s is an exit. Otherwise, consider the case where a drives or flies some vehicle x from y to z. Then s' is an exit: no optimal plan moves the same vehicle twice in a row—all locations are immediately accessible, if accessible at all—so one optimal plan for s' starts by loading (unloading) some package to (from) x. Thus an upper limit to the exit distance is 1.

If, for example, in the situation described at the end of the proof to Theorem 10, x is a truck, y is the airport, and the package must be transported to a different city, then the relaxed plan from s' must include the action driving x back from z to y, so then the exit distance is exactly 1.

12 Miconic-STRIPS

In the *Miconic-STRIPS* domain, a lift shall transport a collection of passengers from their origin- to their destination-floors.

Definition 45. *The Miconic-STRIPS domain is $Miconic\text{-}STRIPS = (\mathcal{P}, \mathcal{O}, \mathcal{I})$, where the set of predicates considered is*

$$\begin{aligned}\mathcal{P} = \{\ & (\mathit{passenger}, 1),\\ & (\mathit{floor}, 1),\\ & (\mathit{origin}, 2),\\ & (\mathit{destin}, 2),\\ & (\mathit{above}, 2),\\ & (\mathit{boarded}, 1),\\ & (\mathit{served}, 1),\\ & (\mathit{lift\text{-}at}, 1)\ \}\end{aligned}$$

$\mathcal{O}$ consists of the operators

- *board:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\mathit{board}(x,y) := (\ & \{\ (\mathit{floor}, x), (\mathit{passenger}, y),\\ & \quad (\mathit{lift\text{-}at}, x), (\mathit{origin}, y, x)\ \},\\ & \{\ (\mathit{boarded}, y)\ \},\\ & \quad \emptyset\ )\end{aligned}$$

- *depart:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\mathit{depart}(x,y) := (\ & \{\ (\mathit{floor}, x), (\mathit{passenger}, y),\\ & \quad (\mathit{lift\text{-}at}, x), (\mathit{destin}, y, x), (\mathit{boarded}, y)\ \},\\ & \{\ (\mathit{served}, y)\ \},\\ & \{\ (\mathit{boarded}, y)\ \}\ )\end{aligned}$$

- *up:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\mathit{up}(x,y) := (\ & \{\ (\mathit{floor}, x), (\mathit{floor}, y),\\ & \quad (\mathit{lift\text{-}at}, x), (\mathit{above}, x, y)\ \},\\ & \{\ (\mathit{lift\text{-}at}, y)\ \},\\ & \{\ (\mathit{lift\text{-}at}, x)\ \}\ )\end{aligned}$$

- *down:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\mathit{down}(x,y) := (\ & \{\ (\mathit{floor}, x), (\mathit{floor}, y),\\ & \quad (\mathit{lift\text{-}at}, x), (\mathit{above}, y, x)\ \},\\ & \{\ (\mathit{lift\text{-}at}, y)\ \},\\ & \{\ (\mathit{lift\text{-}at}, x)\ \}\ )\end{aligned}$$

and the set of instances is

$$\begin{aligned}\mathcal{I} = \{\ & (\ P \cup F, \textit{where } F = \{f_1, \ldots, f_n\},\\ & \{(\mathit{passenger}, x) \mid x \in P\} \cup\\ & \{(\mathit{floor}, x) \mid x \in F\} \cup\\ & \{(\mathit{above}, f_i, f_j) \mid i < j\} \cup\\ & \{(\mathit{origin}, x, of(x)) \mid x \in P\} \cup\\ & \{(\mathit{destin}, x, df(x)) \mid x \in P\} \cup\\ & \{(\mathit{lift\text{-}at}, if)\},\\ & \{(\mathit{served}, x) \mid x \in P\}\ )\\ & \mid (P \cup F) \subseteq \Sigma^+, |F| \geq 1,\\ & \quad of : P \mapsto F, df : P \mapsto F, if \in F\ \}\end{aligned}$$

In the definition of the instances, of and df are the origin and destination floors of the passengers, and if is the floor where the lift is initially positioned. Obviously, all $Miconic\text{-}STRIPS$ instances are solvable.

Proposition 3. *Given an instance (C, I, G) of the $Miconic\text{-}STRIPS$ domain, the corresponding state space is harmless.*

Proof. Obviously, the state space is not undirected, as boarding a passenger, for example, can not be undone. On the other hand, there are no dead ends, which can be shown by a slight extension of the proof to Lemma 2. Actions moving the lift up and down are invertible—by each other—in the sense of Definition 35. depart(x, y) actions have static add effects in the sense of Definition 36; their delete effects are not strictly irrelevant according to that definition, as all departing actions share, and delete, the precondition (boarded, y); however, only a single departing action—at the respective passenger's destination floor, namely—will ever become applicable, so that action's delete is irrelevant in the set of all possible actions. Finally, board(x, y) actions have irrelevant (no) deletes; their add effect, (boarded, y), is deleted by the respective departing action; however, that action will not be applied in the first loop described in the proof to Lemma 2—it does not invert any other action—and in the second loop it will be applied *after* board(x, y), and at most once if the original plan from the initial state is assumed optimal—the precondition (boarded, y) can only be made true by board(x, y), and departing adds a single fact that is static. Altogether, thus, the argumentation in the proof to Lemma 2 remains valid.

There are surely simpler ways than the proof given above to see that $Miconic\text{-}STRIPS$ instances are dead-end free. We remark, however, that the properties extending Lemma 2 to the $Miconic\text{-}STRIPS$ domain are purely syntactical:

- that no departing action except that at the respective passenger's destination floor will ever become applicable can be found out by checking static predicates—(destin, y, x) is not added by any action;
- that no departing action can be used to invert any other action is a purely syntactical property;
- the same holds for the precondition (boarded, y) of departing being only be made true by board(x, y), and the add effects of departing being static.

Thus, an appropriate—however much more dificult to read—extension of Lemma 2 defines a syntactical way of recognizing at most harmless planning tasks, including those in the $Miconic\text{-}STRIPS$ domain.

Theorem 11. *Given an instance (C, I, G) of the $Miconic\text{-}STRIPS$ domain, the corresponding state space does not contain any local minima under h^+.*

Proof. As the argumentation to Proposition 3 has shown, all actions are either invertible or have irrelevant deletes in the set of all actions that can ever become applicable. Therefore, with Lemma 4 it suffices to show that all actions are

respected by the relaxation. For boarding and departing actions, this is obvious. For up(x, y) and down(x, y) actions, if an optimal plan starts with such an action, then at least one passenger must board or depart at floor y, so any relaxed plan will have to visit that floor; when this is done does not matter in the relaxation—the (lift-at, y) fact remains true throughout relaxed plan execution—so one can start an optimal relaxed plan by moving the lift to y.

Theorem 12. *Given an instance (C, I, G) of the Miconic-STRIPS domain, the maximal exit distance in the corresponding state space is at most 1 under h^+.*

Proof. By the argumentation in the proof to Theorem 11, the instance fulfills the requirements for Lemma 4, so h^+ never increases on optimal solution paths. Say we have a reachable state s and an optimal starting action a, yielding the successor state s'. Say also that a starts the optimal relaxed plan P^+ for s. As seen above, the boarding and departing actions have irrelevant deletes (in the set of actions that can become applicable), so if a is one of those, then s is an exit by (a slight extension of) Proposition 2. In the other case, if a is an up or down action, then s' falls into the first case because no optimal plan moves the lift twice in a row. So an upper limit to the exit distance is 1.

If, for example, in the situation described at the end of the proof to Theorem 12, there are two floors, the lift is in the bottom floor in s, and the single passenger wants to travel from the top to the bottom floor, then the relaxed plan from s' must include the action moving the lift back from the top to the bottom floor, so then the exit distance is exactly 1.

13 Miconic-SIMPLE

Miconic-SIMPLE is an ADL domain. A lift shall transport a collection of passengers from their origin- to their destination-floors. Whenever the lift stops at a floor, all passengers that are waiting (boarded) get in (out) by a conditional effect.

Definition 46. *The Miconic-SIMPLE domain is $Miconic\text{-}SIMPLE = (\mathcal{P}, \mathcal{O}, \mathcal{I})$, where the set of predicates considered is*

$\mathcal{P} = \{$ (*passenger*, 1),
(*floor*, 1),
(*origin*, 2),
(*destin*, 2),
(*above*, 2),
(*boarded*, 1),
(*served*, 1),
(*not-served*, 1),
(*lift-at*, 1) $\}$

$\mathcal{O}$ consists of the operators

– *stop:* $\Sigma \times 2^{\Sigma} \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} stop(x, C) := (\ & (floor, x) \wedge (lift\text{-}at, x), \\ & \{\ ((passenger, y) \wedge (boarded, y) \wedge (destin, y, x), \\ & \quad \{(served, y)\}, \{(boarded, y), (not\text{-}served, y)\}) \mid y \in C\ \} \cup \\ & \{\ ((passenger, y) \wedge (not\text{-}served, y) \wedge (origin, y, x), \\ & \quad \{(boarded, y)\}, \emptyset) \mid y \in C\ \}\ ) \end{aligned}$$

– *up:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} up(x, y) := (\ & \{\ (floor, x), (floor, y), \\ & \quad (lift\text{-}at, x), (above, x, y)\ \}, \\ & \{\ (lift\text{-}at, y)\ \}, \\ & \{\ (lift\text{-}at, x)\ \}\ ) \end{aligned}$$

– *down:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} down(x, y) := (\ & \{\ (floor, x), (floor, y), \\ & \quad (lift\text{-}at, x), (above, y, x)\ \}, \\ & \{\ (lift\text{-}at, y)\ \}, \\ & \{\ (lift\text{-}at, x)\ \}\ ) \end{aligned}$$

and the set of instances is

$$\begin{aligned} \mathcal{I} = \{\ & (\ P \cup F, where\ F = \{f_1, \ldots, f_n\}, \\ & \quad \{(passenger, x) \mid x \in P\} \cup \\ & \quad \{(floor, x) \mid x \in F\} \cup \\ & \quad \{(above, f_i, f_j) \mid i < j\} \cup \\ & \quad \{(not\text{-}served, x) \mid x \in P\} \cup \\ & \quad \{(origin, x, of(x)) \mid x \in P\} \cup \\ & \quad \{(destin, x, df(x)) \mid x \in P\} \cup \\ & \quad \{(lift\text{-}at, if)\}, \\ & \quad \bigwedge\{(served, x) \mid x \in P\}\ ) \\ & \mid (P \cup F) \subseteq \Sigma^{+}, |F| \geq 1, \\ & \quad of : P \mapsto F, df : P \mapsto F, if \in F\ \} \end{aligned}$$

Like in $Miconic\text{-}STRIPS$, instances are defined using the initial floor if, and the origin and destination floor functions of and df. All $Miconic\text{-}SIMPLE$ instances are solvable.

Proposition 4. *Given an instance (C, I, G) of the $Miconic\text{-}SIMPLE$ domain, the corresponding state space is harmless.*

Proof. Obviously, the state space is not undirected, as the effects of stopping can not be undone. The actions moving the lift up and down are invertible. The deletes of the stopping effects are all irrelevant in the sense that they are not needed anymore once those effects have occured. One can thus, from any reachable state s, construct a plan by inverting all lift moves that have been made on the way to s, and executing a plan that solves the task from the initial state.

Theorem 13. *Given an instance (C, I, G) of the Miconic-SIMPLE domain, the corresponding state space does not contain any local minima under h^+.*

Proof. Like for the STRIPS domains seen before, we prove that the heuristic value does not increase on optimal solution paths. The $\text{up}(x, y)$ and $\text{down}(x, y)$ actions are respected by the relaxation: if an optimal plan starts with such an action, then at least one passenger must board or depart at floor y, so any relaxed plan will have to visit that floor, and one can start an optimal relaxed plan by moving the lift to y. Also, the moving actions are invertible and thus do not increase the value of the heuristic function with Lemma 3. Say $a = \text{stop}(x, C)$ starts an optimal plan for some reachable state s, and P^+ is an optimal relaxed plan for s. Then, P^+ is still a relaxed plan for the state s' that results from applying a: the only deletes that can occur concern passengers that reach their goal destination in s'.

Theorem 14. *Given an instance (C, I, G) of the Miconic-SIMPLE domain, the maximal exit distance in the corresponding state space is at most 1 under h^+.*

Proof. Say s is a reachable state. We distinguish between different cases for actions starting an optimal *relaxed* plan for s. If there is such a plan P^+ that starts with $a = \text{stop}(x, C)$, then removing a from P^+ yields a relaxed plan for $\mathit{Result}(s, \langle a \rangle)$: like observed above, the only deletes that can occur concern passengers that have then reached their goal destination. So s is an exit in that case. Otherwise, if all optimal relaxed starting actions are lift moves, we choose one such plan P^+ and move to the floor where the first stopping action, $\text{stop}(x, C)$, in P^+ occurs—without loss of generality, the respective lift move starts P^+, followed directly by $\text{stop}(x, C)$. Now, either the lift move can be skipped from P^+ to obtain a relaxed plan for the successor state—in which case s is an exit—or the inverse lift move can be inserted after $\text{stop}(x, C)$, yielding an optimal relaxed plan for the successor state, which then falls into the first case. So an upper limit to the exit distance is 1.

In the same situation that is described for the *Miconic-STRIPS* domain below Theorem 12, the exit distance is exactly 1.

14 Tyreworld

In *Tyreworld*, a number of flat tyres need to be replaced, using a collection of tools.

Definition 47. *The Tyreworld domain is $\mathit{Tyreworld} = (\mathcal{P}, \mathcal{O}, \mathcal{I})$, where the set of predicates considered is*

$\mathcal{P} = \{\ (wheel, 1),$
$(nut, 1),$
$(obj, 1),$
$(hub, 1),$
$(open\text{-}boot, 0),$
$(closed\text{-}boot, 0),$
$(in\text{-}boot, 1),$
$(have, 1),$
$(intact, 1),$
$(inflated, 1),$
$(not\text{-}inflated, 1),$
$(loose, 2),$
$(tight, 2),$
$(on\text{-}ground, 1),$
$(not\text{-}on\text{-}ground, 1),$
$(fastened, 1),$
$(not\text{-}fastened, 1),$
$(free, 1),$
$(on, 2)\ \}$

$\mathcal{O}$ *consists of the operators*

- *open-boot:* $\Sigma^0 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $open\text{-}boot() := (\ \{\ (closed\text{-}boot)\ \},$
  $\{\ (open\text{-}boot)\ \},$
  $\{\ (closed\text{-}boot)\ \}\ )$

- *close-boot:* $\Sigma^0 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $close\text{-}boot() := (\ \{\ (open\text{-}boot)\ \},$
  $\{\ (closed\text{-}boot)\ \},$
  $\{\ (open\text{-}boot)\ \}\ )$

- *fetch:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $fetch(x) := (\ \{\ (obj, x),$
  $(in\text{-}boot, x), (open\text{-}boot)\ \},$
  $\{\ (have, x)\ \},$
  $\{\ (in\text{-}boot, x)\ \}\ )$

- *put-away:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $put\text{-}away(x) := (\ \{\ (obj, x),$
  $(have, x), (open\text{-}boot)\ \},$
  $\{\ (in\text{-}boot, x)\ \},$
  $\{\ (have, x)\ \}\ )$

- *inflate:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $inflate(x) := (\ \{\ (wheel, x),$
  $(have, pump), (not\text{-}inflated, x), (intact, x)\ \},$
  $\{\ (inflated, x)\ \},$
  $\{\ (not\text{-}inflated, x)\ \}\ )$

– *loosen:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{loosen}(x,y) := (\ &\{\ (\mathit{nut},x), (\mathit{hub},y),\\ &\quad (\mathit{have},\mathit{wrench}), (\mathit{tight},x,y), (\mathit{on\text{-}ground},y)\ \},\\ &\{\ (\mathit{loose},x,y)\ \},\\ &\{\ (\mathit{tight},x,y)\ \}\ ) \end{aligned}$$

– *tighten:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{tighten}(x,y) := (\ &\{\ (\mathit{nut},x), (\mathit{hub},y),\\ &\quad (\mathit{have},\mathit{wrench}), (\mathit{loose},x,y), (\mathit{on\text{-}ground},y)\ \},\\ &\{\ (\mathit{tight},x,y)\ \},\\ &\{\ (\mathit{loose},x,y)\ \}\ ) \end{aligned}$$

– *jack-up:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{jack\text{-}up}(x) := (\ &\{\ (\mathit{hub},x),\\ &\quad (\mathit{have},\mathit{jack}), (\mathit{on\text{-}ground},x)\ \},\\ &\{\ (\mathit{not\text{-}on\text{-}ground},x)\ \},\\ &\{\ (\mathit{have},\mathit{jack}), (\mathit{on\text{-}ground},x)\ \}\ ) \end{aligned}$$

– *jack-down:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{jack\text{-}down}(x) := (\ &\{\ (\mathit{hub},x),\\ &\quad (\mathit{not\text{-}on\text{-}ground},x)\ \},\\ &\{\ (\mathit{have},\mathit{jack}), (\mathit{on\text{-}ground},x)\ \},\\ &\{\ (\mathit{not\text{-}on\text{-}ground},x)\ \}\ ) \end{aligned}$$

– *undo:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{undo}(x,y) := (\ &\{\ (\mathit{nut},x), (\mathit{hub},y),\\ &\quad (\mathit{have},\mathit{wrench}), (\mathit{fastened},y), (\mathit{loose},x,y), (\mathit{not\text{-}on\text{-}ground},y)\ \},\\ &\{\ (\mathit{have},x), (\mathit{not\text{-}fastened},y)\ \},\\ &\{\ (\mathit{fastened},y), (\mathit{loose},x,y)\ \}\ ) \end{aligned}$$

– *do-up:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{do\text{-}up}(x,y) := (\ &\{\ (\mathit{nut},x), (\mathit{hub},y),\\ &\quad (\mathit{have},\mathit{wrench}), (\mathit{have},x), (\mathit{not\text{-}fastened},y), (\mathit{not\text{-}on\text{-}ground},y)\ \},\\ &\{\ (\mathit{fastened},y), (\mathit{loose},x,y)\ \},\\ &\{\ (\mathit{have},x), (\mathit{not\text{-}fastened},y)\ \}\ ) \end{aligned}$$

– *remove-wheel:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{remove\text{-}wheel}(x,y) := (\ &\{\ (\mathit{wheel},x), (\mathit{hub},y),\\ &\quad (\mathit{on},x,y), (\mathit{not\text{-}on\text{-}ground},y), (\mathit{not\text{-}fastened},y)\ \},\\ &\{\ (\mathit{have},x), (\mathit{free},y)\ \},\\ &\{\ (\mathit{on},x,y)\ \}\ ) \end{aligned}$$

– *put-on-wheel:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{put\text{-}on\text{-}wheel}(x,y) := (\ &\{\ (\mathit{wheel},x), (\mathit{hub},y),\\ &\quad (\mathit{have},x), (\mathit{free},y), (\mathit{not\text{-}on\text{-}ground},y), (\mathit{not\text{-}fastened},y)\ \},\\ &\{\ (\mathit{on},x,y)\ \},\\ &\{\ (\mathit{have},x), (\mathit{free},y)\ \}\ ) \end{aligned}$$

and the set of instances is

$$\begin{aligned}\mathcal{I} = \{\ (\ &\{pump, wrench, jack\} \cup N \cup H \cup W,\\ &\textit{where } N = \{n_1, \ldots, n_n\}, H = \{h_1, \ldots, h_n\}, W = \{w_1, s_1, \ldots, w_n, s_n\},\\ &\{(obj, pump), (obj, wrench), (obj, jack)\} \cup\\ &\{(nut, x) \mid x \in N\} \cup\\ &\{(hub, x) \mid x \in H\} \cup\\ &\{(wheel, x) \mid x \in W\} \cup\\ &\{(obj, x) \mid x \in N \cup W\} \cup\\ &\{(on, w_i, h_i), (on\text{-}ground, h_i), (tight, n_i, h_i), (fastened, h_i) \mid 1 \leq i \leq n\} \cup\\ &\{(intact, s_i), (in\text{-}boot, s_i), (not\text{-}inflated, s_i) \mid 1 \leq i \leq n\} \cup\\ &\{(in\text{-}boot, pump), (in\text{-}boot, wrench), (in\text{-}boot, jack), (closed\text{-}boot)\},\\ &\{(on, s_i, h_i), (inflated, s_i), (tight, n_i, h_i), (in\text{-}boot, w_i) \mid 1 \leq i \leq n\} \cup\\ &\{(in\text{-}boot, pump), (in\text{-}boot, wrench), (in\text{-}boot, jack), (closed\text{-}boot)\}\ )\\ \mid\ &\{pump, wrench, jack\} \cup N \cup H \cup W \subseteq \Sigma^+, n \geq 1\ \}\end{aligned}$$

All instances share the constants pump, wrench, and jack, i.e., the tools required for performing the different woeking steps. The wheels w_i shall be replaced with the respective spare wheels s_i. Replacing one wheel involves (in order of appearance of the respective operators) opening the boot, fetching the tools, inflating the spare, loosening the nuts, jacking up the hub, undoing the nuts, removing the wheel, and putting on the spare. Afterwards, the whole process must be reversed, i.e., the nuts must be done up, the hub must be jacked down, and the nuts must be tightened. Finally, put the wheel and all tools into the boot, and close the boot. All instances of *Tyreworld* are solvable, by replacing all wheels in turn.

Corollary 6. *Given an instance (C, I, G) of the Tyreworld domain, the corresponding state space is harmless.*

Proof. The state space is not undirected because inflating a wheel can not be undone. Transforming with respect to static predicates makes sure that actions are type correct i.e., put-on-wheel(x, y) actions for example are only there for wheels x and hubs y. After transformation, all actions except inflate(x) are invertible in the sense of Definition 35 (for example, open-boot() and close-boot() invert each other). As inflate(x) actions have irrelevant delete effects and static add effects in the sense of Definition 37, we are finished with Lemma 2.

Theorem 15. *Given an instance (C, I, G) of the Tyreworld domain, the corresponding state space does not contain any local minima under h^+.*

Proof. The inflate(x) actions have irrelevant delete effects, and all other actions are invertible, so with Lemma 4 it suffices to show that all actions except inflate(x) ones are respected by the relaxation. This is quite obvious: if any action a starts an optimal plan from a reachable state, then that action adds a fact that *must* be achieved, and that *can* only be achieved by a. As an example, consider the actions opening or closing the boot, or, slightly less trivial, actions that loosen or tighten nuts: if those actions are needed to achieve the goal, they

can not be avoided by the relaxation (in fact, this also holds for the inflate(x) actions).

Theorem 16. *Given an instance (C, I, G) of the Tyreworld domain, the maximal exit distance in the corresponding state space is at most 6 under h^+.*

Proof. By the argumentation in the proof to Theorem 15, the instance fulfills the requirements for Lemma 4, so h^+ never increases on optimal solution paths. It now suffices to show that a state s' with a better evaluated neighbour can be reached on an optimal solution path from any reachable state s in at most 6 steps. We will use the fact that a state s' is an exit if there is an action a starting an optimal plan for (A, s', G), such that removing a from an optimal relaxed plan P^+ for (A, s', G) yields a relaxed plan for $(A, Result(s', \langle a \rangle), G)$.

Consider the close-boot(), put-away(x), inflate(x), tighten(x, y), jack-down(x), do-up(x, y), remove-wheel(x, y), and put-on-wheel(x, y) actions. For all such actions it holds that when they start an optimal plan for a state, then the facts that they delete are not needed anymore after applying them, neither by a real nor by a relaxed plan: a do-up(x, y) action, for example, deletes (have, x) and (not-fastened, y) for nuts x and hub y; if do-up(x, y) starts an optimal plan, then this means that the spare wheel has been put on the hub y already, so neither of both facts will be needed for achieving the goals. Our proof thus comes down to identifying in at most how many steps on an optimal path a state s' can be reached from which an action produced by one of the above operators starts an optimal plan. Let us distinguish the following cases:

1. There is a spare wheel that is not inflated, (not-inflated, s_i) $\in s$ for some s_i. The wheel must be inflated, for which one must at most open the boot and fetch the pump, yielding the distance 2.
2. The above case 1 does not hold, and there is a (flat) wheel that is still on, or a spare wheel that is on the wrong hub, i.e., (on, w_i, h_j) $\in s$, (on, w_i, h_j) $\notin G$ for some w_i and h_j. We identify an optimal path to a state where remove-wheel(w_i, h_j) can be applied (and thus starts an optimal plan). One must at most open the boot, fetch the wrench, loosen the nuts tightened to h_j (3 steps). If some other hub with a flat or wrong wheel on is already jacked up, one must at most jack down that hub (which is necessary anyway in order to loosen the nuts on the respective hub); if a hub with a good wheel is jacked up, one can jack down that hub, yielding a better state; else, the jack must at most be fetched from the boot (1 more step). Jacking up h_j and undoing the respective nuts takes at most another 2 steps, so our path is at most 6 steps long. Note that all steps on the path are inavoidable implying that the path is an optimal solution path from s.
3. The above cases do not hold, and there is a hub that is still free, (free, h_i) $\in s$ for some h_i. One must put on the respective spare. If it is still in the boot, then fetching the spare yields a state with better h^+. Else, one must at most perform the same steps as in the above case 2, yielding again an inavoidable path of maximal length 6.

4. The above cases do not hold, and some nuts are still not fastened to the correct hub, $(\text{loose}, n_i, h_i) \notin s$ for some n_i and h_i with $(\text{tight}, n_i, h_i) \in G$. If the nuts are in the boot, fetching them yields a better state. If some hub is jacked up, we can assume that either the correct nuts are fastened to that hub, or that the hub is h_i. Therefore, if some other nuts are fastened to h_i, maximal 5 unavoidable steps similar like above lead to a state where those other nuts can be undone, yielding a better state; otherwise maximal 4 unavoidable steps lead to a state where do-up(n_i, h_i) can be applied.
5. The above cases do not hold, and some hub is still jacked up, $(\text{not-on-ground}, h_i) \in s$ for some h_i. The hub can immediately be jacked down.
6. The above cases do not hold, and some nuts are still loose, $(\text{loose}, n_i, h_i) \in s$ for some n_i and h_i. The nuts must be tightened which involves at most opening the boot and fetching the wrench.
7. If none of the above cases holds, then there is either a tool or a flat wheel that can be put away, or one can close the boot.

Consider any *Tyreworld* instance and the state where all hubs except one have already been processed completely (spare wheel inflated and on, nuts tight, flat wheel in boot), the boot is closed, the wrench and the jack are in the boot, the last spare wheel is already inflated, and nothing has yet been done about the last flat wheel. We are in the second case of the list given in the above proof to Theorem 16, and we require the whole sequence of opening the boot, fetching the wrench and the jack, loosing the nuts, jacking up the hub, and undoing the nuts, in order to get to a state where the flat wheel can be removed. All the actions in the sequence must be inverted later on, and there is no other way of improving the h^+ value, so there the exit distance is exactly 6.

We have now finished looking at those domains where our hypotheses included a limited maximal exit distance, and move on to the domains where the non-existence of local minima is hypothesized. We start with a simple ADL version of the classical transportation problem.

15 Briefcaseworld

The *Briefcaseworld* is an ADL domain. Portables are transported by putting them inside a briefcase. Whenever the briefcase moves, all portables inside it are moved along by a conditional effect.

Definition 48. *The Briefcaseworld domain is* $Briefcaseworld = (\mathcal{P}, \mathcal{O}, \mathcal{I})$, *where the set of predicates considered is*

$$\begin{aligned}\mathcal{P} = \{\ & (\textit{not-eq}, 2),\\ & (\textit{location}, 1),\\ & (\textit{portable}, 1),\\ & (\textit{at}, 2),\\ & (\textit{in}, 1),\\ & (\textit{not-in}, 1),\\ & (\textit{is-at}, 1)\ \}\end{aligned}$$

$\mathcal{O}$ consists of the operators

- *move:* $\Sigma^2 \times 2^\Sigma \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$move(x,y,C) := (\ (not\text{-}eq,x,y), (location,x) \wedge (location,y) \wedge (is\text{-}at,x),$$
$$\{(\top, \{(is\text{-}at,y)\}, \{(is\text{-}at,x)\})\} \cup$$
$$\{((portable,z) \wedge (in,z), \{(at,z,y)\}, \{(at,z,x)\}) \mid z \in C\}\ )$$

- *take-out:* $\Sigma \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$take\text{-}out(x) := (\{(portable,x)\}, \{(not\text{-}in,x)\}, \{(in,x)\})$$

- *put-in:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$put\text{-}in(x,y) := (\ \{\ (portable,x), (location,y),$$
$$(not\text{-}in,x), (at,x,y), (is\text{-}at,y)\ \},$$
$$\{\ (in,x)\ \},$$
$$\{\ (not\text{-}in,x)\ \}\ )$$

and the set of instances is

$$\mathcal{I} = \{\ (\ L \cup P,$$
$$\{(not\text{-}eq,x,y) \mid x,y \in L, x \neq y\} \cup$$
$$\{(location,x) \mid x \in L\} \cup$$
$$\{(portable,x) \mid x \in P\} \cup$$
$$\{(is\text{-}at,il)\} \cup \{(at,x,pil(x)) \mid x \in P\},$$
$$(is\text{-}at,gl) \wedge \bigwedge\{(at,x,pgl(x)) \mid x \in P\}\ )$$
$$\mid (L \cup P) \subseteq \Sigma^+, |L| \geq 1,$$
$$il \in L, pil : P \mapsto L,$$
$$gl \in L, pgl : P \mapsto L\ \}$$

In the definition of the instances, *il* and *gl* are the initial and goal locations of the briefcase, while *pil* and *pgl* are the initial and goal location functions for portables. Obviously, all instances of $Briefcaseworld$ are solvable.

Proposition 5. *Given an instance (C,I,G) of the $Briefcaseworld$ domain. The corresponding state space is undirected.*

Proof. Though none of the actions is invertible in the strict syntactical sense of Definition 35, to each reachable state and applicable action a yielding the successor state s' there is an action $\overline{a}$ leading back to s from s': if a is a

- move(x,y,C), it can be undone by move(y,x,C),
- put-in(x,y), it can be undone by take-out(x),
- take-out(x), it can be undone by put-in(x,y) where y is the current location of the briefcase and x (with x being inside prior to the application of take-out(x), it is at the same location as the briefcase in s).

Theorem 17. *Given an instance (C,I,G) of the $Briefcaseworld$ domain, the corresponding state space does not contain any local minima under h^+.*

Proof. We prove that the heuristic value does not increase on optimal solution paths. The put-in(x, y) action is respected by the relaxation, because if such an action starts an optimal plan, then x is not at its goal location, so it must be put in. Also, as seen in the proof to Proposition 5, the action can be inverted. Now say we got a reachable state $s \in S$, and an optimal plan for (A, s, G) starting with a take-out(x) action. Say P^+ is an optimal relaxed plan for (A, s, G). Then P^+ is also an optimal relaxed plan for (A, s', G), where s' is the state that results from applying take-out(x) to s: x is already at its goal location, so P^+ does not do anything with it (x can be kept inside the briefcase in the relaxation, as moving away does not delete its current location there). Finally, say we got a reachable state $s \in S$ and an optimal plan for (A, s, G) starting with a move(x, y, C) action. Say P^+ is an optimal relaxed plan for (A, s, G), and s' is the state that results from applying move(x, y, C) to s. If P^+ starts with move(x, y, C), then we are finished with invertibility. In the other case, move(x, y, C) is either not contained in P^+ at all, or there is some other move action move(x, z, C) preceeding it (because moving away starts an optimal real plan, no portables must be put in at the current location). We construct a relaxed plan for (A, s', G) from P^+ by: replacing move(x, y, C), if it is contained in P^+, with move(z, y, C); and replacing all other move(x, y', C) actions in P^+ with move(y, y', C).

The shortest path to an exit can be arbitrarily long in the *Briefcaseworld* domain. Consider the instances given by $C = L \cup P = \{l_1, l_2\} \cup \{p_1, \ldots, p_n\}$ with $n \geq 1$, $il = l_1$ and $gl = l_1$, $pil(p_1) = l_2$ and $pgl(p_1) = l_1$, and $pil(p_i) = l_1, pgl(p_i) = l_2$ for $i \neq 1$. The portable p_1 must be transported from l_2 to l_1 while $n - 1$ other portables $p_2, \ldots, p_n$ must be transported from l_1 to l_2. An optimal plan puts in all of $p_2, \ldots, p_n$, moves to l_2, and puts in p_1; it then takes out all of $p_2, \ldots, p_n$, and moves back to l_1. Consider the state s where the briefcase is in l_2, and all portables are inside. $h^+(s) = 1$, as moving to l_1 does the job in the relaxation. However, the nearest state s' with $h^+(s') < h^+(s)$ is the nearest goal state, which is n steps away.

16 Fridge

In *Fridge*, for a number of fridges a broken compressor needs to be replaced with a new one. This involves loosening the screws holding the compressors—which can only be done when the respective fridge is off—exchanging the compressors, and reassembling the fridges. The representation makes use of a simple logical construct in one precondition, in order to allow for a flexible number of screws holding the compressors.

Definition 49. *The Fridge domain is* $Fridge = (\mathcal{P}, \mathcal{O}, \mathcal{I})$*, where the set of predicates considered is*

$\mathcal{P} = \{\ (\mathit{fridge}, 1),$
$(\mathit{compressor}, 1),$
$(\mathit{screw}, 1),$
$(\mathit{not\text{-}screw}, 1),$
$(\mathit{fits}, 2),$
$(\mathit{not\text{-}fits}, 2),$
$(\mathit{fridge\text{-}on}, 1),$
$(\mathit{not\text{-}fridge\text{-}on}, 1),$
$(\mathit{screwed}, 1),$
$(\mathit{not\text{-}screwed}, 1),$
$(\mathit{attached}, 1),$
$(\mathit{free}, 1)\ \}$

$\mathcal{O}$ *consists of the operators*

- *stop-fridge:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\mathit{stop\text{-}fridge}(x) := (\ \{\ (\mathit{fridge}, x), (\mathit{not\text{-}fridge\text{-}on}, x)\ \},$
  $\{\ (\mathit{fridge\text{-}on}, x)\ \},$
  $\{\ (\mathit{not\text{-}fridge\text{-}on}, x)\ \}\ )$

- *start-fridge:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\mathit{start\text{-}fridge}(x) := (\ \{\ (\mathit{fridge}, x), (\mathit{fridge\text{-}on}, x)\ \},$
  $\{\ (\mathit{not\text{-}fridge\text{-}on}, x)\ \},$
  $\{\ (\mathit{fridge\text{-}on}, x)\ \}\ )$

- *unfasten:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\mathit{unfasten}(x, y, z) := (\ \{\ (\mathit{screw}, x), (\mathit{compressor}, y), (\mathit{fridge}, z),$
  $(\mathit{screwed}, x), (\mathit{fits}, x, y), (\mathit{attached}, y, z), (\mathit{not\text{-}fridge\text{-}on}, z)\ \},$
  $\{\ (\mathit{not\text{-}screwed}, x)\ \},$
  $\{\ (\mathit{screwed}, x)\ \}\ )$

- *fasten:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\mathit{fasten}(x, y, z) := (\ \{\ (\mathit{screw}, x), (\mathit{compressor}, y), (\mathit{fridge}, z),$
  $(\mathit{not\text{-}screwed}, x), (\mathit{fits}, x, y), (\mathit{attached}, y, z), (\mathit{not\text{-}fridge\text{-}on}, z)\ \},$
  $\{\ (\mathit{screwed}, x)\ \},$
  $\{\ (\mathit{not\text{-}screwed}, x)\ \}\ )$

- *remove-compressor:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\mathit{remove\text{-}compressor}(x, y) := (\ (\mathit{compressor}, x) \wedge (\mathit{fridge}, y) \wedge$
  $(\mathit{attached}, x, y) \wedge (\mathit{not\text{-}fridge\text{-}on}, y) \wedge$
  $\forall z : (\mathit{not\text{-}screw}, z) \vee (\mathit{not\text{-}fits}, z, x) \vee (\mathit{not\text{-}screwed}, z),$
  $\{(\top, \{(\mathit{free}, x), (\mathit{free}, y)\}, \{(\mathit{attached}, x, y)\})\}\ )$

- *attach-compressor:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\mathit{attach\text{-}compressor}(x, y) := (\ \{\ (\mathit{compressor}, x), (\mathit{fridge}, y),$
  $(\mathit{free}, x), (\mathit{free}, y), (\mathit{fits}, x, y), (\mathit{not\text{-}fridge\text{-}on}, y)\ \},$
  $\{\ (\mathit{attached}, x, y)\ \},$
  $\{\ (\mathit{free}, x), (\mathit{free}, y)\ \}\ )$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{\ & (\ F \cup C \cup S, \\
& \quad \textit{where } F = \{f_1, \ldots, f_n\}, C = \{b_1, n_1, \ldots, b_n, n_n\}, \\
& \quad S = \{s_{11}, \ldots s_{1m}, \ldots, s_{n1}, \ldots, s_{nm}\}, \\
& \quad \{(\textit{fridge}, x) \mid x \in F\} \cup \\
& \quad \{(\textit{compressor}, x) \mid x \in C\} \cup \\
& \quad \{(\textit{screw}, x) \mid x \in S\} \cup \\
& \quad \{(\textit{not-screw}, x) \mid x \in F \cup C\} \cup \\
& \quad \{(\textit{fits}, b_i, f_i), (\textit{fits}, n_i, f_i) \mid 1 \leq i \leq n\} \cup \\
& \quad \{(\textit{fits}, s_{ij}, b_i), (\textit{fits}, s_{ij}, n_i) \mid 1 \leq i \leq n, 1 \leq j \leq m\} \cup \\
& \quad \{(\textit{not-fits}, s_{ij}, b_{i'}), (\textit{not-fits}, s_{ij}, n_{i'}) \mid 1 \leq i \neq i' \leq n, 1 \leq j \leq m\} \cup \\
& \quad \{(\textit{fridge-on}, x) \mid x \in F\} \cup \\
& \quad \{(\textit{screwed}, s_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq m\} \cup \\
& \quad \{(\textit{attached}, b_i, f_i) \mid 1 \leq i \leq n\} \cup \\
& \quad \{(\textit{free}, n_i) \mid 1 \leq i \leq n\}, \\
& \quad \textstyle\bigwedge \{(\textit{fridge-on}, x) \mid x \in F\} \wedge \\
& \quad \textstyle\bigwedge \{(\textit{screwed}, s_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq m\} \wedge \\
& \quad \textstyle\bigwedge \{(\textit{attached}, n_i, f_i) \mid 1 \leq i \leq n\}\ ) \\
& \mid (F \cup C \cup S) \subseteq \Sigma^+, n \geq 1, m \geq 1\ \}
\end{aligned}
$$

In the instances, F is the set of fridges, C the set of compressors, and S the set of screws. For each fridge f_i, there is the broken compressor b_i that is initially attached, and the new compressor n_i that shall be mounted. For each compressor b_i or n_i, there are the m screws $s_{i1}, \ldots, s_{im}$ that fit for the compressor. All instances of *Fridge* are solvable, by in turn stopping a fridge, unfastening the screws, replacing the broken compressor with the new one, fastening the screws, and switching the fridge back on.

Proposition 6. *Given an instance (C, I, G) of the Fridge domain, the corresponding state space is undirected.*

Proof. After transformation with respect to static predicates, obviously stop-fridge(x) and start-fridge(x) (fasten(x, y, z) and unfasten(x, y, z)) actions are invertible in the sense of Definition 35, by each other. For remove-compressor(x, y) and attach-compressor(x, y), matters are a bit more complicated; while they obviously undo each other's effects, applicability is slightly less obvious. Say a compressor c has just been removed from a fridge f in a reachable state s. Then, c fits f—otherwise it couldn't have been attached in s—-so one can attach c back onto f. Now say a compressor c has just been attached to a fridge f in a reachable state s. The question is, are the screws all unscrewed? If not, then they must have been fastened (and not unfastened afterwards) to a matching compressor that is already attached to a fridge. However, both matching compressors can be attached only to the same fridge f, and f has been free in s.

Theorem 18. *Given an instance (C, I, G) of the Fridge domain, the corresponding state space does not contain any local minima under h^+.*

Proof. We show that all actions are respected by the relaxation. Considering the argumentation given above in the proof to Proposition 6, this suffices by Lemma 4 (or rather by a slight extension of that lemma, allowing for the inverse relation between remove-compressor(x, y) and attach-compressor(x, y), which does not strictly syntactically follow the rules for at least relaxed invertibility, but does have the same consequences semantically).

It is obvious that start-fridge(x) and attach-compressor(x, y) actions are respected by the relaxation: such actions can only start an optimal plan if they add the respective goal, which can not be added by any other action. So those actions are then contained in any optimal relaxed plan P^+, and can be used to start P^+ because they are applicable. If fasten(x, y, z) starts an optimal plan, then this means that y is the new compressor for z, so that action is also part of an optimal relaxed plan (the relaxed plan could also achieve the goal (screwed, x) by fastening x to the broken compressor for z, but this involves more steps). If stop-fridge(x) starts an optimal plan, then this means that either the broken compressor is attached to x, or that one of the screws is still unfastened. In both cases a relaxed plan must also stop x in order to apply the respective actions: the first case is obvious, and the second case follows because the screws can only be fastened to a matching compressor, which in turn can only be attached to x. If unfasten(x, y, z) starts an optimal plan, then y is the broken compressor for z, so unfastening x prior to removing y from z can not be avoided in a relaxed plan. Similarly, remove-compressor(x, y) starting an optimal plan means that x is the broken compressor for y, removing which can not be avoided.

The maximal exit distance in *Fridge* is not limited because there can be an arbitrary number of screws for each compressor. Consider the instances with a single fridge and n screws. From the initial state, an optimal relaxed plan stops the fridge, unfastens all screws, removes the broken compressor, and attaches the new one, so $h^+(I) = n + 3$. The nearest better state is the one where the broken compressor has been removed already: in all states where that has not been done, one must in the relaxed plan always do the removing and attaching action, either start or stop the fridge, and either fasten or unfasten each screw, depending on the current state of the fridge or screw (if a screw is unfastened, it needs to be fastened for achieving the respective goal, and if a screw is fastened, it needs to be unfastened in order to remove the broken compressor; similarly for the fridge). So h^+ is constantly $h^+(s) = n + 3$ for all states s where the broken compressor is attached to the fridge. Getting from the initial state to a state where the broken compressor can be removed requires $n + 1$ steps, which is a lower limit to the maximal exit distance. [4]

[4] As a matter of fact, that is exactly the maximal exit distance: from a state where no compressor is attached, one can get to a better state by stopping the fridge if it's on, and attaching the new compressor; from a state where the new compressor is attached one gets to a better state by stopping the fridge if it's on, and fastening one more screw

17 Blocksworld-no-arm

In the *Blocksworld*, a number of stackable blocks need to be rearranged on a table with infinite space. In one of the two common representational variants, blocks are moved around directly, without an explicit notion of a robot arm.

Definition 50. *The Blocksworld-no-arm domain is Blocksworld-no-arm* $= (\mathcal{P}, \mathcal{O}, \mathcal{I})$*, where the set of predicates considered is*

$$\mathcal{P} = \{(\textit{not-eq}, 2), (\textit{clear}, 1), (\textit{on-table}, 1), (\textit{on}, 2)\}$$

$\mathcal{O}$ *consists of the operators*

- *move-table-to-block:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\textit{move-table-to-block}(x, y) := ($ { $(\textit{not-eq}, x, y), (\textit{clear}, x), (\textit{clear}, y), (\textit{on-table}, x)$ },
  { $(\textit{on}, x, y)$ },
  { $(\textit{clear}, y), (\textit{on-table}, x)$ })

- *move-block-to-table:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\textit{move-block-to-table}(x, y) := ($ { $(\textit{clear}, x), (\textit{on}, x, y)$ },
  { $(\textit{on-table}, x), (\textit{clear}, y)$ },
  { $(\textit{on}, x, y)$ })

- *move-block-to-block:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

  $\textit{move-block-to-block}(x, y, z) := ($ { $(\textit{not-eq}, x, z), (\textit{clear}, x), (\textit{on}, x, y), (\textit{clear}, z)$ },
  { $(\textit{on}, x, z), (\textit{clear}, y)$ },
  { $(\textit{on}, x, y), (\textit{clear}, z)$ })

and the set of instances is

$$\begin{aligned}
\mathcal{I} = \{\ (B,\ & \{(\textit{not-eq}, x, y) \mid x, y \in B, x \neq y\} \cup \\
& \{(\textit{on-table}, x) \mid x \in B, \forall\, y : (x, y) \notin I\} \cup \\
& \{(\textit{on}, x, y) \mid (x, y) \in I\} \cup \\
& \{(\textit{clear}, x) \mid x \in B, \forall\, y : (y, x) \notin I\}, \\
& \{(\textit{on}, x, y) \mid (x, y) \in G\}\ ) \\
\mid\ & B \subseteq \Sigma^+, |B| \geq 1, \\
& I \subseteq B \times B, (B, I) \textit{ is cycle-free graph with in- and out-degree 1}, \\
& G \subseteq B \times B, (B, G) \textit{ is cycle-free graph with in- and out-degree 1}\ \}
\end{aligned}$$

In the instances, the edge sets I and B represent the initial and goal states, respectively, by the semqantics that $(x, y) \in I$ $(\in G)$ if and only if x is initially (shall be finally) located on y. Note that an edge set represents a legal state iff there are no cycles in the corresponding on relation, each block has at most one block located on top of it, and each block is located on top of at most one other block. All *Blocksworld-no-arm* instances are trivially solvable by moving all blocks to the table from top to bottom, and building the goal stacks from bottom to top.

Corollary 7. *Given an instance (C, I, G) of the Blocksworld-no-arm domain, the corresponding state space is undirected.*

Proof. After transformation with respect to static predicates, all actions are invertible in the sense of Definition 35: move-table-to-block(x, y) is inverted by move-block-to-table(x, y) and vice versa, and move-block-to-block(x, y, z) is inverted by move-block-to-block(x, z, y). By applying Lemma 1, we are finished.

Theorem 19. *Given an instance (C, I, G) of the Blocksworld-no-arm domain, the corresponding state space does not contain any local minima under h^+.*

Proof. We prove that there is always at least one optimal solution path on which the heuristic value does not increase. Let $s \in S$ be a reachable state with $gd(s) > 0$. We distinguish the following two cases:

1. There is an optimal plan for (A, s, G) starting with an action a that puts some block x into its goal position on block y.
2. No optimal plan for (A, s, G) starts with such an action.

In the first case, we will see that there is an optimal relaxed plan for (A, s, G) containing a. In the second case, we will identify an optimal plan P_s for (A, s, G) starting with an action a that moves a block x onto the table, and an optimal relaxed plan P_s^+ for (A, s, G) containing an action a' that moves x. In both cases, we prove that $h^+(\mathit{Result}(s, \langle a \rangle)) \leq h^+(s)$, which finishes the argumentation.

Say there is an optimal plan for (A, s, G) starting with a, putting block x into its goal position on block y, $(\text{on}, x, y) \in G$, i.e., $a =$ move-table-to-block(x, y) or $a =$ move-block-to-block(x, z, y) for some third block z. Obviously, all relaxed plans for (A, s, G) must also at some point move x onto y. If this is done in a different way than by a, then x must first be moved to a different position, involving one more step. Moreover, moving x to a different position can achieve—apart from the (clear, z) fact that is then also achieved by a—only an $(\text{on-table}, x)$ or (on, x, u) fact. Those facts are of no other use than for moving around x. It follows that all optimal relaxed plans for (A, s, G) use a for achieving (on, x, y). With invertibility, we are finished.

Now say that there is no optimal plan for (A, s, G) that starts by putting some block into its goal position. We have a closer look at the situation. The blocks are arranged on the table in stacks—a bottom block directly on the table, a sequence of blocks transitively on that block, a clear top block. The *trivial* stack contains only a single block that is bottom and top at once. We say that a stack is a *good* stack, if it is consistent with the goal conditions, i.e., if $(\text{on}, x, y) \in G$ and x or y are in the stack, then x is located on y (implying that both blocks are in the stack). Obviously, optimal plans leave good stacks unchanged—blocks in these stacks are either not mentioned in the goal, or already in their final position. Focussing on the stacks in s that are *not* good, we observe the following:

1. Any plan for (A, s, G), relaxed or real, will need to move the top blocks x of all stacks in s that are not good: either x itself is not in its goal position, or some block y below isn't, in which case x must be moved away for clearing y.

2. Let a be an action that moves a block x to the table where x is top of a non-trivial stack that is not good. Then a starts an optimal plan for (A, s, G): the blocks x must be moved, but by prerequisite no such block x can be optimally moved into its goal position right away; moving x to the table can only be better than moving it onto some block where it does not need to be finally.

Moreover, we observe that there is at least one non-trivial non-good stack in s: otherwise, all non-good stacks would consist of a single block, implying that some block *could* be put into its goal position right away. Let x be the top block of a non-trivial non-good stack in s, located on y, and $a :=$ move-block-to-table(x, y). By the above second observation, a starts an optimal plan P_s for (A, s, G), and by the above first observation, an arbitrary optimal relaxed plan P_s^+ for (A, s, G) contains an action moving x away from y; let a' be the first such action. If $a' =$ move-block-to-table(x, y), we are finished with invertibility. Say $a' =$ move-block-to-block(x, y, z). Denote by $s' := \mathit{Result}(s, \langle a \rangle)$ the result of applying a in s. We distinguish two cases:

1. $(\text{on}, x, y) \in G$: Construct an action sequence $P_{s'}^+$ by replacing a' in P_s^+ with move-table-to-block(x, y). $P_{s'}^+$ is a relaxed plan for (A, s', G): no action before a' needs (on, x, y) as precondition (by prerequisite, a' is the first action moving x away from y); executing the relaxed version of move-block-to-block(x, y, z) in P_s^+ achieves one more fact, (on, x, z), but x is already in its goal position, so the optimal relaxed plan P_s^+ moves x only once, thus (on, x, z) is not needed by any other action.
2. $(\text{on}, x, y) \notin G$: Construct an action sequence $P_{s'}^+$ by replacing a' in P_s^+ with move-table-to-block(x, z). Now, $P_{s'}^+$ is a relaxed plan for (A, s', G) unless there is some action that needs (on, x, y) as precondition, which must be a move-block-to-block(x, y, u) (moving x to the table is useless after applying the relaxed version of a'). Such an action comes after a' in P_s^+ (remember that a' is the first action moving x away from y), so it can be replaced by move-block-to-block(x, z, u) in $P_{s'}^+$. This replacement does not achieve (clear, y), but that fact has already been true in s' anyway.

In both cases, we have a relaxed plan $P_{s'}^+$ for (A, s', G) with $|P_{s'}^+| = |P_s^+|$, which concludes our argumentation.

The maximal exit distance in *Blocksworld-no-arm* is not limited. Consider an instance with $n + 1$ blocks $\{b_1, \ldots, b_{n+1}\}$, initial state $I = \{(\text{clear}, b_1)$, (on, b_1, b_2), $\ldots$, $(\text{on}, b_{n-1}, b_n)$, $(\text{on-table}, b_n)$, (clear, b_{n+1}), $(\text{on-table}, b_{n+1})\}$ and goal state $G = \{(\text{on}, b_1, b_2)$, $\ldots$, $(\text{on}, b_n, b_{n+1})\}$. Here, the stack $b_1, \ldots, b_n$ needs to be disassembled, and rebuilt on top of b_{n+1}. An optimal relaxed plan for I has length n, by moving $b_1 \ldots b_{n-1}$ to the table, and moving b_n on top of b_{n+1}. Now, $h^+(s') \geq n$ for all reachable states s' where b_n is not on top of b_{n+1}: b_n must be moved to b_{n+1}; to make this possible, b_{n-1} must be moved away from b_n if it is currently on top of it, and otherwise it must be moved to b_n for achieving the goal; the same argument can iteratively applied upwards to b_1. Now consider

again the initial state. Reaching, from there, the nearest state s' where b_n is on top of b_{n+1} involves n steps moving $b_1 \ldots b_{n-1}$ away, and moving b_n onto b_{n+1}. Thus the maximal exit distance in the instance is at least n.

18 Grid

In the *Grid* domain, one moves on a grid map of places; some places are initially locked, and need to be opened with matching keys for passing through. Some keys must be transported to their goal places. One key can be carried at a time.

Definition 51. *The Grid domain is* $Grid = (\mathcal{P}, \mathcal{O}, \mathcal{I})$*, where the set of predicates considered is*

$$\begin{aligned}\mathcal{P} = \{\ & (\mathit{place}, 1),\\ & (\mathit{key}, 1),\\ & (\mathit{shape}, 1),\\ & (\mathit{conn}, 2),\\ & (\mathit{key\text{-}shape}, 2),\\ & (\mathit{lock\text{-}shape}, 2),\\ & (\mathit{open}, 1),\\ & (\mathit{locked}, 1),\\ & (\mathit{at}, 2),\\ & (\mathit{at\text{-}robot}, 1),\\ & (\mathit{holding}, 1),\\ & (\mathit{arm\text{-}empty}, 1)\ \}\end{aligned}$$

$\mathcal{O}$ *consists of the operators*

- *move:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\mathit{move}(x, y) := (\ & \{\ \{(\mathit{place}, x), (\mathit{place}, y),\\ & \quad (\mathit{conn}, x, y), (\mathit{at\text{-}robot}, x), (\mathit{open}, y)\},\\ & \{\ (\mathit{at\text{-}robby}, y)\},\\ & \{\ (\mathit{at\text{-}robby}, x)\}\ )\end{aligned}$$

- *pickup:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\mathit{pickup}(x, y) := (\ & \{\ (\mathit{place}, x), (\mathit{key}, y),\\ & \quad (\mathit{at\text{-}robot}, x), (\mathit{at}, y, x), (\mathit{arm\text{-}empty})\ \},\\ & \{\ (\mathit{holding}, y)\ \},\\ & \{\ (\mathit{at}, y, x), (\mathit{arm\text{-}empty})\ \}\ )\end{aligned}$$

- *pickup-and-loose:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}\mathit{pickup\text{-}and\text{-}loose}(x, y, z) := (\ & \{\ (\mathit{place}, x), (\mathit{key}, y), (\mathit{key}, z),\\ & \quad (\mathit{at\text{-}robot}, x), (\mathit{holding}, y), (\mathit{at}, z, x)\ \},\\ & \{\ (\mathit{holding}, z), (\mathit{at}, y, x)\ \},\\ & \{\ (\mathit{holding}, y), (\mathit{at}, z, x)\ \}\ )\end{aligned}$$

– *putdown:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{putdown}(x,y) := (\ &\{\ (\mathit{place},x), (\mathit{key},y),\\ &\quad (\mathit{at\text{-}robot},x), (\mathit{holding},y)\ \},\\ &\{\ (\mathit{at},y,x), (\mathit{arm\text{-}empty})\ \},\\ &\{\ (\mathit{holding},y)\ \}\ ) \end{aligned}$$

– *unlock:* $\Sigma^4 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{unlock}(x,y,z,v) := (\ &\{\ (\mathit{place},x), (\mathit{place},y), (\mathit{key},z), (\mathit{shape},v),\\ &\quad (\mathit{conn},x,y), (\mathit{at\text{-}robot},x), (\mathit{holding},z),\\ &\quad (\mathit{key\text{-}shape},z,v), (\mathit{lock\text{-}shape},y,v), (\mathit{locked},y)\ \},\\ &\{\ (\mathit{open},y)\ \},\\ &\{\ (\mathit{locked},y)\ \}\ ) \end{aligned}$$

and the set of instances is

$$\begin{aligned} \mathcal{I} = \{\ (\ & P \cup S \cup K, \textit{where } P = \{p_{11}, \ldots, p_{nm}\},\\ & \{(\mathit{place},x) \mid x \in P\} \cup\\ & \{(\mathit{shape},x) \mid x \in S\} \cup\\ & \{(\mathit{key},x) \mid x \in K\} \cup\\ & \{(\mathit{conn}, p_{ij}, p_{i'j'}) \mid i = i', |j - j'| = 1 \textit{ or } j = j', |i - i'| = 1\} \cup\\ & \{(\mathit{open},x) \mid x \in P'\} \cup\\ & \{(\mathit{locked},x) \mid x \in P \setminus P'\} \cup\\ & \{(\mathit{lock\text{-}shape}, x, ls(x)) \mid x \in P \setminus P'\} \cup\\ & \{(\mathit{key\text{-}shape}, x, ks(x)) \mid x \in K\} \cup\\ & \{(\mathit{at\text{-}robot}, ip) \cup \{(\mathit{at}, x, kip(x)) \mid x \in K\} \cup\\ & \{(\mathit{arm\text{-}empty})\},\\ & \{(\mathit{at}, x, kgp(x)) \mid x \in K'\}\ )\\ \mid\ & (P \cup S \cup K) \subseteq \Sigma^+, |P| \geq 1,\\ & P' \subseteq P, ls : P \setminus P' \mapsto S, ks : K \mapsto S,\\ & ip \in P', kip : K \mapsto P, kgp : K \mapsto P\ \} \end{aligned}$$

In the instances, P' is the set of places that are initially already open. *ls* assigns to the other, locked, places, their lock shape, and *ks* assigns to all keys their shape. $ip \in P'$ is the initial place of the robot, which *must be open*—this is an intuitively justified restriction (after all the robot is supposed to only be standing at unlocked places), and is necessary to prove the non-existence of local minima. *kip* assigns the initial-, and *kgp* the goal- key places. *Grid* instances can be unsolvable when, e.g., some important place is locked with a shape for which there is no matching key.

Proposition 7. *Given a solvable instance (C, I, G) of the Grid domain, the corresponding state space is harmless.*

Proof. The state space is not undirected as unlocking a place can not be undone. Any pickup(x, y), putdown(x, y), and pickup-and-loose(x, y, z) actions are invertible in the sense of Definition 35. move(x, y) actions are inverted by move(y, x) actions except the missing guarantee that the preconditions

(conn, y, x) and (open, x) will be true; the former precondition is true because the static (conn, x, y) relation is symmetric, and the latter precondition is true because moves can only be made starting in open positions. An $\text{unlock}(x, y, z, v)$ action does have static add effects in the sense of Definition 37; the delete effects are not irrelevant in the strict sense of that definition because (locked, y) is the precondition of other actions that unlock the same place y; however, once $\text{unlock}(x, y, z, v)$ has been applied, those actions—which have exactly the same static add effects—are not needed anymore. Thus, altogether a similar argumentation like that of Lemma 2 proves the proposition.

We remark here that all extensions to Lemma 2 that are made in the proof above are syntactical (the other unlock actions having the same static add effects), or could be detected by dealing with static facts (the (conn, y, x) relation is static), except the fact that a $\text{move}(x, y)$ action will always be applied in a state where (open, x) is true.

Theorem 20. *Given a solvable instance (C, I, G) of the Grid domain, the corresponding state space does not contain any local minima under h^+.*

Proof. Let $s \in S$ be a reachable state with $gd(s) > 0$. We identify a flat path to an exit, i.e., a path on which h^+ does not increase. Iterating the argument, we obtain a flat path to a goal state, which proves the non-existence of local minima.

Let p be the current position of the robot, $(\text{at-robot}, p) \in s$. Let P_s^+ be an optimal relaxed plan for (A, s, G) (with solvability and Proposition 7, there is at least one such P_s^+). We focus in the following on one action a in P_s^+ that makes use of a key k at place p'. If there is an unlock action in P_s^+, then we select the first such action $a = \text{unlock}(p', y, k, v)$. If there is no single unlock, then we select an arbitrary putdown action $a = \text{putdown}(p', k)$; without loss of generality, there is at least one such action because we can assume that the last action in P_s^+ is a putdown (remember that the only goals are to have some keys at their goal places). We distinguish between three cases:

1. The robot is holding k in s, $(\text{holding}, k) \in s$.
2. The arm is empty in s, $(\text{arm-empty}) \in s$.
3. The robot is holding $k' \neq k$ in s, $(\text{holding}, k') \in s, k' \neq k$.

In the first case, we identify a flat path to an exit state s'. In the second and third cases, we identify a flat path to a state s' where the robot is holding k, i.e., where the first case holds.

Say the robot is holding key k. The action a we have selected is either the first unlock in P_s^+, or a putdown, using k. In both cases, a could be executed given the robot was at place p', and in both cases, P_s^+ contains what we will call an *open path from p to p'* in the following: P_s^+ contains, preceding a, a subsequence $\langle \text{move}(p = p_1, p_2), \ldots, \text{move}(p_{n-1}, p_n = p') \rangle$ of moving actions connecting p to p', such that $(\text{open}, p_i) \in s$ for $1 \leq i \leq n$; the moving actions are there because otherwise a could not be applied at the point in P_s^+ where it appears; p_1 is open

because the robot can only be at open places in all reachable states by definition, and $p_2, \ldots, p_n$ are open because we selected a to be the *first* unlock in P_s^+. Let $s_i := \mathit{Result}(s, \langle \text{move}(p_1, p_2), \ldots, \text{move}(p_{i-1}, p_i)\rangle)$. Our flat path is the above open path, and our exit state s' is $s' := s_n$. It suffices to show that $h^+(s_i) \leq h^+(s)$ for $1 \leq i \leq n$, and that there is a state $s'', (s', s'') \in E, h^+(s'') < h^+(s)$. Consider s_2, resulting from s by executing move(p, p_2). As that action is contained in P_s^+, Lemma 3 allows us to construct a relaxed plan $P_{s_2}^+$ for (A, s_2, G) by removing move(p, p_2), and adding $\langle \text{move}(p_2, p)\rangle$ as a prefix. Therefore, $h^+(s_2) \leq h^+(s)$. Now, the next action move(p_2, p_3) on our open path is still contained in $P_{s_2}^+$, so the same argument applies to s_3, and so forth. Let $P_{s'}^+$ be the thus constructed relaxed plan for $s' = s_n$. Our action a is applicable in s', and it is contained in $P_{s'}^+$. Let $s'' := \mathit{Result}(s', a)$. We construct a relaxed plan for s'' as follows. First, we remove a from $P_{s'}^+$. If $a = \text{unlock}(p', y, k, v)$, that does the job: the single delete is (locked, y), which is only needed for actions that unlock y. Now, the optimal relaxed plan P_s^+ for (A, s, G) does of course not open y twice, and we did not add new unlock actions when constructing $P_{s'}^+$. If $a = \text{putdown}(p', k)$, the delete is $(\text{holding}, k)$. No unlock actions are contained in $P_{s'}^+$, but $(\text{holding}, k)$ might be needed by pickup-and-loose(x, k, z) actions contained in $P_{s'}^+$. These can, however, be replaced by pickup(x, z) actions when the relaxed plan starts out in s''. In both cases for the selction of a, the resulting relaxed plan for (A, s'', G) is one step shorter than that for (A, s', G), which concludes the argument.

We still need to care about the cases where the arm is empty in s, or where the robot is holding some different key $k' \neq k$. In both cases, we can move along open paths in P_s^+ to end up in a state s' where the robot holds k. Consider the case where $(\text{arm-empty}) \in s$. At the point in P_s^+ where a is applied, $(\text{holding}, k)$ has been achieved. This was done via a pickup(p', k) or a pickup-and-loose(p', k'', k) action preceding a in P_s^+. In the first case, we stop, in the second case, we apply the same argument to k'', i.e., before pickup-and-loose(p', k'', k) is applied in P_s^+ there must be some action achieving $(\text{holding}, k'')$. At some point, a key must be picked up, as initially in s no key is holded. So we end up with a list $k_1, \ldots, k_n = k$ of keys such that $\langle \text{pickup}(p_1, k_1)$, pickup-and-loose$(p_2, k_1, k_2)$, $\ldots$, pickup-and-loose$(p_n, k_{n-1}, k_n = k)\rangle$ is a subsequence of P_s^+ for certain places $p_1, \ldots, p_n$. P_s^+ contains open paths from p to all $p_i, 1 \leq i \leq n$ (remember that either our selected action a using k is the very first unlock in P_s^+, or there are no unlocks needed at all). We move along the path from p to p_1, execute pickup(p_1, k_1), and move back from p_1 to p. At all points on this path we can use, like above, Lemma 3 to construct a relaxed plan for the new state that has the same number of steps like P_s^+: putdown(p_1, k_1) inverts pickup(p_1, k_1), and the actions moving back from p_1 to p are all contained in the relaxed plan constructed when moving to p_1. Moving back to p constructs a relaxed plan that contains exactly the same move actions as P_s^+. We can thus in the same fashion move from p to p_i, $2 \leq i \leq n-1$, execute pickup-and-loose(p_i, k_{i-1}, k_i) (inverted by pickup-and-loose(p_i, k_i, k_{i-1})), and move back to p. We finally move from p to p_n and execute pickup-and-loose$(p_n, k_{n-1}, k_n = k)$ which brings us to the desired state s' where the robot is holding the key k.

Finally, consider the case where $(\text{holding}, k') \in s, k' \neq k$. Like in the above case, at the point in P_s^+ where a is applied, $(\text{holding}, k)$ has been achieved. So, like above, there is either a pickup(p', k) or a pickup-and-loose(p', k'', k) action preceding a in P_s^+, the same holding for k'' unless $k'' = k'$. Iterating the argument gives us a key list $k_1, \ldots, k_n = k$ such that either $\langle$pickup(p_1, k_1), pickup-and-loose(p_2, k_1, k_2), $\ldots$, pickup-and-loose$(p_n, k_{n-1}, k_n = k)\rangle$ or $\langle$pickup-and-loose$(p_1, k' = k_1, k_2)$, $\ldots$, pickup-and-loose$(p_n, k_{n-1}, k_n = k)\rangle$ is a subsequence of P_s^+. The first case can be reduced to the second case by replacing pickup(p_1, k_1) with pickup-and-loose(p_1, k', k_1) in P_s^+, and adding k' as the front member of the key list. Just like we did above, we can then move along the open paths from p to p_i, execute the adequate pickup-and-loose(p_i, k_{i-1}, k_i) action, and move back to p, finally ending up in the state s' where pickup-and-loose$(p_n, k_{n-1}, k_n = k)$ has been executed, and the robot is thus holding the correct key, which case we have treated earlier.

We remark two things about the above argumentation: first, Theorem 20 also holds when there is no pickup-and-loose operator (in fact, the proof is a lot easier in this case); second, the identified flat paths can be arbitrarily long and the maximal exit distance in *Grid* is thus *not* limited by a constant. This can also be seen by considering the simple example where the robot and one key k have initially n moving steps distance, and k shall be put to the robot's starting place. The nearest state with improved h^+ value is the state where k has been picked up, which is $n + 1$ steps away.

19 Hanoi

The *Hanoi* domain is an encoding of the classical Towers of Hanoi problem.

Definition 52. *The Hanoi domain is* $Hanoi = (\mathcal{P}, \mathcal{O}, \mathcal{I})$*, where the set of predicates considered is*

$$\mathcal{P} = \{(smaller, 2), (clear, 1), (on, 2)\}$$

$\mathcal{O}$ *consists of the single operator*

- *move:* $\Sigma^3 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} move(x, y, z) := (\ &\{\ (on, x, y), (clear, x), \\ &\quad (clear, z), (smaller, x, z)\ \}, \\ &\{\ (on, x, z), (clear, y)\ \}, \\ &\{\ (on, x, y), (clear, z)\ \}\ ) \end{aligned}$$

and the set of instances is

$$\begin{aligned} \mathcal{I} = \{\ (\ &\{p_1, p_2, p_3, d_1, \ldots, d_n\}, \\ &\{(smaller, d_i, p_1), (smaller, d_i, p_2), (smaller, d_i, p_3) \mid 1 \leq i \leq n\} \cup \\ &\{(smaller, d_i, d_j) \mid 1 \leq i < j \leq n\} \cup \\ &\{(on, d_n, p_1)\} \cup \{(on, d_i, d_{i+1}) \mid 1 \leq i \leq n-1\} \cup \\ &\{(clear, d_1), (clear, p_2), (clear, p_3)\}, \\ &\{(on, d_n, p_3)\} \cup \{(on, d_i, d_{i+1}) \mid 1 \leq i \leq n-1\}\ ) \\ \mid\ &\{p_1, p_2, p_3, d_1, \ldots, d_n\} \subseteq \Sigma^+, n \geq 1\ \} \end{aligned}$$

In the instance encodings, p_1, p_2, and p_3 are the three "pegs", while $d_1, \ldots, d_n$ are the "discs" in increasing order of size. A move(x, y, z) action moves disc x from disc (or peg) y to disc (or peg) z. All instances of *Hanoi* are solvable, by the well known solution algorithm.

Corollary 8. *Given an instance (C, I, G) of the Hanoi domain, the corresponding state space is undirected.*

Proof. After transformation with respect to static predicates, all actions are invertible in the sense of Definition 35: move(x, y, z) is inverted by move(x, z, y). Finished with Lemma 1.

An exact measure of $h^+(s)$ for any reachable state s is the following.

Lemma 5. *Given an instance (C, I, G) of the Hanoi domain, and a reachable state $s \in S$. Denote, for ease of reading, p_3 with d_{n+1}, and let $sg(s)$ be defined as follows:*

$$sg(s) := min\{m \mid 1 \leq m \leq n+1, (on, d_m, d_{m+1}), \ldots, (on, d_n, d_{n+1}) \in s\}$$

Then $h^+(s) = sg(s) - 1$ holds.

Proof. $sg(s)$ is the number of the smallest disc that is already in its goal position in s, with all discs below it also being in their goal position ($sg(s) = n+1$ if no disc, or only p_3, is already in its goal position). If $sg(s) = 1$, the goal is achieved and there is nothing to show. Else, we show per induction over i, $i \leq n$, that i relaxed steps can get us to a state s' where (on, d_1, d_2), $\ldots$, $(\text{on}, d_i, d_{i+1}) \in s'$. With that, the lemma follows: after $sg(s)-1$ steps, (on, d_1, d_2), $\ldots$, $(\text{on}, d_{sg(s)-1}, d_{sg(s)}) \in s'$, which gives us $G \subseteq s'$ as $(\text{on}, d_{sg(s)}, d_{sg(s)+1})$, $\ldots$, $(\text{on}, d_n, d_{n+1} = p_3)$ are true in s already (also, at least $sg(s) - 1$ relaxed steps are necessary as that many $(\text{on}, d_i, d_{i+1})$ facts from G are missing in s, and each action achieves at most one such fact).

We prove the stronger result that, after i relaxed steps, we can get to a state s' where (on, d_1, d_2), $\ldots$, $(\text{on}, d_i, d_{i+1}) \in s'$, (clear, d_1), $\ldots$, $(\text{clear}, d_{i+1}) \in s'$, and $(\text{clear}, d) \in s'$ for all discs or pegs d such that $(\text{on}, d_j, d) \in s$ for some $j \leq i$.

Base case $i = 1$: $(\text{clear}, d_1) \in s$ simply because s is reachable and no disc can be placed onto the smallest disc d_1. Either d_1 is on d_2 in s, or d_1 is on some other disc or peg d. In the first case, we execute the relaxed step move$(d_1, d_2, z)^+$ for some z that is clear in s; in the second case, d_2 is clear (no disc other than d_1 can be on d_2), so we can execute move$(d_1, d, d_2)^+$. In both cases, (clear, d_1), (clear, d_2) and (on, d_1, d_2) are contained in the resulting state s', and also (clear, d) holds for the disc or peg d on which d_1 is in s.

Inductive case $i \longrightarrow i+1, i+1 \leq n$: let s' be the state reached after i relaxed steps, with the required properties. Either d_{i+1} is on d_{i+2} in s (remember that we denote $d_{n+1} := p_3$), or d_{i+1} is on some other disc or peg d. In both cases there is an action applicable to s' which yields, in its relaxed version, a state that satisfies the requirements. Consider the second case. We can apply move$(d_{i+1}, d, d_{i+2})^+$: $(\text{clear}, d_{i+1}) \in s'$ per induction hypothesis; if d_{i+2} was not clear, this would be

due to a smaller disc d_j, $j < i+1$, being on d_{i+2} in s, so again $(\text{clear}, d_{i+2}) \in s'$ per induction hypothesis. The resulting state has d clear, and d_{i+1} on d_{i+2}, so it fulfills the requirements. Consider now the first case, where d_{i+1} is on d_{i+2} in s. Applying $\text{move}(d_{i+1}, d_{i+2}, z)^+$ for some z would obviously yield a state satisfying the requirements (the only fact that must be achieved is (clear, d_{i+2})). Concerning the applicability of $\text{move}(d_{i+1}, d_{i+2}, z)^+$ in s', $(\text{clear}, d_{i+1}) \in s'$ per induction hypothesis. It remains to find a disc or peg z with $(\text{smaller}, d_{i+1}, z) \in s'$ and $(\text{clear}, z) \in s'$. Let p be a peg such that d_{i+1} is not transitively on p in s, and z be the topmost disc transitively on p in s such that $(\text{smaller}, d_{i+1}, z) \in s'$, or $z := p$ if there is no such disc. z is clear: if not, $(\text{on}, d_j, z) \in s$ would hold for some disc $j \leq i$ smaller than d_{i+1}, so $(\text{clear}, z) \in s'$ per induction hypothesis.

Theorem 21. *Given an instance (C, I, G) of the Hanoi domain, the corresponding state space does not contain any local minima under h^+.*

Proof. Follows from Lemma 5 and the fact that an optimal plan does not move around the discs that are already in their goal position. In a reachable non-goal state s, with $sg(s) > 1$ being the number of the smallest disc that is already in its goal position in s (all discs below it also being in their goal position), the optimal plan from s moves around smaller discs until the next smaller disc is in its goal position. Lemma 5 proves that, before this is achieved, h^+ remains $sg(s) - 1$ constantly, and that, when this is achieved, h^+ goes down to $sg(s) - 2$. Iterating the argument, h^+ does not increase on the optimal solution path from s.

We remark that the length of the flat paths is not limited by a constant; quite differently, their length is exponential in the number of discs in the instance: from the initial state of an instance with n discs, the nearest state where the largest disc d_n is in its goal position is 2^{n-1} steps away.

Before we turn our attention to the *Assembly* domain—our last domain where the non-existence of local minima is hypothesized—let us now quickly prove the hypotheses that we have about the *Blocksworld-arm* and *Schedule* domains, which concern only the dead end class of the respective instances.

20 Blocksworld-arm

In the other common representational variant of the *Blocksworld*, blocks are moved around using a robot arm that can hold one block at a time.

Definition 53. *The Blocksworld-arm domain is Blocksworld-arm $= (\mathcal{P}, \mathcal{O}, \mathcal{I})$, where the set of predicates considered is*

$$\mathcal{P} = \{(\textit{clear}, 1), (\textit{holding}, 1), (\textit{arm-empty}, 0), (\textit{on-table}, 1), (\textit{on}, 2)\}$$

$\mathcal{O}$ consists of the operators

– *pickup:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{pickup}(x) := (\ &\{\ (\mathit{clear}, x), (\mathit{on\text{-}table}, x), (\mathit{arm\text{-}empty})\ \}, \\ &\{\ (\mathit{holding}, x)\ \}, \\ &\{\ (\mathit{clear}, x), (\mathit{on\text{-}table}, x), (\mathit{arm\text{-}empty})\ \}\ ) \end{aligned}$$

– *putdown:* $\Sigma^1 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{putdown}(x) := (\ &\{\ (\mathit{holding}, x)\ \}, \\ &\{\ (\mathit{clear}, x), (\mathit{on\text{-}table}, x), (\mathit{arm\text{-}empty})\ \}, \\ &\{\ (\mathit{holding}, x)\ \}\ ) \end{aligned}$$

– *unstack:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{unstack}(x, y) := (\ &\{\ (\mathit{clear}, x), (\mathit{on}, x, y), (\mathit{arm\text{-}empty})\ \}, \\ &\{\ (\mathit{holding}, x), (\mathit{clear}, y)\ \}, \\ &\{\ (\mathit{clear}, x), (\mathit{on}, x, y), (\mathit{arm\text{-}empty})\ \}\ ) \end{aligned}$$

– *stack:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} \mathit{stack}(x, y) := (\ &\{\ (\mathit{holding}, x), (\mathit{clear}, y)\ \}, \\ &\{\ (\mathit{clear}, x), (\mathit{on}, x, y), (\mathit{arm\text{-}empty})\ \}, \\ &\{\ (\mathit{holding}, x), (\mathit{clear}, y)\ \}\ ) \end{aligned}$$

and the set of instances is

$$\begin{aligned} \mathcal{I} = \{\ (&B, \\ &\{(\mathit{arm\text{-}empty})\} \cup \\ &\{(\mathit{on\text{-}table}, x) \mid x \in B, \forall\ y : (x, y) \notin I\} \cup \\ &\{(\mathit{on}, x, y) \mid (x, y) \in I\} \cup \\ &\{(\mathit{clear}, x) \mid x \in B, \forall\ y : (y, x) \notin I\}, \\ &\{(\mathit{on}, x, y) \mid (x, y) \in G\}\ ) \\ \mid\ &B \subseteq \Sigma^+, |B| \geq 1, \\ &I \subseteq B \times B, (B, I)\ \textit{is cycle-free graph with in- and out-degree 1}, \\ &G \subseteq B \times B, (B, G)\ \textit{is cycle-free graph with in- and out-degree 1}\ \} \end{aligned}$$

The instances are exactly the same as described in Section 17, except that the arm is empty initially, which must be stated explicitly. All *Blocksworld-arm* instances are trivially solvable by moving all blocks to the table from top to bottom, and building the goal stacks from bottom to top.

Corollary 9. *Given an instance (C, I, G) of the Blocksworld-arm domain, the corresponding state space is undirected.*

Proof. All actions are invertible in the sense of Definition 35: pickup(x) and putdown(x) invert each other, the same for stack(x, y) and unstack(x, y). Finished with Lemma 1.

21 Schedule

Schedule is an ADL domain using simple conditional effects. Some objects (parts) shall be processed on a number of machines. Conditional effects are used to perform the updates on an object's state after it has undergone a working step.

Definition 54. *The Schedule domain is* $Schedule = (\mathcal{P}, \mathcal{O}, \mathcal{I})$*, where the set of predicates considered is*

$$\begin{aligned}\mathcal{P} = \{\ & (equal, 2),\\ & (not\text{-}equal, 2),\\ & (part, 1),\\ & (ashape, 1),\\ & (atemperature, 1),\\ & (surface, 1),\\ & (colour, 1),\\ & (width, 1),\\ & (anorient, 1),\\ & (machine, 1),\\ & (shape, 2),\\ & (temperature, 2),\\ & (surface\text{-}condition, 2),\\ & (painted, 2),\\ & (has\text{-}hole, 3),\\ & (not\text{-}has\text{-}hole, 3),\\ & (busy, 1),\\ & (not\text{-}busy, 1),\\ & (scheduled, 1),\\ & (not\text{-}scheduled, 1),\\ & (objscheduled, 1),\\ & (not\text{-}objscheduled, 1),\\ & (has\text{-}paint, 2),\\ & (has\text{-}bit, 2),\\ & (can\text{-}orient, 2)\ \}\end{aligned}$$

$\mathcal{O}$ *consists of the operators*

- *do-roll:* $\Sigma^1 \times 2^\Sigma \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$\begin{aligned}do\text{-}roll(x, C) := (\ & (part, x) \wedge\\ & (not\text{-}busy, roller) \wedge (not\text{-}scheduled, x),\\ & \{(\ \top,\\ & \quad \{\ (busy, roller), (scheduled, x),\\ & \quad\ (temperature, x, hot), (shape, x, cylindrical)\},\\ & \quad \{(not\text{-}busy, roller), (not\text{-}scheduled, x)\})\} \cup\\ & \{(\ (not\text{-}objscheduled),\end{aligned}$$

$\{(objscheduled)\},$
$\{(not\text{-}objscheduled)\})\} \cup$
$\{(\ (surface, y) \wedge (surface\text{-}condition, x, y),$
$\emptyset,$
$\{(surface\text{-}condition, x, y)\}) \mid y \in C\} \cup$
$\{(\ (colour, y) \wedge (painted, x, y),$
$\emptyset,$
$\{(painted, x, y)\}) \mid y \in C\} \cup$
$\{(\ (width, y) \wedge (anorient, z) \wedge (has\text{-}hole, x, y, z),$
$\{(not\text{-}has\text{-}hole, x, y, z)\},$
$\{(has\text{-}hole, x, y, z)\}) \mid y, z \in C\} \cup$
$\{(\ (ashape, y) \wedge (shape, x, y) \wedge (not\text{-}equal, y, cylindrical),$
$\emptyset,$
$\{(shape, x, y)\}) \mid y \in C\} \cup$
$\{(\ (atemperature, y) \wedge (temperature, x, y) \wedge (not\text{-}equal, y, hot),$
$\emptyset,$
$\{(temperature, x, y)\}) \mid y \in C\})$

– *do-lathe:* $\Sigma^1 \times 2^\Sigma \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$do\text{-}lathe(x, C) := (\ (part, x) \wedge$
$(not\text{-}busy, lathe) \wedge (not\text{-}scheduled, x),$
$\{(\ \top,$
$\{\ (busy, lathe), (scheduled, x),$
$(surface\text{-}condition, x, rough), (shape, x, cylindrical)\},$
$\{(not\text{-}busy, lathe), (not\text{-}scheduled, x)\})\} \cup$
$\{(\ (not\text{-}objscheduled),$
$\{(objscheduled)\},$
$\{(not\text{-}objscheduled)\})\} \cup$
$\{(\ (colour, y) \wedge (painted, x, y),$
$\emptyset,$
$\{(painted, x, y)\}) \mid y \in C\} \cup$
$\{(\ (ashape, y) \wedge (shape, x, y) \wedge (not\text{-}equal, y, cylindrical),$
$\emptyset,$
$\{(shape, x, y)\}) \mid y \in C\} \cup$
$\{(\ (surface, y) \wedge (surface\text{-}condition, x, y) \wedge (not\text{-}equal, y, rough),$
$\emptyset,$
$\{(surface\text{-}condition, x, y)\}) \mid y \in C\})$

– *do-polish:* $\Sigma^1 \times 2^\Sigma \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$do\text{-}polish(x, C) := (\ (part, x) \wedge$
$(not\text{-}busy, polisher) \wedge (not\text{-}scheduled, x) \wedge (temperature, x, cold),$

$$
\begin{array}{l}
\{(\ \top, \\
\quad \{(busy, polisher), (scheduled, x), (surface\text{-}condition, x, polished)\}, \\
\quad \{(not\text{-}busy, polisher), (not\text{-}scheduled, x)\})\} \cup \\
\{(\ (not\text{-}objscheduled), \\
\quad \{(objscheduled)\}, \\
\quad \{(not\text{-}objscheduled)\})\} \cup \\
\{(\ (surface, y) \wedge (surface\text{-}condition, x, y) \wedge (not\text{-}equal, y, polished), \\
\quad \emptyset, \\
\quad \{(surface\text{-}condition, x, y)\}) \mid y \in C\})
\end{array}
$$

– *do-grind:* $\Sigma^1 \times 2^\Sigma \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$
\begin{array}{l}
do\text{-}grind(x, C) := (\ (part, x) \wedge \\
\quad (not\text{-}busy, grinder) \wedge (not\text{-}scheduled, x), \\
\quad \{(\ \top, \\
\quad\quad \{(busy, grinder), (scheduled, x), (surface\text{-}condition, x, smooth)\}, \\
\quad\quad \{(not\text{-}busy, grinder), (not\text{-}scheduled, x)\})\} \cup \\
\quad \{(\ (not\text{-}objscheduled), \\
\quad\quad \{(objscheduled)\}, \\
\quad\quad \{(not\text{-}objscheduled)\})\} \cup \\
\quad \{(\ (colour, y) \wedge (painted, x, y), \\
\quad\quad \emptyset, \\
\quad\quad \{(painted, x, y)\}) \mid y \in C\} \cup \\
\quad \{(\ (surface, y) \wedge (surface\text{-}condition, x, y) \wedge (not\text{-}equal, y, smooth), \\
\quad\quad \emptyset, \\
\quad\quad \{(surface\text{-}condition, x, y)\}) \mid y \in C\})
\end{array}
$$

– *do-punch:* $\Sigma^3 \times 2^\Sigma \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$
\begin{array}{l}
do\text{-}punch(x, y, z, C) := (\ (part, x) \wedge (width, y) \wedge (anorient, z) \wedge \\
\quad\quad (not\text{-}busy, punch) \wedge (not\text{-}scheduled, x) \wedge \\
\quad\quad (not\text{-}has\text{-}hole, x, y, z) \wedge \\
\quad\quad (has\text{-}bit, punch, y) \wedge (can\text{-}orient, punch, z) \wedge \\
\quad\quad (temperature, x, cold), \\
\quad\quad \{(\ \top, \\
\quad\quad\quad \{\ (busy, punch), (scheduled, x), \\
\quad\quad\quad\quad (has\text{-}hole, x, y, z), (surface\text{-}condition, x, rough)\}, \\
\quad\quad\quad \{(not\text{-}busy, punch), (not\text{-}scheduled, x), \\
\quad\quad\quad\quad (not\text{-}has\text{-}hole, x, y, z)\})\} \cup \\
\quad\quad \{(\ (not\text{-}objscheduled), \\
\quad\quad\quad \{(objscheduled)\}, \\
\quad\quad\quad \{(not\text{-}objscheduled)\})\} \cup
\end{array}
$$

$$\{(\ (surface, y) \wedge (surface\text{-}condition, x, y) \wedge (not\text{-}equal, y, rough),$$
$$\emptyset,$$
$$\{(surface\text{-}condition, x, y)\}) \mid y \in C\}\ )$$

– *do-drill-press:* $\Sigma^3 \times 2^\Sigma \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$do\text{-}drill\text{-}press(x, y, z, C) := (\ (part, x) \wedge (width, y) \wedge (anorient, z) \wedge$$
$$(not\text{-}busy, drill\text{-}press) \wedge (not\text{-}scheduled, x) \wedge$$
$$(not\text{-}has\text{-}hole, x, y, z) \wedge$$
$$(has\text{-}bit, drill\text{-}press, y) \wedge (can\text{-}orient, drill\text{-}press, z) \wedge$$
$$(temperature, x, cold),$$
$$\{(\ \top,$$
$$\{(busy, drill\text{-}press), (scheduled, x), (has\text{-}hole, x, y, z)\},$$
$$\{\ (not\text{-}busy, drill\text{-}press), (not\text{-}scheduled, x),$$
$$(not\text{-}has\text{-}hole, x, y, z)\})\} \cup$$
$$\{(\ (not\text{-}objscheduled),$$
$$\{(objscheduled)\},$$
$$\{(not\text{-}objscheduled)\})\})$$

– *do-spray-paint:* $\Sigma^2 \times 2^\Sigma \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$do\text{-}spray\text{-}paint(x, y, C) := (\ (part, x) \wedge (colour, y) \wedge$$
$$(not\text{-}busy, spray\text{-}painter) \wedge (not\text{-}scheduled, x) \wedge$$
$$(has\text{-}paint, spray\text{-}painter, y) \wedge (temperature, x, cold),$$
$$\{(\ \top,$$
$$\{(busy, spray\text{-}painter), (scheduled, x), (painted, x, y)\},$$
$$\{(not\text{-}busy, spray\text{-}painter), (not\text{-}scheduled, x)\})\} \cup$$
$$\{(\ (not\text{-}objscheduled),$$
$$\{(objscheduled)\},$$
$$\{(not\text{-}objscheduled)\})\} \cup$$
$$\{(\ (surface, z) \wedge (surface\text{-}condition, x, z),$$
$$\emptyset,$$
$$\{(surface\text{-}condition, x, z)\}) \mid z \in C\} \cup$$
$$\{(\ (colour, z) \wedge (painted, x, z) \wedge (not\text{-}equal, z, y),$$
$$\emptyset,$$
$$\{(painted, x, z)\}) \mid z \in C\}\ )$$

– *do-immersion-paint:* $\Sigma^2 \times 2^\Sigma \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$do\text{-}immersion\text{-}paint(x, y, C) := (\ (part, x) \wedge (colour, y) \wedge$$
$$(not\text{-}busy, immersion\text{-}painter) \wedge (not\text{-}scheduled, x) \wedge$$
$$(has\text{-}paint, immersion\text{-}painter, y),$$
$$\{(\ \top,$$
$$\{\ (busy, immersion\text{-}painter), (scheduled, x),$$
$$(painted, x, y)\},$$
$$\{\ (not\text{-}busy, immersion\text{-}painter),$$
$$(not\text{-}scheduled, x)\})\} \cup$$
$$\{(\ (not\text{-}objscheduled),$$
$$\{(objscheduled)\},$$
$$\{(not\text{-}objscheduled)\})\} \cup$$
$$\{(\ (colour, z) \wedge (painted, x, z) \wedge (not\text{-}equal, z, y),$$
$$\emptyset,$$
$$\{(painted, x, z)\}) \mid z \in C\}\ )$$

– *do-time-step:* $2^{\Sigma} \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$do\text{-}time\text{-}step(C) := (\ (objscheduled),$$
$$\{(\ (part, x) \wedge (scheduled, x),$$
$$\emptyset,$$
$$\{(scheduled, x)\}) \mid x \in C\} \cup$$
$$\{(\ (machine, x) \wedge (busy, x),$$
$$\emptyset,$$
$$\{(busy, x)\}\ ) \mid x \in C\})$$

and the set of instances is

$$\mathcal{I} = \{\ (\ \{\ cylindrical, cold, hot, polished, rough, smooth, roller, lathe, polisher,$$
$$grinder, punch, drill\text{-}press, spray\text{-}painter, immersion\text{-}painter\} \cup$$
$$P \cup S \cup L \cup W \cup O = C,$$
$$\{(equal, x, x), (not\text{-}equal, x, y) \mid x, y \in C, x \neq y\} \cup$$
$$\{(part, x) \mid x \in P\} \cup$$
$$\{(ashape, cylindrical)\} \cup \{(ashape, x) \mid x \in S\} \cup$$
$$\{(atemperature, cold), (atemperature, hot)\} \cup$$
$$\{(surface, polished), (surface, rough), (surface, smooth)\} \cup$$
$$\{(colour, x) \mid x \in L\} \cup$$
$$\{(width, x) \mid x \in W\} \cup$$
$$\{(anorient, x) \mid x \in O\} \cup$$
$$\{\ (machine, polisher), (machine, roller), (machine, lathe),$$
$$(machine, grinder), (machine, punch), (machine, drill\text{-}press),$$
$$((machine, spray\text{-}painter), (machine, immersion\text{-}painter)\} \cup$$
$$\{(has\text{-}paint, spray\text{-}painter, x), (has\text{-}paint, immersion\text{-}painter, x) \mid x \in L\} \cup$$
$$\{(has\text{-}bit, punch, x), (has\text{-}bit, drill\text{-}press, x) \mid x \in W\} \cup$$

$$
\begin{array}{l}
\{(\mathit{can\text{-}orient}, \mathit{punch}, x), (\mathit{can\text{-}orient}, \mathit{drill\text{-}press}, x) \mid x \in O\} \cup \\
\{(\mathit{temperature}, x, \mathit{cold}) \mid x \in P\} \cup \\
\{(\mathit{shape}, x, is(x)) \mid x \in P\} \cup \\
\{(\mathit{surface\text{-}condition}, x, ic(x)) \mid x \in P\} \cup \\
\{(\mathit{painted}, x, ip(x)) \mid x \in P^{ip}\} \cup \\
\{(\mathit{has\text{-}hole}, x, y, z) \mid (y, z) \in ih(x)\}, \\
\bigwedge\{(\mathit{shape}, x, \mathit{cylindrical}) \mid x \in P^{gs}\} \wedge \\
\bigwedge\{(\mathit{surface\text{-}condition}, x, gc(x)) \mid x \in P^{gc}\} \wedge \\
\bigwedge\{(\mathit{painted}, x, gp(x)) \mid x \in P^{gp}\} \wedge \\
\bigwedge\{(\mathit{has\text{-}hole}, x, y, z) \mid (y, z) \in gh(x)\}\ ) \\
\mid\ C \subseteq \Sigma, |P| \geq 1, |L| \geq 1, |W| \geq 1, |O| \geq 1, \\
is : P \mapsto \{\mathit{cylindrical}\} \cup S, \\
ic : P \mapsto \{\mathit{polished}, \mathit{rough}, \mathit{smooth}\}, \\
P^{ip} \subseteq P, ip : P^{ip} \mapsto L, \\
ih : P \mapsto 2^{(W \times O)}, \\
P^{gs} \subseteq P, \\
P^{gc} \subseteq P, gc : P^{gc} \mapsto \{\mathit{polished}, \mathit{rough}, \mathit{smooth}\}, \\
P^{gp} \subseteq P, gp : P^{gp} \mapsto L, \\
gh : P \mapsto 2^{(W \times O)}\}
\end{array}
$$

Initially, all parts are cold, and have a shape and a surface condition specified. Some of the parts are also painted initially, and a part can have none or several holes. In the goal condition, some of the parts can be required to have cylindrical shape (the only shape that can be produced by the machines), some need an arbitrary surface condition, some must be painted, and each part can be required to have an arbitrary number of holes (note here that the parts can not be required to have *no* holes).

Theorem 22. *Given an instance (C, I, G) of the Schedule domain, the corresponding planning task is solvable, and the corresponding state space is recognized under h^+.*

Proof. We must show solvability, and that $h^+(s) = \infty$ for all $s \in S$ with $gd(s) = \infty$. For the initial state I of the task, a solution plan can be constructed as follows. Look at each part p in turn. If $(\text{shape}, p, \text{cylindrical}) \in G$ or $(\text{surface-condition}, p, \text{rough}) \in G$, and the respective fact is not true in I, then apply do-lathe(p, C), and afterwards do-time-step(C) (note that applying do-lathe(p, C) to achieve $(\text{surface-condition}, p, \text{rough})$ deletes all other shapes than $(\text{shape}, p, \text{cylindrical})$; this does not matter as being cylindrical is the only shape that is allowed to be required for the goal). If $(\text{surface-condition}, p, \text{polished}) \in G$ or $(\text{surface-condition}, p, \text{smooth}) \in G$, and the respective fact is not currently true, then apply do-polish(p, C) or do-grind(p, C), respectively, and afterwards do-time-step(C) (note here that p has temperature cold initially, so do-polish(p, C) can be applied). Next step, all still required holes $(\text{has-hole}, p, w, o) \in \mathit{goal}$ can be achieved by iteratively applying do-drill-press(p, w, o, C) and do-time-step(C) (p is still cold, and the drill-press can produce holes in arbitrary widths and orientations). Finally, if

(painted, p, c) $\in$ *goal* for some c, and (painted, p, c) is not currently true, we can apply do-immersion-paint(p, c, C), and are finished (the immersion painter has all required colours). After another application of do-time-step(C), the same process can be iterated for the next part until all parts are processed.

The same algorithm to construct a solution plan can be applied for every reachable state $s \in S$ of the task, unless there is some part p with (surface-condition, p, polished) $\in G$ or (has-hole, p, w, o) $\in G$ for some w and o, such that (temperature, p, cold) $\notin s$: all other preconditions of the required actions are obviously fulfilled at the point where they are needed (except the temperature, required non-static facts are only those concerning busy machines, scheduled objects, or p not having a certain hole; the first two are achieved by doing time steps, if the third one is not true, then the respective do-drill-press action does not need to be applied at all). It can be that a part p has (temperature, p, cold) $\notin s$ when, on the path from the initial state to s, do-roll(p, C) has been applied at some point; as there is no way of re-achieving (temperature, p, cold), s is then a dead end, $gd(s) = \infty$, if p needs to be polished or get holes made to achieve the goal, as described above. But then, $(A, s, G)^+$ is also unsolvable, $h^+(s) = \infty$: all actions that can achieve (surface-condition, p, polished) or (has-hole, p, w, o) for some w and o need (temperature, p, cold) as a precondition, which is not contained in s, and is not added by any action.

22 Assembly

We finally consider the *Assembly* domain. To say that from begin with, the situation in *Assembly* is way more complex than what we have seen in the previous domains, and as it turns out, *Assembly* is the only domain where Hoffmann's hypotheses are false.

In *Assembly*, a complex object needs to be assembled by putting together its parts, where each part can itself need to be assembled beforehand, recursively. The assembling process must obey ordering constraints on the parts of one object, and there can be ("transient") parts that must be assembled somewhere and removed afterwards, again obeying ordering constraints.

Definition 55. *The Assembly domain is Assembly* $= (\mathcal{P}, \mathcal{O}, \mathcal{I})$*, where the set of predicates considered is*

$\mathcal{P} = \{$ (*equal*, 2),
(*assembly*, 1),
(*not-assembly*, 1),
(*resource*, 1),
(*not-resource*, 1),
(*requires*, 2),
(*not-requires*, 2),
(*part-of*, 2),

$(not\text{-}part\text{-}of, 2)$,
$(transient\text{-}part, 2)$,
$(not\text{-}transient\text{-}part, 2)$,
$(assemble\text{-}order, 3)$,
$(not\text{-}assemble\text{-}order, 3)$,
$(remove\text{-}order, 3)$,
$(not\text{-}remove\text{-}order, 3)$,
$(commited, 2)$,
$(available, 1)$,
$(incorporated, 2)$,
$(not\text{-}incorporated, 2)$,
$(complete, 1)$ }

$\mathcal{O}$ *consists of the operators*

– *commit:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$commit(x, y) := (\ \{(resource, x), (assembly, y), (available, x)\},\ \{(committed, x, y)\},\ \{(available, x)\})$$

– *release:* $\Sigma^2 \mapsto \mathcal{A}^{\mathcal{P}}$ *defined by*

$$release(x, y) := (\ \{(resource, x), (assembly, y), (commited, x, y)\},\ \{(available, x)\},\ \{(committed, x, y)\})$$

– *assemble:* $\Sigma^2 \times 2^{\Sigma} \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$\begin{aligned} assemble(x, y, C) := (\ & (assembly, x) \wedge (assembly, y) \wedge \\ & (\forall z : (not\text{-}resource, z) \vee (not\text{-}requires, y, z) \vee \\ & \quad (committed, z, y)) \wedge \\ & ((part\text{-}of, x, y) \vee (transient\text{-}part, x, y)) \wedge \\ & (available, x) \wedge \\ & (\forall z : (not\text{-}assembly, z) \vee (not\text{-}assemble\text{-}order, z, x, y) \vee \\ & \quad (incorporated, z, y)), \\ & \{(\ \top, \\ & \quad \{(incorporated, x, y)\}, \\ & \quad \{(available, x), (not\text{-}incorporated, x, y)\})\} \cup \\ & \{(\ (part\text{-}of, x, y) \wedge \\ & \quad (\forall z : (not\text{-}assembly, z) \vee (not\text{-}part\text{-}of, z, y) \vee \\ & \quad\quad (equal, z, x) \vee (incorporated, z, y)) \wedge \end{aligned}$$

$$
\begin{array}{l}
\quad (\forall z : (\mathit{not\text{-}assembly}, z) \vee (\mathit{not\text{-}transient\text{-}part}, z, y) \vee \\
\qquad (\mathit{not\text{-}incorporated}, z, y)), \\
\quad \{(\mathit{complete}, y), (\mathit{available}, y)\}, \\
\quad \emptyset)\})
\end{array}
$$

– *remove:* $\Sigma^2 \times 2^{\Sigma} \mapsto {\mathcal{A}_{adl}}^{\mathcal{P}}$ *defined by*

$$
\begin{array}{l}
\mathit{remove}(x, y, C) := (\ (\mathit{assembly}, x) \wedge (\mathit{assembly}, y) \wedge \\
\quad (\forall z : (\mathit{not\text{-}resource}, z) \vee (\mathit{not\text{-}requires}, y, z) \vee \\
\qquad (\mathit{committed}, z, y)) \wedge \\
\quad (\mathit{incorporated}, x, y) \wedge \\
\quad (\ (\mathit{transient\text{-}part}, x, y) \wedge \\
\qquad (\forall z : (\mathit{not\text{-}assembly}, z) \vee (\mathit{not\text{-}remove\text{-}order}, z, x, y) \vee \\
\qquad\quad (\mathit{incorporated}, z, y)) \vee \\
\qquad (\mathit{part\text{-}of}, x, y) \wedge \\
\qquad (\forall z : (\mathit{not\text{-}assembly}, z) \vee (\mathit{not\text{-}assemble\text{-}order}, z, x, y) \vee \\
\qquad\quad (\mathit{not\text{-}incorporated}, z, y))), \\
\qquad \{(\ \top, \\
\qquad\quad \{(\mathit{available}, x), (\mathit{not\text{-}incorporated}, x, y)\}, \\
\qquad\quad \{(\mathit{incorporated}, x, y)\})\} \cup \\
\qquad \{(\ (\mathit{transient\text{-}part}, x, y) \wedge \\
\qquad\quad (\forall z : (\mathit{not\text{-}assembly}, z) \vee (\mathit{not\text{-}part\text{-}of}, z, y) \vee \\
\qquad\quad\quad (\mathit{incorporated}, z, y)) \wedge \\
\qquad\quad (\forall z : (\mathit{not\text{-}assembly}, z) \vee (\mathit{not\text{-}transient\text{-}part}, z, y) \vee \\
\qquad\quad\quad (\mathit{equal}, z, x) \vee (\mathit{not\text{-}incorporated}, z, y)), \\
\qquad\quad \{(\mathit{complete}, y), (\mathit{available}, y)\}, \\
\qquad\quad \emptyset)\})
\end{array}
$$

and the set of instances is

$$
\begin{array}{l}
\mathcal{I} = \{\ (\ A \cup R, \\
\qquad \{(\mathit{equal}, x, x) \mid x \in A \cup R\} \cup \\
\qquad \{(\mathit{assembly}, x) \mid x \in A\} \cup \{(\mathit{not\text{-}assembly}, x) \mid x \in R\} \cup \\
\qquad \{(\mathit{resource}, x) \mid x \in R\} \cup \{(\mathit{not\text{-}resource}, x) \mid x \in A\} \cup \\
\qquad \{(\mathit{requires}, x, y) \mid x \in A, y \in \mathit{re}(x)\} \cup \\
\qquad \{(\mathit{not\text{-}requires}, x, y) \mid x \in A, y \in R \setminus \mathit{re}(x)\} \cup \\
\qquad \{(\mathit{part\text{-}of}, x, y) \mid y \in A, x \in \mathit{pa}(y)\} \cup \\
\qquad \{(\mathit{not\text{-}part\text{-}of}, x, y) \mid y \in A, x \in A \setminus \mathit{pa}(y)\} \cup \\
\qquad \{(\mathit{transient\text{-}part}, x, y) \mid y \in A, x \in \mathit{tr}(y)\} \cup \\
\qquad \{(\mathit{not\text{-}transient\text{-}part}, x, y) \mid y \in A, x \in A \setminus \mathit{tr}(y)\} \cup \\
\qquad \{(\mathit{assemble\text{-}order}, x, x', y) \mid y \in A, (x, x') \in \mathit{ao}(y)\} \cup \\
\qquad \{(\mathit{not\text{-}assemble\text{-}order}, x, x', y) \mid y \in A, (x, x') \in (A \times A) \setminus \mathit{ao}(y)\} \cup \\
\qquad \{(\mathit{remove\text{-}order}, x, x', y) \mid y \in A, (x, x') \in \mathit{ro}(y)\} \cup \\
\qquad \{(\mathit{not\text{-}remove\text{-}order}, x, x', y) \mid y \in A, (x, x') \in (A \times A) \setminus \mathit{ro}(y)\} \cup \\
\qquad \{(\mathit{available}, x) \mid x \in A, \mathit{pa}(x) = \emptyset\} \cup
\end{array}
$$

$\{(\mathit{not\text{-}incorporated}, x, y) \mid y \in A, x \in pa(y) \cup tr(y)\},$
$\{(\mathit{complete}, g)\})$
$\mid A \cup R \subseteq \Sigma, |A| \geq 2,$
$re : \{x \in A \mid pa(x) \neq \emptyset\} \mapsto 2^R,$
$pa : A \mapsto 2^A, (A, \{(x, y) \mid y \in A, x \in pa(y)\})$ *is tree with root* $g,$
$tr : \{x \in A \mid pa(x) \neq \emptyset\} \mapsto 2^A,$
$tr(y) \cap pa(y) = \emptyset$ *for all* $y,$
$ao : A \mapsto 2^{(A \times A)}, ao(y) \subseteq (pa(y) \cup tr(y)) \times (pa(y) \cup tr(y)),$
$(pa(y) \cup tr(y), ao(y))$ *is cycle-free for* $y \in A,$
$ro : A \mapsto 2^{(A \times A)}, ro(y) \subseteq (pa(y) \cup tr(y) \times tr(y),$
$(pa(y) \cup tr(y), ro(y))$ *is cycle-free for* $y \in A\}$

In the instances, the *pa* function defines the part-of relation, which resembles a tree where the root node is the goal assembly. Each non-leaf node of the tree can require an arbitrary amount of resources specified via the *re* function, and each non-leaf node can have arbitrary transient parts except its own parts, given by the *tr* function. The assemble order $ao(z)$ is a consistent partial ordering between parts and transient parts of an assembly z, and the remove order $ro(z)$ is a consistent partial ordering between (transient) parts and transient parts of an assembly z.

22.1 Solvability

We first make a few simple observations on solvable states, giving us an exact criterion of when an *Assembly* instance is solvable. Throughout the subsequent investigation we use, for convenience, the notations for an instance as introduced above. The following definition captures a kind of backbone structure to a specific reachable state in an *Assembly* instance, specifying which assemblies need still be assembled into which other ones.

Definition 56. *Given an instance* (C, I, G) *of the Assembly domain, and a reachable state* $s \in S$*. We define*

$$\begin{aligned}
NA_0(s) := \{(x, y) \mid\ & y \in A, (\mathit{complete}, y) \notin s, x \in pa(y), (\mathit{incorporated}, x, y) \notin s\} \cup \\
\{(x, y) \mid\ & y \in A, (\mathit{complete}, y) \notin s, x \in tr(y), (\mathit{incorporated}, x, y) \notin s, \\
& \exists\, z : z \in tr(y), (x, z) \in ro(y), (\mathit{incorporated}, z, y) \in s\} \\
NA_{i+1}(s) := \{(x, y) \mid\ & y \in A, (\mathit{complete}, y) \notin s, x \in tr(y), (\mathit{incorporated}, x, y) \notin s, \\
& \exists\, z : (z, y) \in NA_i(s), (x, z) \in ao(y) \cup ro(y)\} \\
NA(s) := \textstyle\bigcup_{i=0}^{\infty} & NA_i(s)
\end{aligned}$$

We call $NAG(s) := (A, NA(s))$ *the* needed assembles graph to s*, and* $NAG(I)$ *the* needed assembles graph to the instance*. In the following, a* leaf *in a graph will denote a node that has no ingoing edges, and a* relevant leaf *will denote a leaf that has at least one outgoing edge.*

Lemma 6. *Given an instance* (C, I, G) *of the Assembly domain with action set* A*, and a reachable state* $s \in S$ *such that* $NAG(s)$ *is cycle-free. If, for all relevant leafs* x *in* $NAG(s)$*,* $(\mathit{available}, x) \in s$*, then* (A, s, G) *is solvable.*

Proof. If $(\text{complete}, g) \in s$, we have nothing to prove. Else, as $NAG(s)$ is cycle-free, there are either no relevant leafs at all, or there is at least one non-leaf node y such that all sons of y are leafs. Say there are no relevant leafs at all. It follows that all parts, and transient parts needed for removing other transient parts, are incorporated into g already (because there are no ingoing edges to g), so one can remove the transient parts in an order obeying $ro(g)$ (note that $ro(g)$ is cycle-free by definition), after committing (and beforehand releasing, if necessary) the resources required by g.

In the other case, select a node y that is not a leaf, but for which all sons are leafs. Let us focus on y's ingoing edges $(x, y) \in NA(s)$. All x are available by prerequisite. Also, from the definition of $NA(s)$, the set of those x is all we need for assembling y: all parts of y, all transient parts needed for assembling that parts, all transient parts needed to remove other transient parts again, and all transient parts needed for assembling those, have an edge to y if they are not yet assembled. Say we commit (and beforehand release) all resources required by y, and carry out the respective sequence of assemble(x, y) actions (which can be applied in some order because $ao(y)$ is cycle-free by definition). We can then remove all transient parts from y by a remove(x, y) sequence (which can be applied because all parts with a remove order have been assembled). This gets us to a state s' where $(\text{complete}, y) \in s'$. Afterwards, we can also remove all parts of y again in a bottom-to-top order regarding $ao(y)$; note that this does not delete $(\text{complete}, y)$. We then have a state s'' where all assemblies that have been available in s are again available, and where there is one more complete assembly, namely y. The new assembles graph $NAG(s'')$ differs from $NAG(s)$ only in the sense that all edges (x, y) have been removed. The new graph is therefore again cycle-free; the only node that might be a new leaf is y, and $(\text{available}, y) \in s''$. We can thus iterate the argument and are finished with the monotonicity of completeness.

Lemma 7. *Given an instance (C, I, G) of the Assembly domain with action set A, and a reachable state $s \in S$. If $NAG(s)$ contains a cycle, then (A, s, G) is unsolvable.*

Proof. With $(x, y) \in NA(s)$, we know that y is not complete in s; it can also not be available initially, i.e., $pa(y) \neq \emptyset$, so it follows that y is not available in s. Further, all such assemblies y must be made complete (and with that available) at some point because the pa relation is a tree with root g. Finally, $(x, y) \in NA(s)$ implies that, in any solution plan from s, x must be available at some point *before* y can be made available. Putting all of this together, a cycle in $NAG(s)$ gives us a sequence $y_0, \ldots, y_{n-1}, y_n = y_0$ of assemblies such that y_i must in any solution from s be made available before y_{i+1} is made available, which is a contradiction.

Theorem 23. *Given an instance (C, I, G) of the Assembly domain. The instance is solvable if and only if the needed assembles graph $NAG(I)$ to the instance is cycle-free.*

Proof. Consequence of Lemmata 6 and 7 and the fact that initially, the leafs of the needed assemble graph are exactly the trivial assemblies x for which $pa(x) = \emptyset$. Those are available.

22.2 Some Observations Concerning Dead Ends

We now first focus on the role of dead ends in the *Assembly* domain. As it turns out, the hypothesis that all dead ends are recognized under h^+ is *not* correct: there can be unrecognized dead ends, which immediately also falsifies the other hypothesis that there are no local minima under h^+ in any *Assembly* instance. Have a look at Figure 2, where the part-of relations are depicted by dotted arrows, the transient part relations are depicted by dashed arrows, and the ordering relations are full arrows; for ordering relations, the kind of ordering and the respective higher level assembly are written at the left hand side of the relation, i.e., the beginning of the arrow.

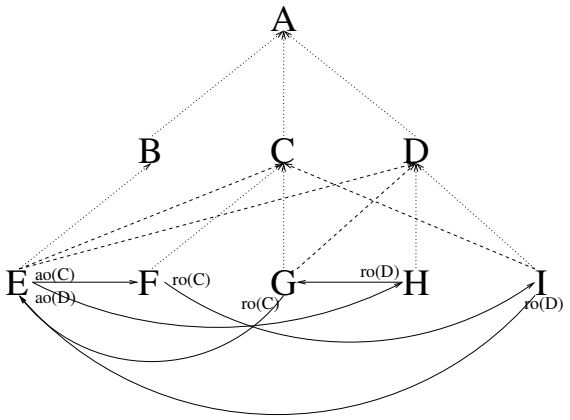

Fig. 2. An assembly instance where an unrecognized dead end arises in the situation where G is incorporated into D, and I is incorporated into C.

The situation s where G is incorporated into D and I is incorporated into C is an unrecognized dead end: either G or I must be removed first; say one tries G. This requires assembling E into D, in order to assemble H. But once E is assembled into D, one must assemble I to get it out again; this is, however, impossible, as I is at that point still assembled into C, and needs E to be assembled in the first place for being removed. A symmetrical situation arises when one tries to remove I from C first, starting from s. On the other hand, a relaxed plan for (A, s, G) can be constructed by assembling E into both C and D (the availability of E remains undeleted), then assembling F and H, then removing G and I, and so on.

We remark here that the instance depicted in Figure 2 does not exploit the full freeness allowed for the ordering relations in arbitrary *Assembly* instances; in fact, our example instance adheres to quite strong restrictions:

1. any assembly is only transient part of assemblies at a higher tree level;
2. $ao(y) \subseteq pa(y) \cup tr(y) \times pa(y)$ for all $y \in A$, i.e., there are no assemble orders before transient parts;
3. $ro(y) \subseteq pa(y) \times tr(y)$ for all $y \in A$, i.e., no transient parts are needed for removing other transients.

The above three restrictions are exactly those that are imposed on the *Assembly* instances produced by Hoffmann's random generator (note here that the first restriction implies solvability with Theorem 23). Thus, the above instance could have been produced by that generator (a description of the generator is available at `http://www.informatik.uni-freiburg.de/ hoffmann/ff-domains.html`).

One might wonder why amongst Hoffmann's hundred random instances there was no single unrecognized dead end, when there could have been theoretically. It is important to note here the role of *how* the instances are generated, i.e., how frequently assemble- and remove-orders appear. Like it is the case in the published benchmark suit, the random instances were generated such that assemble- and in particular remove-orders are quite seldom. We will prove below that, unless an instance contains a quite complex struture of remove order constraints, there are no dead ends at all.

The following definition formalizes the situation where some incorporated assembly needs other assemblies to be incorporated in order to be removed—this plays a key role in states where some parts that are needed elsewhere are "stuck" somewhere.

Definition 57. *Given an instance (C, I, G) of the Assembly domain, and a reachable state $s \in S$. For assemblies x such that $(incorporated, x, y) \in s$ for some y, we define the* needed removers $NR(x, s)$ for x in s *as:*

1. *if $x \in tr(y)$: those $z \in A$, $(incorporated, z, y) \notin s$ with $(z, x) \in ro(y)$, and*
2. *if $x \in pa(y)$: those $z \in A$, $(incorporated, z, y) \notin s$ such that there is some $z' \in tr(y), (z, z') \in ro(y)$ such that ex. $z' = z_0, \ldots, z_n = x$ with $(z_i, z_{i+1}) \in ao(y), (incorporated, z_i, y) \in s$ for $i \geq 0$ and $z_i \in pa(y)$ for $i \geq 1$.*

Lemma 8. *Given an instance (C, I, G) of the Assembly domain, and a reachable state $s \in S$. If $x_1, \ldots x_n$ are assemblies such that, for some y, $(incorporated, x_i, y) \in s$ for $1 \leq i \leq n$, and $NR(x_i, s) = \emptyset$ for all i, then there is an action sequence that, when applied to s, yields a state s' where $(available, x_i) \in s'$ for all i.*

Proof. First of all, one can commit (and beforehand, if necessary, release) the resources required by y. Then collect together as the relevant transients all $x_i \in tr(y)$, and all $z \in tr(y)$ that are transitively ordered before some $x_i \in pa(y)$

regarding $ao(y)$. As $NR(x_i, s) = \emptyset$, all parts that have a remove order before a relevant transient are incorporated into y. Thus, one can apply a sequence of actions removing all relevant transients, in some order respecting $ro(y)$ (this is posssible because $ro(y)$ is cycle-free by definition). In advance, all relevant parts—the $x_i \in pa(y)$ together with all $z \in pa(y)$ that are transitively ordered before some $x_i \in pa(y)$ regarding $ao(y)$—can be removed in some order respecting inverse $ao(y)$ (which is, again, cycle-free by definition).

If a part x is incorporated in y', but needed in y, and can not be removed (and thus made available) easily, then there is a constraint between y' and y in the sense that y' must be disassembled before one can care about y. The following definition formalizes that situation.

Definition 58. *Given an instance (C, I, G) of the Assembly domain, and a reachable state $s \in S$. The* remove conflicts $RC(s)$ of s *are defined as*

$$RC(s) := \{(y', y) \mid (complete, y) \notin s, \\ \exists\, x \in pa(y) \cup tr(y) : (incorporated, x, y') \in s, NR(x, s) \neq \emptyset\}$$

The next is the core lemma. We will later identify a situation, not unlikely given the way random instances were generated in Hoffmann's experiment, where the seemingly very strong prerequisite of the lemma is fulfilled. The proof is an extension of the proof to Lemma 6.

Lemma 9. *Given an instance (C, I, G) of the Assembly domain, with action set A. If, for all reachable states $s \in S$, $(A, NA(s) \cup RC(s))$ is cycle-free, then (A, s, G) is solvable for all such s, i.e., the instance is dead-end free.*

Proof. If all $a \in A$ have $(\text{complete}, a) \in s$, then we are finished. If all nodes in $(A, NA(s) \cup RC(s))$ are leafs, then in particular the goal assembly g has no ingoing edges in $NA(s)$, so we can make it complete by committing the required resources, and removing all transient parts.

In the non-trivial case, let y be a node in $(A, NA(s) \cup RC(s))$ that is not a leaf, but for which all sons are leafs. In particular, $(\text{complete}, y) \notin s$. We construct an action sequence leading to a state s' where $(\text{complete}, y) \in s'$. Iterating the argument proves the lemma with the monotonicity of completeness. The construction proceeds as follows.

1. Process all sons y', $(y', y) \in RC(s)$: let, for such a y', $x \in pa(y) \cup tr(y)$ be incorporated into y'. If y' was not complete, then $NR(x, s) \neq \emptyset$ would imply an ingoing edge to y' in $NA(s)$ (obvious if $x \in tr(y')$; if $x \in pa(y')$, there is an incorporated transient part z' for which a remover z must be incorporated). Thus, y' is complete in s, and thus, all $z \in pa(y') \cup tr(y')$ are complete in s. As y' has no ingoing edges in $RC(s)$, all such z that are incorporated into some y'' have $NR(z, s) = \emptyset$. We can thus apply the following action sequence:
   (a) remove, using Lemma 8, all $z \in pa(y') \cup tr(y')$ from those y'' were they are incorporated (if they are);

(b) commit the resources required by y', and assemble all $z \in pa(y') \cup tr(y')$ into y' in some order respecting $ao(y')$;
(c) completely disassemble y', by first removing all transients in an order respecting $ro(y')$, then removing all parts in an order respecting $ao(y')$ (note that this does not delete the completeness of y').

2. Process all sons x, $(x, y) \in NA(s), (\text{complete}, x) \notin s$: as such x have no ingoing edges in $NA(s)$, all parts $z \in pa(x)$ are incorporated, and all parts needed for removing transients are incorporated, too. So x can be made complete by committing the resources, and removing all transients in an order respecting $ro(x)$.
3. Process all sons x, $(x, y) \in NA(s), (\text{incorporated}, x, y') \in s, NR(x, s) = \emptyset$: with Lemma 8, all those x can be removed from the assemblies y'.

After the above steps have been carried out, it holds that all x with $(x, y) \in NA(s)$ are available: if, in s, they have been incorporated into some y' with $NR(x, s) \neq \emptyset$, then they have been removed from y' in the first step; if they have not been complete, they have been made so, and available, in step 2; if they have been incorporated into some y' with $NR(x, s) = \emptyset$, then they have been removed from y' in the third step. We can therefore now get to our desired state s' by committing the resources required by y, assembling all x with $(x, y) \in NA(s)$ into y in an order respecting $ao(y)$, and afterwards removing the transient parts, making y complete.

We now identify cases where Lemma 9 can be applied. First, if remove orders behave in the manner imposed by Hoffmann's generator, then the needed assembles graph is monotonic in the following sense.

Lemma 10. *Given an instance (C, I, G) of the Assembly domain, such that $ro(y) \subseteq pa(y) \times tr(y)$ for all $y \in A$. Then, for all reachable states $s \in S$, the needed assembles graph to s is a subgraph of the needed assembles graph to the instance, i.e., $NA(s) \subseteq NA(I)$ holds.*

Proof. With $ro(y) \subseteq pa(y) \times tr(y)$ for all y, the role of incorporated transients in a state is diminished to the effect that all removers are parts and must therefore be incorporated anyway. The lemma then follows because in the intial state, both the set of parts being not complete is maximal (all parts), and the set of pairs (x, y) where x is not incorporated into y is maximal (no part is incorporated into any other part). The other constituents of Definition 56 are static (the pa, tr, ao and ro functions). The proof proceeds by induction.

$NA_0(s) \subseteq NA_0(I)$ follows from the above, with the observation that the second set in the definition of $NA_0(s)$ always yields the empty set because of our prerequisite $ro(y) \subseteq pa(y) \times tr(y)$ for all y: for a pair (x, y) in the second set, $x \in tr(y)$ and $(x, z) \in ro(y)$ hold for some z, in contradiction. The induction step from $NA_i \longrightarrow NA_{i+1}$ is straightforward: in addition to I's maximality properties of non-completeness and not being incorporated, we know from the induction hypothesis that each $(z, y) \in NA_i(s)$ is also contained in $NA_i(I)$. This gives us $NA_{i+1}(s) \subseteq NA_{i+1}(I)$, and consequently $NA(s) \subseteq NA(I)$.

Second observation, obviously a remove conflict between y' and y implies that the respective ordering constraints are present between y' and y in the instance, i.e., there must be a potential remove conflict between those two in the following sense.

Definition 59. *Given an instance (C, I, G) of the Assembly domain. The* potential remove conflicts pRC of the instance *are defined as*

$$\begin{aligned} pRC := \{(y', y) \mid \exists\ x \in (pa(y) \cup tr(y)) \cap tr(y') : \exists\ z : (z, x) \in ro(y')\} \cup \\ \{(y', y) \mid \exists\ x \in tr(y) \cap pa(y') : \\ \exists\ \ z' \in tr(y), z_1, \ldots, z_n \in pa(y) : \\ (z', z_1) \in ao(y), (z_i, z_{i+1}) \in ao(y) \text{ for all } i, (z_n, x) \in ao(y), \\ \exists\ z : (z, z') \in ro(y)\} \end{aligned}$$

The set in the first line of the above definition has not many members if remove orders are seldom. Even more so for the second set, where there must be a complex construct of assemble orders in addition to the remove order. Now, if those sets of potential remove conflicts are cycle-free together with the needed assembles of an instance as generated by Hoffmann's generator (or rather, an instance where no remove orders are starting in transient parts), then the instance is dead-end free.

Theorem 24. *Given an instance (C, I, G) of the Assembly domain, with action set A. If $ro(y) \subseteq pa(y) \times tr(y)$ for all $y \in A$, and $(A, NA(I) \cup pRC)$ is cycle-free, then (A, s, G) is solvable for all $s \in S$, i.e., the instance is dead-end free.*

Proof. With Lemma 9, it suffices to show that $(A, NA(s) \cup RC(s))$ is cycle-free for all $s \in S$. With Lemma 10, we have $NA(s) \subseteq NA(I)$ for all such s. Furthermore, $RC(s) \subseteq pRC$ is obvious because any remove conflict (y', y) in s constitutes the potential remove conflict (y', y). We are therefore finished with our prerequisite that $(A, NA(I) \cup pRC)$ is cycle-free.

22.3 Some Observations Concerning Local Minima

There is more to be said about local minima in the *Assembly* domain. First, let us state that there can be local minima under h^+ even if there are no unrecognized dead ends. An example, adhering to the restrictions of Hoffmann's generator, is depicted in Figure 3. The meaning of the arrows is the same as in Figure 2.

The problem here is that, with D and G incorporated, $h^+(s) = 6$, assembling E or F does not improve the situation: while putting E into C retains $h^+(s') = 6$, assembling F into C afterwards worsens the situation to $h^+(s'') = 7$ because G needs to be removed from C due to its assemble order with respect to F. A symmetrical situation arises when assembling F into B first.

Once again, one wonders why no local minima occured in a hundred random instances. While we do not have a fully satisfactory answer to that, we *will* in the following make an interesting observation concerning the behavior of local minima under h^+ in *Assembly* instances, and the ease of solving such instances

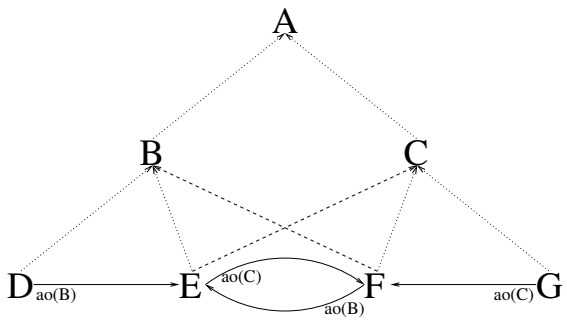

Fig. 3. An assembly instance where a local minimum arises when D is incorporated into B, and G is incorporated into C.

using that heuristic: unless there are complex structures concerning assemble orders, the *intial state* of an *Assembly* instance, and many other states, have a path to the goal on which h^+ decreases monotonically; in particular these states do not lie on local minima.

In the instance from Figure 3, the intial state does not have such a path. This is because there is a circle between B and C in the sense that they are conflicted (B, C) and (C, B) as follows.

Definition 60. *Given an instance (C, I, G) of the Assembly domain, and a reachable state $s \in S$. The* assemble conflicts $AC(s)$ of s *are defined as*

$$AC(s) := \{(y', y) \mid \exists\, x, z \in pa(y) : \begin{array}{l}(x, y) \in NA(s),\\ (z, x) \in ao(y),\\ (x, y') \in NA(s)\}\end{array}$$

$AC(I)$ *are the* assemble conflicts of the instance.

The problem with an assemble conflict between y' and y with regards to an assembly x is that x is needed for both y' and y, but once x is incorporated into y we will have trouble removing it because there is another part z with an assemble order before it. So one should care about y' first. If this is always possible (which it is not in the example Figure 3 because that kind of conflict appears between B and C in both directions), then states where all relevant leafs in the needed assembles graph are available have a monotonically decreasing path to the goal, under the condition that no needed assemble constraint is due only to a remove order.

Lemma 11. *Given an instance (C, I, G) of the Assembly domain with action set A, and a reachable state $s \in S$, such that for all relevant leafs x in $NAG(s)$, $(available, x) \in s$. If it holds that*

1. *for all $(x, y) \in NA(s)$, there is a sequence $x = x_0, x_1, \ldots, x_n$ such that $x_n \in pa(y)$, $(x_{i-1}, x_i) \in ao(y)$ and $(x_i, y) \in NA(s)$ for $1 \leq i \leq n$, and*

2. $(A, NA(s) \cup AC(s))$ *is cycle-free,*

then s *has a path to the goal on which* h^+ *decreases monotonically..*

Proof. We prove that h^+ decreases monotonically on a solution path that can be obtained by Lemma 6 ($NAG(s)$ is cycle-free by prerequisite 2). This iteratively processes an arbitrary non-leaf assembly y where all sons x are leafs in $NAG(s)$; if all assemblies are leafs in $NAG(s)$, achieving the goal can be done by committing (and beforehand releasing) the resources required by the goal assembly g, and then removing the transient parts from g, all of which actions are contained in an optimal relaxed solution, and delete only facts that are not needed anymore. We change the algorithm from Lemma 6 to iteratively process nodes y where all sons are leafs in $(A, NA(s) \cup AC(s))$. If a node has no ingoing edges in $NA(s)$, then it has no outgoing edges in $AC(s)$ ($(y', y) \in AC(s)$ implies, by definition, $(x, y') \in NA(s)$ for some x), so the only ingoing edges (x, y) of y are such that $(x, y) \in NA(s)$. The solution algorithm first releases and commits all resources required by y, then assembles all x with $(x, y) \in NA(s)$ into y in an appropriate order, and afterwards removes all transient parts and parts of y in an appropriate order. The process ends up in a state where prerequisites 1 and 2 are again fulfilled; for prerequisite 1, this is obvious (all edges (x, y) are removed from $NA(s)$, and the edges (x, y') for other assemblies y' remain unchanged), and for prerequisite 2 it holds because no new edges are introduced into $NA(s)$, which implies that no new edges are introduced into $AC(s)$. So let us consider the optimal relaxed solution lengths on the constructed path. Removing an assembly x from y only deletes (incorporated, x, y), which is not needed anymore once (complete, y) has been achieved; it is also not needed anymore when all parts of y have been incorporated and x is a transient which is not needed for removing any other transient, both of which conditions are fulfilled when the algorithm removes a transient part x from y. As y is not complete, the required resources must be released and committed to y at some point in an optimal relaxed solution from s; as releasing and committing actions are invertible, those our actions do therefore not increase optimal relaxed solution length by Lemma 3. Thus, the only actions left to take care of are the assemble(x, y) parts of the process, transforming a state s' into another state s''. It suffices to show that there is a relaxed solution to s'' that is at most as long as an optimal relaxed solution to s'. Say $P^+_{s'}$ is an optimal relaxed solution to (A, s', G). x may be a part, or a transient part of y.

In the first case, assemble(x, y) is contained in $P^+_{s'}$ simply because x must be incorporated into y at some point. Note that $(x, y) \in NA_0(s)$. If there is no $z \in pa(y)$, $(z, x) \in ao(y)$, then a relaxed plan $P^+_{s''}$ for (A, s'', G) can be constructed by removing assemble(x, y) from $P^+_{s'}$, and placing remove(x, y) up front. If there is such a z, then, because y has no ingoing edges from $AC(s)$, it follows that there is no edge (x, y') in $NA(s)$ (otherwise, $(y', y) \in AC(s)$ would hold), so (available, x) is not needed anymore; neither do we need (not-incorporated, x, y) when x is a part (not a transient part) of y, so we can construct $P^+_{s''}$ by removing assemble(x, y) from $P^+_{s'}$ and are finished.

In the second case, assemble(x, y) is contained in $P^+_{s'}$ because by prerequisite 1 x must be assembled into y at some point for the integration of all parts. So our first step in the construction of a relaxed plan $P^+_{s''}$ for (A, s'', G) is to remove assemble(x, y) from $P^+_{s'}$. It then suffices to show that remove(x, y) can be applied at some point in the resulting relaxed plan before (available, x) or (not-incorporated, x, y) are needed. This follows again from prerequisite 1: remove(x, y) can be applied in $P^+_{s''}$ as soon as all z with $(z, x) \in ro(y)$ have been incorporated into y. With prerequisite 1, all z that have this property must already be incorporated, or assembled into y at some point. All these actions—and the actions assembling their necessary (transient) parts $(z', z) \in ao(y)$—can be moved to the front of $P^+_{s''}$ in the appropriate order. We insert remove(x, y) directly after those actions. This is early enough: obviously, none of the assemble(z, y) actions needs (available, x) as prerequisite; for (not-incorporated, x, y), matters are a bit more complicated, as the last of the assemble(z, y) actions might have been responsible for adding (complete, y); but in that case, all parts of y are incorporated at that point, and all transient parts except x have not been incorporated in s', so remove(x, y) achieves (complete, y) at that point of $P^+_{s''}$ starting from s''. Our construction $P^+_{s''}$ is therefore a relaxed plan for (A, s'', G).

Let us identify situations where the prerequisites of Lemma 11 are fulfilled. First, observe that, under the restrictions imposed by Hoffmann's generator on remove orders, the assemble conflicts are monotonic, simply because the needed assembles are.

Lemma 12. *Given an instance (C, I, G) of the Assembly domain, such that $ro(y) \subseteq pa(y) \times tr(y)$ for all $y \in A$. Then, for all reachable states $s \in S$, $AC(s) \subseteq AC(I)$ holds.*

Proof. Follows directly from Lemma 10 and the definition of assemble conflicts (which appear more often the more edges there are in the needed assembles graph to a state).

The same restriction on remove orders implies the other prerequisite of Lemma 11—no needed assembles due only to remove orders—so we know that, under that restriction, the initial state has a monotonically decreasing path to the goal unless there is a cycle in the assemble conflicts with the needed assembles.

Theorem 25. *Given an instance (C, I, G) of the Assembly domain, such that*

1. *$ro(y) \subseteq pa(y) \times tr(y)$ for all $y \in A$, and*
2. *$(A, NA(I) \cup AC(I))$ is cycle-free.*

Then, any reachable state $s \in S$ where all relevant leafs x in $NAG(s)$ are available, in particular the initial state, has a path to the goal on which h^+ decreases monotonically.

Proof. The prerequisites 1 and 2 are sufficient for prerequisites 1 and 2 of Lemma 11, respectively: with $ro(y) \subseteq pa(y) \times tr(y)$ for all $y \in A$, the only edges (x, y) in $NA(s)$ inserted due to a remove order are such that $x \in pa(y)$, so those edges are contained in $NA_0(s)$ anyway; the other implication follows directly from Lemmata 10 and 12.

Finally, a short remark on the existence of assemble orders: like for the remove orders, it is easy to identify potential assemble conflicts, which must be present for an assemble conflict to come up. So if those potential assemble conflicts—which we do not define explicitly due to their trivial nature—do not form a cycle with the needed assemblies, then Theorem 25 can be applied.

Proposition 8. *Given an instance (C, I, G) of the Assembly domain, and a reachable state $s \in S$. Then, $AC(s) \subseteq \{(y', y) \mid \exists\, x \in tr(y') \cap pa(y), z \in pa(y) : (z, x) \in ao(y)\}$ holds.*

Proof. Simply because an edge $(y', y) \in AC(s)$ implies the existence of x and z as specified above.

23 The Taxonomy

Our results together with the negative examples found in Hoffmann's empirical investigation [13] prove the picture specified in Figure 4.

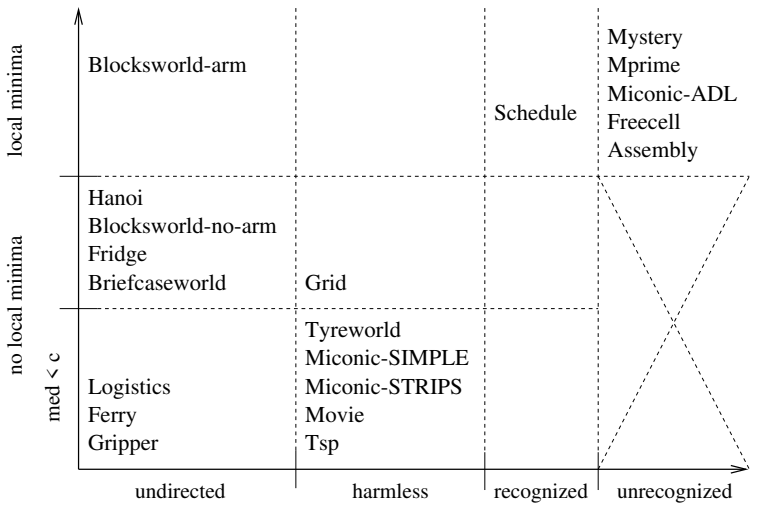

Fig. 4. The extended and revised planning domain taxonomy, overviewing our results.

All of the competition domains belong to exactly those parts of the taxonomy, shown in Figure 4, where Hoffmann's observations suggested to put them; except

the *Assembly* domain. Obviously, most of the domains are to be found near the left bottom side of the taxonomy.

As discussed by Hoffmann, from the point of view of heuristic planners like HSP or FF which derive their heuristic function by approximating h^+, the left bottom side of the taxonomy is intuitively the "simple" corner: there are no or only recognized dead ends, and h^+ is a heuristic of high quality. In contrast, the top right corner of the taxonomy contains the "demanding" domains: there can be dead ends that are not recognized by h^+. Consistently with this, the *Freecell* and *Miconic-ADL* domains constituted much more of a problem to the heuristic planners in the AIPS-2000 competition than, for example, the *Logistics* domain did. More intriguingly, we have seen that FF's search algorithm is polynomial when the heuristic has the quality corresponding to the lowermost part of the taxonomy, i.e., a limited maximal exit distance. For 8 of our 20 domains, h^+ does in fact have this quality.

24 Conclusion and Outlook

Looking at a large collection of commonly used benchmark domains, we have proven that in the majority of these domains the h^+ heuristic function has the qualities hypothesized by Hoffmann [13]. As many current state-of-the-art heuristic planners work by approximating that same h^+ function, this suggests to interpret those planner's success as utilizing the quality of h^+.

In the process of proving our results, we have also determined the reasons behind the quality of h^+, from a more structural point of view: the reasons are mainly that in many domains all actions are (at least) invertible or need not be inverted (like when their adds are static and their deletes are irrelevant); and that the actions are respected by the relaxation, i.e., often there is simply no other way to achieve the goal than by applying them. The knowledge about these structural implications may be useful when designing new benchmarks. Moreover, one might try to recognize the structural properties automatically, and thereby predict the performance of planners like FF. In fact, most of the definitions given in our theoretical core, Section 6, are purely syntactical. Lemma 37 gives a new sufficient criterion for recognizing planning tasks where there are no dead ends in the state space. The only non-syntactical prerequisite of Lemma 37 is inconsistency, for which there are several good approximation techniques in the literature [7, 9, 18]. The challenge is how to determine that an action is respected by the relaxation, and thereby recognize tasks where h^+ does not yield local minima.

The main piece of work left to do is to corroborate on the argumentation that planners like FF and HSP are in essence utilizing the quality of h^+—that is, it must be verified to which extent those planner's approximative heuristic functions really have the same quality as h^+. Hoffmann [13] reports that FF's heuristic function is similar to h^+ on his collection of small examples. To verify this observation in general, one can look at fragments of the state spaces of larger

planning tasks, and compute statistics about the distribution of local minima etc.

Another interesting future direction is to try and prove properties of h^+ for domain *classes* rather than for single domains in turn. Recent work by Malte Helmert [11] has defined a hierarchy of transportation domains in the context of investigating their complexity. Judging from our results, it seems to be the case that, with Helmert's terminology, any transportation domain where there is unlimited fuel does not contain local minima under h^+. The main difficulty in such an investigation is the definition of the domain class: in difference to Helmert who focusses on the *semantics* of the domains, for our purposes we need a strictly *syntactical* definition: after all, h^+ depends directly on the syntax of the operators.

Looking at Helmert's results (some of which are yet only to be found in his master's thesis [10]), there is one more intriguing parallelity to our investigation: the domains in the top right corner of our taxonomy coincide, except the *Assembly* domain, exactly with those domains where Helmert found that deciding plan existence is NP-hard. The question is whether there is any provable phenomenon behind this observation.

References

1. *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence (AAAI-99)*, Austin, TX, July 2000. MIT Press.
2. Fahiem Bacchus. The AIPS'00 planning competition. *The AI Magazine*, 22(3):47–56, 2001.
3. Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.
4. Blai Bonet, Gábor Loerincs, and Héctor Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence (AAAI-97)*, pages 714–719. MIT Press, July 1997.
5. Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.
6. Stefan Edelkamp and Malte Helmert. The model checking integrated planning system (MIPS). *AI Magazine*, 22(3):67–71, 2001.
7. Maria Fox and Derek Long. The automatic inference of state invariants in tim. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.
8. Jeremy Frank, Peter Cheeseman, and John Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.
9. A. Gerevini and L. Schubert. Inferring state constraints in DISCOPLAN: Some new results. In AAAI-00 [1], pages 761–767.
10. Malte Helmert. On the complexity of planning in transportation and manipulation domains. Master's thesis, Albert-Ludwigs-Universität Freiburg, 2001. Postscript available at http://www.informatik.uni-freiburg.de/~ki/theses.html.
11. Malte Helmert. On the complexity of planning in transportation domains. In *Recent Advances in AI Planning. 6th European Conference on Planning (ECP'01)*, pages 349–360, Toledo, Spain, September 2001. Springer-Verlag.

12. Jörg Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. Submitted to AIPS-02.
13. Jörg Hoffmann. Local search topology in planning benchmarks: An empirical analysis. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 453–458, Seattle, Washington, USA, August 2001. Morgan Kaufmann.
14. Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.
15. Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000.
16. Jana Koehler and Jörg Hoffmann. On the instantiation of ADL operators involving arbitrary first-order formulas. In *Proceedings ECAI-00 Workshop on New Results in Planning, Scheduling and Design*, 2000.
17. Drew McDermott. The 1998 AI planning systems competition. *The AI Magazine*, 21(2):35–55, 2000.
18. Jussi Rintanen. An iterative algorithm for synthesizing invariants. In AAAI-00 [1], pages 806–811.