University of Basel

Generalization of Cycle-Covering Heuristics

Master's Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence
http://ai.dmi.unibas.ch

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Thomas Keller

Clemens Büchner
clemens.buechner@unibas.ch
2015-059-603

April 20, 2020

Acknowledgments

This thesis would not have been such a personal success without the help of so many people. I am very grateful for all the support and encouragement I received during this time.
First of all, I thank Thomas Keller for his advice and all the time he put into this. The insights we gained during our endless discussions were invaluable for making great progress. I owe my thanks to Florian Pommerening who has joined our meetings and inspired me to dig deeper. His help with the code base of Fast Downward at all times is also greatly appreciated. I thank Prof. Dr. Malte Helmert and the whole AI research group at the University of Basel for this great opportunity and their assistance.
I am endlessly grateful for Stefanie who has participated in all my ups and downs during this thesis and the entire degree. Thank you very much for all the support and sacrifice, the suggestions and criticism, and last but not least the encouragement.
I also appreciate the effort invested by all my other proofreaders, discussion partners, and most of all friends. Thank you Andrea, Augusto, Manuel, Michael, Songül, and Yanick for your detailed feedback and corrections which have helped me improve the thesis.
Last but not least, I thank my family and friends for the mental support and all your caring.

Abstract

In this thesis, we consider cyclical dependencies between landmarks for cost-optimal planning. Landmarks denote properties that must hold at least once in all plans. However, if the orderings between them induce cyclical dependencies, one of the landmarks in each cycle must be achieved an additional time. We propose the generalized cycle-covering heuristic which considers this in addition to the cost for achieving all landmarks once.
Our research is motivated by recent applications of cycle-covering in the Freecell and logistics domain where it yields near-optimal results. We carry it over to domain-independent planning using a linear programming approach. The relaxed version of a minimum hitting set problem for the landmarks is enhanced by constraints concerned with cyclical dependencies between them. In theory, this approach surpasses a heuristic that only considers landmarks.
We apply the cycle-covering heuristic in practice where its theoretical dominance is confirmed; Many planning tasks contain cyclical dependencies and considering them affects the heuristic estimates favorably. However, the number of tasks solved using the improved heuristic is virtually unaffected. We still believe that considering this feature of landmarks offers great potential for future work.

Table of Contents

1 Introduction

Sometimes, one needs to take a step back in order to move forward. In essence, this is what this thesis aims to show for planning problems. But in order to do so, we first need to take a step back ourselves and start at the very basics of what planning is.

Classical planning formalizes the problem of finding a way to achieve a desired outcome. Such problems are specified as follows: given an *initial state*, which sequence of *actions* changes the state so that the *goal* is reached? The solution is a sequence of actions called a *plan* that ends in a *goal state*. In *cost-optimal* planning the aim is to find the cheapest plan using actions with associated *costs*.

Planning is PSPACE-complete. We use *heuristics* to provide guidance when searching for the *optimal plan*. Heuristics are functions that estimate the cost of reaching the goal from a given state. Search algorithms explore the *state space* by applying one action at a time. If the heuristic estimate is lower after applying an action this indicates that the corresponding state is closer to the goal and should be explored further. In cost-optimal planning heuristics which provide high estimates but do not overestimate the true cost are preferable.

In this thesis we are interested in a domain-independent version of the cycle-covering heuristics as proposed by Paul and Helmert (2016). Cycle-covering is a concept based on *landmarks* for planning. Each landmark describes a property that must be part of every plan for a given problem. It is also possible to obtain *landmark orderings* which represent dependencies between landmarks. Thus, they provide a sequence in which the landmarks must be processed in order to find a plan. If these orderings entail cyclical dependencies, it means that a *deadlock* is encountered. Deadlocks are situations where each object of a set is blocked. Every such object is dependent on another object from the set to move first in order to make progress. Resolving a deadlock requires an action that does not directly contribute towards the goal. If deadlocks are present, finding optimal solutions for planning tasks is especially hard (Gupta and Nau 1992). The principle of cycle-covering considers such deadlocks in addition to the landmark information in order to increase the heuristic estimates.

Previous work on landmarks and landmark orderings has neglected cyclical dependencies due to their increased complexity (Hoffmann, Porteous, and Sebastia 2004; Richter 2010). In this thesis we show that the information that is lost when discarding cycles can be of much value. Our approach, inspired by Paul, Röger, Keller, and Helmert (2017), recognizes

the need for landmarks to occur multiple times in a plan if cycles are present. Remember what we stated in the beginning: in order to move forward it is sometimes necessary to take a step back. In terms of landmarks, a plan must cross the same waypoint twice in order to resolve the cyclical dependencies between them. We propose the generalized cycle-covering heuristic h^{cycle} which uses a *linear programming* approach to evaluate the encountered states. The problem encoding is based on a *minimum hitting set* for the landmarks enhanced by constraints which represent the cyclical aspect.
We provide the theoretical basics to compute the cycle-covering heuristic based on *landmark graphs*. Furthermore, we show in an experimental evaluation that it leads to improved heuristic values in practice. Cycles in the landmark graphs of the analyzed planning tasks are not a rare occurrence. We use benchmark sets based on planning tasks from the International Planning Competition (IPC). Our evaluations support the claim that valuable information is contained in cycles although our approach does not outperform the state of the art.
In this thesis, we describe our work on a generalized, domain-independent cycle-covering heuristic. We first provide a background on classical planning (Chapter 2) and landmarks (Chapter 3). Landmarks in planning entail two different kinds of research areas. On the one hand, there is the question of how to approximate them for a given state in a planning task. On the other hand, the question of interest is how a set of landmarks can be used in order to solve the planning task for which they were created. Our contributions aim for the latter of these problems, using landmark approximation methods suggested in the literature. We explain how linear programming can be used to compute heuristic estimates based on landmarks (Chapter 4). Based on this knowledge we present the details of our cycle-covering heuristic and show that it dominates the analogous landmark heuristic which disregards cycles (Chapter 5). The ordering-aware cycle-covering heuristic h^{ord} uses stronger constraints in the linear program to improve the heuristic further. We implement these heuristics and evaluate them empirically (Chapter 6). Lastly, we show how integer programming influences the results of the cycle-covering heuristic (Chapter 7) and discuss ways to use our findings for future work (Chapter 8).

2 Classical Planning

In this chapter we formalize the planning problem as it is studied in artificial intelligence research. In particular, we consider *classical planning* restricting the problems to be fully observable, meaning that no information is hidden to the planning agent (e.g., Chess is fully observable whereas Poker is not, because it involves hidden cards).[1] Furthermore, only deterministic actions are present in classical planning, meaning there is no element of chance in the outcome of taking an action (e.g., Chess is deterministic whereas Backgammon is not, because it involves dice rolling).

2.1 Planning Tasks and State Spaces

In the following, we consider planning in the SAS$^+$ planning formalism (Bäckström and Nebel 1995). Definition 1 is adapted from Richter (2010).

Definition 1 (Planning task)**.** *A* SAS$^+$ planning task *is a 4-tuple* $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A} \rangle$ *with the following components:*

- $\mathcal{V}$ *is a finite set of* finite-domain state variables*. A* fact $v \mapsto d$ *is a value assignment of value* $d \in \mathrm{dom}(v)$ *to* $v \in \mathcal{V}$*, where* $\mathrm{dom}(v)$ *is the finite domain of* v*. (We use set notation such as* $v \mapsto d \in s$ *and function notation such as* $s(v) = d$ *interchangeably.) A* partial variable assignment *is a set of facts, each with a different variable. A* state *is a variable assignment defined on all variables in* $\mathcal{V}$*.*
- s_0 *is a state called the* initial state*.*
- G *is a partial variable assignment called the* goal*.*
- $\mathcal{A}$ *is a finite set of* actions *(also referred to as* operators*), each associated with two partial variable assignments* $pre(a)$ *and* $eff(a)$*. The facts in* $pre(a)$ *and* $eff(a)$ *are called the* preconditions *and* effects *of action* $a \in \mathcal{A}$*, respectively. Each action furthermore has an associated non-negative* cost $cost(a)$*.*

[1] Even though games are not classical planning problems, they are suitable for illustrating the described properties.

An action $a \in \mathcal{A}$ is *applicable* in a state s if $s(v) = d$ for all $v \mapsto d \in pre(a)$. Applying an applicable a in s results in $s' = s[\![a]\!]$ where $s'(v) = d$ for all $v \mapsto d \in \mathit{eff}(a)$ and $s'(v) = s(v)$ otherwise. An *action sequence* $\pi = \langle a_1, \ldots, a_n \rangle$ is applicable in $s = s_1$ if $s_{i+1} = s_i[\![a_i]\!]$ for $i = 1, \ldots, n$ and each action a_i is applicable in s_i. The state that results from applying π in s is written as $s[\![\pi]\!]$.

An *s-plan* is an action sequence π such that $G \subseteq s[\![\pi]\!]$. With Π_s we refer to the set of all s-plans. If s is clear from the context, we use the term *plan* and Π is the set of all such plans (e.g., the s_0-plans of a planning task $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A} \rangle$). The *cost* of a plan $\pi = \langle a_1, \ldots, a_n \rangle$ is the sum over the action costs of the sequence: $cost(\pi) = \sum_{i=1}^{n} cost(a_i)$. A plan π is *optimal* (denoted as π^*) if it has minimal cost among all plans: $\pi^* \in \arg\min_{\pi \in \Pi} [cost(\pi)]$.

Every planning task induces a state space. It displays the full list of states together with the possible actions between them.

Definition 2 (State space)**.** *The planning task $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A} \rangle$ induces a* state space $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ *as follows.*

- *S is the set of all* states *over $\mathcal{V}$.*
- *A is the set of actions $\mathcal{A}$.*
- *$cost : A \to \mathbb{R}_0^+$ are the* action costs*.*
- *$T = \{\langle s, a, s' \rangle \mid s, s' \in S, a$ applicable in $s, s' = s[\![a]\!]\}$ is the* transition relation *from one state to another through an applicable action.*
- *$s_0 \in S$ is the* initial state*.*
- *$S_\star = \{s \in S \mid G \subseteq s\}$ is the set of* goal states*.*

2.2 Heuristic Search

One approach to solving planning tasks is heuristic search. A *heuristic* is a function that estimates the cost of a state s to the nearest goal state (i.e., the cost of the optimal s-plan). It is used by search algorithms to evaluate promising actions and avoid irrelevant areas of the corresponding state space. In this thesis, we always use the A* search algorithm (Hart, Nilsson, and Raphael 1968). States are *expanded* by generating successor states for all applicable actions. Repeatedly, the state with the lowest total cost of the path to reach it combined with its heuristic estimate is expanded to eventually reach a goal state. When the first goal state is expanded, the search terminates. If no goal state exists (or none is reachable), A* exhaustively expands all reachable states before it terminates without finding a solution or only dead-end states remain (i.e., all states s where $h(s) = \infty$ are dead-ends). We say that a heuristic h_1 *dominates* another heuristic h_2 if $h_1(s) \geq h_2(s)$ for all s. By h^* we refer to the perfect heuristic which returns the cost of an optimal s-plan for all states $s \in S$ or ∞ if no plan exists. Computing h^* is as hard as planning which is PSPACE-complete (Bylander 1991). A heuristic h is called *admissible* if it never overestimates the true distance to the goal (i.e., it is dominated by h^*). A* guarantees to find an optimal plan when an admissible heuristic is used (Hart et al. 1968).

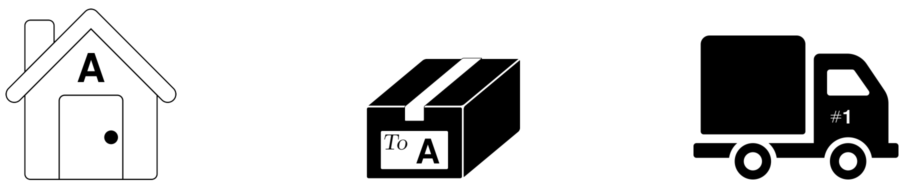
A heuristic is called *path-dependent* if the estimated cost of an s-plan depends on the path π from s_0 to s. The according evaluation function $h(s, \pi)$ is not a heuristic in the usual sense, but can be used as such. If a set of paths Π to s are considered in the heuristic (i.e., $h(s, \Pi)$), it is called *multi-path-dependent.* This may increase the information considered to estimate the cost of s-plans and can thus lead to more accurate heuristics.

2.3 Problem Domains

The formalization of planning tasks as given in Definition 1 gives a common ground to a wide range of specific problems. In planning, we talk about a problem *domain* to denote a specific class of problems. Each domain has its own specified rules of which predicates exist and how they interact with each other. All problems within the same domain are considered similar, which is not necessarily the case for two problems of completely different domains.

2.3.1 Logistics

In this thesis there will be example planning tasks explaining some ideas, properties, or problems. Instead of formally describing them every time, we hereby introduce a format that we use whenever suited. The domain of our examples is a simplified version of the commonly used logistics domain (McDermott 2000). Figure 2.1 shows the available objects.



(a) Location named A (b) Package with destination A (c) Truck with ID 1

Figure 2.1: Objects available in our simplified logistics domain.

The logistics domain describes problems where packages need to be transported between locations. There are vehicles which can execute this task. A package is either at a location or loaded onto a truck. Trucks can move freely between locations (i.e., routes are connecting every location with all other locations directly). The goal is to bring all packages to their destination location. A logistics planning task can have an arbitrary number of locations, packages, and trucks.[2]

[2] The original logistics domain is more complex than the version we use. Locations and trucks are associated with a city and trucks can only move between locations in their city. Each city has an airport (which is also a location) and airplanes can transport packages between the airports of different cities.

Definition 3 (Logistics problem)**.** *A* logistics problem *is specified by a 5-tuple* $\langle L, P, T, \mathit{init}, \mathit{goal}\rangle$ *as follows:*

- *L, P, and T are the sets of locations, packages, and trucks.*
- *$\mathit{init} : P \cup T \to L$ assigns a location to every package and truck to denote where they are located initially.*
- *$\mathit{goal} : P \to L$ maps each package to its destination.*

Every logistics problem induces a planning task $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A}\rangle$ according to the following rules.

- The set of variables $\mathcal{V}$ has one entry per package and truck. There is a variable v_p for each package $p \in P$ with $\mathrm{dom}(v_p) = L \cup T$. Furthermore, there is a variable v_t for every truck $t \in T$ so that $\mathrm{dom}(v_t) = L$.
- The initial state is $s_0 = \bigcup_{o \in P \cup T}\{v_o \mapsto \mathit{init}(v_o)\}$.
- The goal of the planning task is to bring all packages to their destination: $G = \bigcup_{p \in P}\{p \mapsto \mathit{goal}(p)\}$.
- The set of actions $\mathcal{A}$ consists of the following actions.
  - There is an action for every truck to move between any two (distinct) locations: For all $\langle t, l_1, l_2\rangle \in T \times L \times L$ where $l_1 \neq l_2$ there is an action $\mathit{move}(t, l_1, l_2)$. It has $\mathit{pre}(\mathit{move}(t, l_1, l_2)) = \{t \mapsto l_1\}$ and $\mathit{eff}(\mathit{move}(t, l_1, l_2)) = \{t \mapsto l_2\}$.
  - For every package, there is an action to load that package onto a truck at each location. For $\langle p, t, l\rangle \in P \times T \times L$ we have $\mathit{load}(p, t, l)$ with $\mathit{pre}(\mathit{load}(p, t, l)) = \{p \mapsto l, t \mapsto l\}$ and $\mathit{eff}(\mathit{load}(p, t, l)) = \{p \mapsto t\}$.
  - Similarly, there is an unload action $\mathit{unload}(p, t, l)$ for all $\langle p, t, l\rangle \in P \times T \times L$. It has $\mathit{pre}(\mathit{unload}(p, t, l)) = \{p \mapsto t, t \mapsto l\}$ and $\mathit{eff}(\mathit{unload}(p, t, l)) = \{p \mapsto l\}$.

  All these actions have uniform cost, i.e., $\mathit{cost}(a) = 1$ for all $a \in \mathcal{A}$. There is no limitation to the number of packages carried by a truck at the same time.

For the remainder of this thesis we do not give the planning tasks for logistics examples explicitly. Instead we sketch them using the symbols from Figure 2.1. Every truck and package is placed in front of a location to denote its initial value. The goal is given by the destinations marked on all packages. The sets of variables and actions are implied by the number of locations, packages, and trucks as listed above. Variables are named according to their identifications displayed in the example figures. The objects from Figure 2.1 would be named A for the location and t_1 for the truck. The package $p_{L\to A}$ would be sketched in front of another location L. We will not use examples where two packages have the same origin and destination, which would lead to ambiguous variable names under this naming policy.

2.3.2 Domain-Independent Planning

If a planning domain is understood well enough, a good and simple heuristic may be designed easily. The application of such a heuristic is limited to problems in this domain, though, because it is usually based on specific characteristics absent in other domains. However, if the domain is known and shared by all planning tasks one aims to solve, this approach is a reasonable choice. For example, a package delivery service will always encounter the same class of problems, namely *logistics*. A heuristic for logistics could count how many packages have not been delivered yet. This heuristic could be further improved upon by incorporating more knowledge about this domain. Paul et al. (2017) applied this technique for solving logistics tasks using a *cycle-covering* heuristic. The idea is based on another *domain-dependent* heuristic for solving Freecell problems optimally (Paul and Helmert 2016).
If the domain is unspecified or one wants to solve problems in various domains using the same idea, a *domain-independent* heuristic is what they aim for. The logistics heuristic described above is inapplicable in Freecell problems because it is unclear what an “undelivered package” relates to in a card-game. Even though the heuristic might yield satisfying results in the domain it is designed for, it is inapplicable in others. The STRIPS (or goal-count) heuristic (Fikes and Nilsson 1971) is a domain-independent approach similar to the logistics heuristic; instead of undelivered packages, it counts the number of predicates that are different from the goal.
In this thesis we introduce a domain-independent version of the cycle-covering heuristic suggested in the aforementioned papers. Since this is quite an abstract concept, we only use the simplified logistics domain for our examples. However, our heuristics are also applicable in other domains.

3 Landmarks and Landmark Orderings

Landmarks play a crucial part in this thesis. They are properties shared by all plans of a planning task. In this chapter we discuss fundamental landmark background, methods to find landmarks, and literature that relates to our research.

3.1 Landmarks

As outlined above, landmarks in planning describe properties that are shared by all plans. Porteous, Sebastia, and Hoffmann (2001) originally introduced the idea to identify facts which must be true at some point in every plan of a planning task. All facts which hold in the initial state, as well as facts of the goal, are trivial *fact landmarks.* It is possible to deduce further fact landmarks through logical reasoning. For example, in the logistics problem sketched in Figure 3.1 one package needs to be transported from B to C. As the truck must pick up the package at B, a non-trivial fact landmark for this planning task is $t_1 \mapsto B$. Although not required by the goal, $t_1 \mapsto C$ is also a landmark because the package must be unloaded from the truck at location C in every plan. Fact landmarks can be interpreted as sub-goals, which must be achieved in all plans for a given planning task.

Just as facts that must hold at some point in every plan can be deduced, it is possible to identify actions that must be part of every plan. All packages which are not initially at their destination must be loaded and unloaded to solve the planning task. Thus, the actions $\mathit{load}(p_{B \to C}, t_1, B)$ and $\mathit{unload}(p_{B \to C}, t_1, C)$ are part of every plan for the planning task in Figure 3.1. Such a property of a planning task is called an *action landmark* and is first described by Zhu and Givan (2003).

In the example, $\mathit{move}(t_1, A, B)$ is not an action landmark because it is possible to move to B via C. This is sub-optimal in the example, but we cannot generally make such a deduction. We can still capture the need to move to B by allowing landmarks of the form "either move from A to B or move from C to B". Such landmarks are called *disjunctive action landmarks* (Helmert and Domshlak 2009). They can contain any number of alternatives of which at least one must appear in every plan.

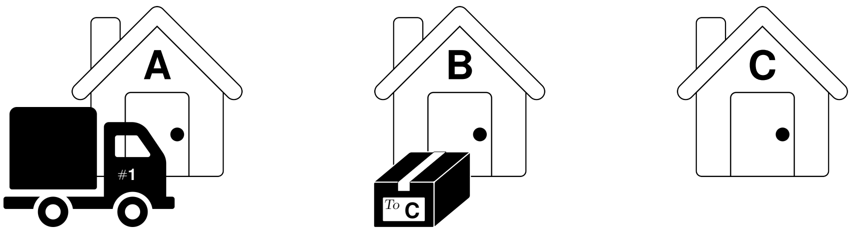


Figure 3.1: Example planning task to demonstrate how landmarks can be deduced. The truck must *load* the package at B and later unload it at C. Thus, the truck must eventually also be at these locations.

Definition 4 (Disjunctive action landmark)**.** *Let $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A} \rangle$ be a planning task and let s be a state of $\mathcal{T}$.*
A disjunctive action landmark *of s is a non-empty set of actions $\ell \subseteq \mathcal{A}$ such that every s-plan contains an action $a \in \ell$.*

It can be confusing to use the term landmark for all of the above ideas. Hence, we provide a formal definition only for disjunctive action landmarks. They are particularly suited for large parts of this thesis. Whenever possible, we use disjunctive action landmarks in provided examples or to explain concepts. Therefore, the term landmark refers to disjunctive action landmarks unless explicitly stated otherwise. This is not much of a limitation as it is possible to transform fact landmarks into disjunctive action landmarks and vice versa. A fact landmark for fact $v \mapsto d$ induces a disjunctive action landmark $\{a \in \mathcal{A} \mid v \mapsto d \in \mathit{eff}(a)\}$. An action landmark is simply a disjunctive action landmark with only one alternative. Landmarks of a planning task correspond to the initial state s_0 unless specified otherwise.
However, we do not deem it sensible to completely refraining from using fact landmarks. First of all, most literature considered uses fact landmarks in one way or another. Their ideas are based on facts (e.g., various landmark generators in Section 3.4). The meaning behind these ideas is not as intuitive when translated into the context of disjunctive action landmarks. Furthermore, we based our implementations on the Fast Downward planning system (Helmert 2006). It has been equipped with landmark fundamentals since its early days. These implementations are mainly based on fact landmarks, so we implemented our extensions accordingly. At some point, we need to transition to disjunctive action landmarks, which we do exactly as explained above: use the set of actions that produce a fact as the disjunctive action landmark representing that fact landmark.

3.2 Landmark Orderings

It is intuitive that in logistics problems packages must be loaded before they are unloaded. This idea can be formalized by introducing a *landmark ordering* between the two corresponding landmarks. In the example from Figure 3.1, such an ordering would be $\{\mathit{load}(p_{B \to C}, t_1, B)\} \to \{\mathit{unload}(p_{B \to C}, t_1, C)\}$. It states that before unloading the package at C, it must be picked up at B.
In the literature, several *types* of orderings are distinguished. Richter (2010) provides a detailed overview, which we will only briefly represent here. In the following we use fact

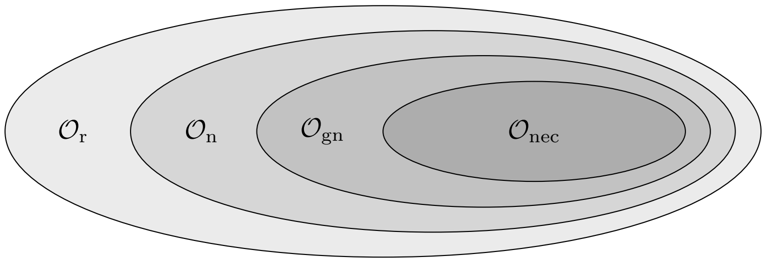

$\mathcal{O}_{\mathrm{r}}$
$\mathcal{O}_{\mathrm{n}}$
$\mathcal{O}_{\mathrm{gn}}$
$\mathcal{O}_{\mathrm{nec}}$

Figure 3.2: Relationship between the types of landmark orderings used in this thesis. Reasonable orderings $\mathcal{O}_{\mathrm{r}}$ are the most general, then natural orderings $\mathcal{O}_{\mathrm{n}}$ which must hold in all plans. Greedy-necessary orderings $\mathcal{O}_{\mathrm{gn}}$ are a special case of $\mathcal{O}_{\mathrm{n}}$ and, in turn, necessary orderings $\mathcal{O}_{\mathrm{nec}}$ are more restrictive than $\mathcal{O}_{\mathrm{gn}}$.

landmarks because landmark orderings were originally defined using fact landmarks (Hoffmann et al. 2004). Understanding the underlying meaning of these ordering constraints should also be easier in the context of facts. However, if an ordering exists between two fact landmarks, the ordering also holds for the corresponding disjunctive action landmarks.

In the following, ℓ and ℓ' are two landmarks for a state s in a planning task. There is a *natural ordering* $\ell \rightarrow_{\mathrm{n}} \ell'$ between the two if in every s-plan ℓ must become true before ℓ'. Special cases of this ordering relation are *necessary orderings* $\ell \rightarrow_{\mathrm{nec}} \ell'$ and *greedy-necessary orderings* $\ell \rightarrow_{\mathrm{gn}} \ell'$. The ordering $\ell \rightarrow_{\mathrm{nec}} \ell'$ denotes that ℓ must always hold right before ℓ' becomes true.[3] If this only holds when ℓ' is first added, there is an ordering $\ell \rightarrow_{\mathrm{gn}} \ell'$.

All natural orderings are mandatory in the sense that they must hold along every plan (Hoffmann et al. 2004). Consequently, they cannot be cyclic in solvable tasks, because $\ell \rightarrow_{\mathrm{n}} \ell'$ and $\ell' \rightarrow_{\mathrm{n}} \ell$ are contradictory. There is a class of non-mandatory orderings called *reasonable orderings*.

Definition 5 (Reasonable orderings between landmarks)**.** *Let ℓ and ℓ' be landmarks for a state s in a planning task $\mathcal{T}$. There is a* reasonable ordering *between ℓ and ℓ', written $\ell \rightarrow_{\mathrm{r}} \ell'$, if the following holds: for every s-plan $\pi = \langle a_1, \ldots, a_n \rangle$ where $\ell' \in s[\![\langle a_1, \ldots, a_i \rangle]\!]$, $\ell \in s[\![\langle a_1, \ldots, a_j \rangle]\!]$ with $i < j \leq n$, and $\ell \notin s[\![\langle a_1, \ldots, a_k \rangle]\!]$ for all $1 \leq k < j$; there exists an $l \in \{i+1, \ldots, j\}$ such that $\ell' \notin s[\![\langle a_1, \ldots, a_l \rangle]\!]$ and an $m \in \{j, \ldots, n\}$ such that $\ell' \in s[\![\langle a_1, \ldots, a_m \rangle]\!]$.*

Reasonable orderings denote that in every plan where ℓ' is added before ℓ is added for the first time, it is required to destroy the fact of ℓ' in order to achieve ℓ. Thus, it is reasonable to achieve ℓ before or at the same time as ℓ'.

Another class of non-mandatory orderings called *obedient-reasonable orderings* was introduced by Hoffmann et al. (2004). They represent additional ordering constraints $\ell \rightarrow_{\mathrm{r}}^{\mathcal{O}} \ell'$ which only hold when obeying a set of other ordering constraints $\mathcal{O}$ (e.g., the set of reasonable orderings). These orderings are only valid if the set of orderings $\mathcal{O}$ is considered mandatory during the planning procedure. By definition, reasonable orderings do not have this property when used for cost-optimal planning. Hence, obedient-reasonable orderings are unsuited for our research.

[3] It is a necessary precondition for achieving ℓ'.

The presented landmark ordering types can be hierarchically ordered by strength as displayed in Figure 3.2. Necessary orderings are the most restrictive, followed by greedy necessary orderings. They are both special cases of natural orderings which are in turn stronger than reasonable orderings. The latter do not necessarily hold in all s-plans (i.e., it is possible to achieve ℓ' before ℓ despite the ordering $\ell \rightarrow_{\mathrm{r}} \ell'$). However, an ordering $\ell \rightarrow_{\mathrm{r}} \ell'$ implies the ordering restriction of natural orderings in the sense that ℓ' must be made true after (or at the same time as) ℓ is made true for the first time.

3.3 Landmark Graphs

Landmarks and landmark orderings can be represented in one data structure using graphs.

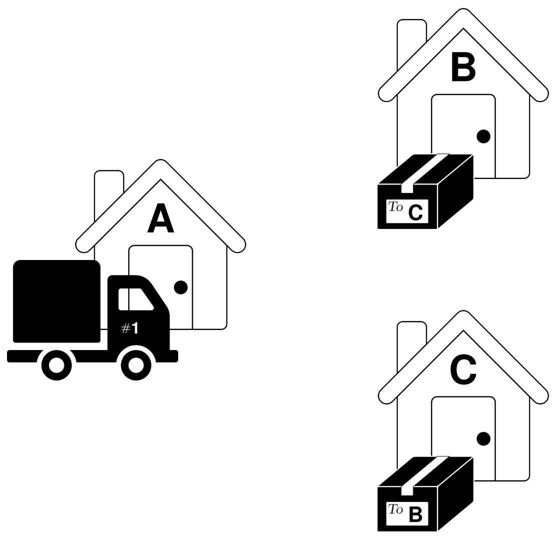
Definition 6 (Landmarks graph)**.** *Let s be a state of planning task $\mathcal{T}$.*
A landmark graph *$\mathcal{G} = \langle V, E\rangle$ is a directed graph with a set of* vertices *V and a set of* edges *E. There is a vertex in V for every landmark for s. The graph has an edge $\langle \ell, \ell'\rangle$ with label t between two vertices ℓ and ℓ' if*

- *there exists a landmark ordering $\ell \rightarrow_t \ell'$ with ordering type t, and*
- *there is no landmark ordering $\ell \rightarrow_{t'} \ell'$ stronger than t.*

In this thesis we are interested in cyclical dependencies between landmarks. They correspond to *deadlocks* in the planning task. A deadlock denotes that the current state requires taking an action that does not directly contribute to the goal (Paul and Helmert 2016). A deadlock in a state s can be *resolved* by applying an action a such that the deadlock is not present in $s[\![a]\!]$. Multiple deadlocks may be resolved simultaneously by a single action (Gupta and Nau 1992).

The example provided in Figure 3.3a entails such a deadlock in the initial state. The package $p_{B\rightarrow C}$ needs to be delivered from B to C while $p_{C\rightarrow B}$ must go from C to B. In order to load and unload the packages, t_1 must be at both locations B and C once for each transported package. Since it is at neither of these locations in the beginning, it must eventually move there, making $\ell_B = \{move(t_1, A, B), move(t_1, C, B)\}$ and $\ell_C = \{move(t_1, A, C), move(t_1, B, C)\}$ disjunctive action landmarks. It is reasonable to drive to C after B because otherwise we cannot deliver the package $p_{B\rightarrow C}$ when arriving at C. Driving to B after C in order to deliver $p_{C\rightarrow B}$ when first arriving at B is equally reasonable. Looking at both packages separately, landmarks ℓ_B and ℓ_C constitute necessary steps to achieve the goal of delivering this package. In the initial state, the only applicable actions are $move(t_1, A, B)$ and $move(t_1, A, C)$. They can both be used to achieve one of the landmarks. However, neither of them adds to the overall goal; the so achieved landmark is required again to deliver the second package.

Recognizing deadlocks provides information about a state in a planning task that can be used in a heuristic; the cost for resolving deadlocks may be accounted to the heuristic estimate of that state. The equivalent of a deadlock in terms of landmarks is a cycle in the landmark graph; it is induced by the landmark orderings which are directed edges between



(a) Planning task

$$\begin{tikzcd}
\ell_B = \left\{ \begin{matrix} move(t_1, A, B) \\ move(t_1, C, B) \end{matrix} \right\} \arrow[d, bend left, "r"] \\
\ell_C = \left\{ \begin{matrix} move(t_1, A, C) \\ move(t_1, B, C) \end{matrix} \right\} \arrow[u, bend left, "r"]
\end{tikzcd}$$

(b) Landmark graph

Figure 3.3: Logistics planning task with a deadlock. For each package individually, moving to B as well as moving to C are landmarks. Since it is necessary to load packages before they can be unloaded at their destination, one of the locations must be driven to twice. A cycle in the landmark graph represents this deadlock.

the landmark nodes. In order to resolve a deadlock, one landmark in such a cycle must be achieved twice along all s-plans.

The packages $p_{B\to C}$ and $p_{C\to B}$ in the planning task from Figure 3.3a constitute reasonable orderings $\ell_B \to_{\mathrm{r}} \ell_C$ and $\ell_C \to_{\mathrm{r}} \ell_B$; they must be picked up at one location before they can be unloaded at the other. The resulting landmark graph is presented in Figure 3.3b. All further landmarks which could be found for the initial state of this example are ignored.

For the cycle-covering heuristic we are interested in finding as many distinct deadlocks as possible. Therefore, we search for all *elementary cycles* in a landmark graph.

Definition 7 (Elementary cycle)**.** *Let $\mathcal{G}$ be a landmark graph.*
A path *in $\mathcal{G}$ is a sequence of vertices $\pi = \langle v_1, \ldots, v_n \rangle$ such that $\langle v_i, v_{i+1} \rangle \in E$ for $1 \leq i < n$.*
A path π is a cycle *iff $v_1 = v_n$ and we write $\pi : v_1 \to \cdots \to v_{n-1} \to v_1$.*
A path is elementary *if no vertex appears twice. A cycle is elementary if no vertex but the first and last appears twice.*
Two elementary cycles are distinct *if one is not a cyclic permutation of the other (e.g., $v \to w \to v$ is a cyclic permutation of $w \to v \to w$ which makes them the same cycle).*

By $\mathcal{L}(c)$ we refer to the set of landmarks that occur in a cycle c. Similarly, $\mathcal{O}(c)$ refers to the orderings which produce that cycle.

Our Definitions 6 and 7 are inspired by the definition of a graph in Johnson (1975). He suggests an algorithm to find all elementary cycles in a graph $\mathcal{G} = \langle V, E \rangle$ efficiently. The procedure returns a deterministic list of elementary cycles C. Its running time is bounded by $O\big((|V| + |E|)(|C| + 1)\big)$ and it has a space-bound of $O\big(|V| + |E| \bigcap\big)$.

Johnson's Algorithm[4] explores all vertices in lexicographical order. All paths starting from the current vertex are explored one after the other. All vertices along one path are marked as blocked. Paths are always elementary because blocked vertices are not considered (except for the starting vertex). As soon as a path returns at the starting vertex, it is added to the list of elementary cycles. In this case, or if a dead end is reached (i.e., no unblocked vertices are available), the procedure goes back along the path, unblocking all vertices until an unexplored path is available. Only distinct cycles are found because all vertices ordered before the starting vertex are ignored during exploration.

3.4 Landmark Generators

Above, we have established the definitions of landmarks and landmark orderings. The problem of finding all landmarks for a planning task is PSPACE-complete (Hoffmann et al. 2004). In this section we present methods that approximate the landmarks along with landmark orderings. This thesis is not concerned with analyzing or improving upon these methods. We rather depend on their generated landmarks for further use in computing a heuristic.

3.4.1 HPS Landmarks

In their article called "Ordered Landmarks in Planning", Hoffmann et al. (2004) describe a backchaining approach to generate ordered landmarks. Their procedure only detects fact landmarks, thus by landmarks we refer to fact landmarks in this section. They construct a landmark graph by iteratively adding new landmark candidates along with greedy-necessary orderings. By showing that the relaxed planning task is unsolvable when a landmark candidate is removed, the candidate is provably a landmark. If the relaxed planning task with the landmark removed is solvable, it might not be a landmark and thus is removed from the landmark graph.

Greedy-necessary orderings are the only natural orderings that their algorithm finds. These are concerned with a landmark being first added. Assuming landmarks are only required once, the greediness lies in the early achievement of landmarks. The authors suggest a similar ordering criterion concerned with adding a fact for the last time. However, they leave this for future work since they claim it to be rather unintuitive.

Based on the landmark graph from above, they approximate reasonable and obedient-reasonable orderings using an *aftermath* relation. If a landmark ℓ' is in the aftermath of another landmark ℓ, this is a sufficient condition for a reasonable order $\ell \rightarrow_{\mathrm{r}} \ell'$. Obedient-reasonable orderings are in turn generated by the landmark graph extended with reasonable orderings and the *obedient aftermath* condition. Their name stems from the underlying assumption that any plan obeys the reasonable orderings, which is not necessarily true.

The algorithm of Hoffmann et al. breaks cycles occurring in the final landmark graph until

[4] Johnson also proposed an algorithm to find all shortest paths between vertices in a graph. In this thesis, however, by "Johnson's Algorithm" we refer to the described algorithm for finding all elementary cycles in a directed graph.

it is acyclic. This is necessary since their search procedure cannot handle cycles. They use the remaining landmarks and orderings for finding suboptimal solutions for planning tasks. Their planner decomposes the original planning task into smaller sub-tasks, which are easier to solve. Landmarks are achieved one after the other and removed from the landmark graph after completion. Destructive interactions between landmarks (e.g. due to conflicting landmarks, which can never be true at the same time) are ignored in their approach. This may lead to a large decrease of solution quality, if a solution is found at all. However, their experiments show significant speedup for many problems compared to previous attempts such as Koehler and Hoffmann's (2000) forced goal orderings.
The article by Hoffmann et al. (2004) provides us with a strong fundamental knowledge of landmarks and landmark orderings. However, since their algorithm may produce unsound landmark orderings makes this method unsuited for cost-optimal planning. We do not consider their method for generating landmarks in this thesis, but use alternative methods based on the same ideas.

3.4.2 RHW Landmarks

Richter (2010) writes about generating landmarks and landmark orderings as part of her dissertation. The algorithm was originally proposed in Richter, Helmert, and Westphal (2008). It also begins with a backchaining procedure but uses a *possibly-before* criterion to ensure only sound orderings are found. Additionally, they consider disjunctive landmarks instead of only fact landmarks, which yields a larger set of landmarks.
Their method generates even more landmarks by using *domain transition graphs* which are atomic projections on the variables (Helmert 2004). A domain transition graph captures how the value of a variable can change given the set of actions. If all paths leading to an already found landmark pass through a common node, the corresponding fact is recognized as a new fact landmark. In order to keep the number of landmarks manageable, their algorithm prevents landmarks from overlapping.
The orderings their method finds are the same as those found by Hoffmann et al.'s (2004) algorithm, although adapted to their found landmarks. They enhance this set of orderings by considering the relaxed planning graph. A landmark that is not possible before some other landmarks is naturally ordered after them. Reasonable and obedient-reasonable orderings are gathered similarly to Hoffmann et al. (2004).

3.4.3 ZG Landmarks

An approach that finds significantly more landmarks than the aforementioned methods was proposed by Zhu and Givan (2003). They were the first to not only consider facts which must become true in every plan as landmarks; their method also discovers actions which must be part of every plan. The method builds the relaxed planning graph and propagates labels referring to actions and facts along paths in that graph. It starts at the first layer where all nodes are labeled solely as themselves. Afterward, a fact node is labeled with the intersection of the labels on its predecessor nodes. In contrast, action nodes are labeled with the union of the labels on their parent nodes. In the final layer, all labels on goal facts

are landmarks for the according planning task. Zhu and Givan claim that their procedure provides a complete list of causal landmarks.
The authors directly suggest an improved version of their algorithm that counts how often an action needs to be applied and how often variables change their value. It requires a slightly different label propagation algorithm which updates the counters at each layer of the relaxed planning graph. A heuristic based on these counters combined with bin-packing[5] is suggested. Orderings between landmarks are only mentioned as a topic of future work in the paper of Zhu and Givan. They could be inferred by any fact or action node in the last layer, because its labels describe the landmarks for this fact or action, respectively.

3.4.4 h^m Landmarks

The last method we consider in this section is from Keyder, Richter, and Helmert (2010) and generates landmarks based on the m-relaxation of a planning task. With this approach it is possible to include delete-effects in the generated landmarks, that are not considered in all previous methods. First, it computes the $\mathcal{T}^m$ problem which is a transformed version of the original planning task $\mathcal{T}$. Every fact of $\mathcal{T}^m$ denotes a set of at most m facts from $\mathcal{T}$. These encode conjunctions of facts which must hold all at once for some action to be taken in the original task. Facts and actions of $\mathcal{T}^m$ encode delete-effects although none are present in its actions. Using the technique of Zhu and Givan (2003), Keyder et al. get conjunctive landmarks for the original problem. These can either be used as such, or be split because every part of a conjunction must be made true for the conjunction to be true. In terms of landmarks this means that every component of a conjunctive landmark is a landmark on its own.
With this approach, it is not only possible to find all causal landmarks for the delete relaxation of a problem, but all causal landmarks for the original planning task. This requires building the $\mathcal{T}^m$ for a sufficiently large m which can be very inefficient. The empirical evaluations of Keyder et al. consider only $m = 1$ (which is identical to Zhu and Givan) and $m = 2$. They claim to find more landmarks than the RHW method in all tested domains for all m. The number of expanded states decreases the most when conjunctive landmarks are used. However, treating all facts of each conjunction as individual landmarks enables solving more problems than using the conjunctions themselves as landmarks.

3.5 Landmark Heuristics

Heuristics based on landmarks have been studied since landmarks were first introduced by Porteous et al. (2001). They were a popular subject in their early days and have made strong appearances in planning competitions (e.g., the LAMA planner by Richter and Westphal (2010), two-time winner of the IPC Sequential Satisficing Track 2008 and 2011). However, contributions to the topic seem to have diminished over the past few years. In this chapter, we revisit some of the heuristics proposed in literature over the last two decades.

[5] This is similar to the hitting set approach as described in Chapter 4.

3.5.1 Landmark-Count Heuristic

The work of Richter (2010) does not only introduce the landmark generator described above; it also suggests a path-dependent landmark heuristic that uses these generated landmarks and landmark orderings. The landmarks are generated once for the initial state. Landmarks for all other states are approximated by the following rule:

$$\mathcal{L}(s,\pi) = \mathcal{L} \setminus (\mathit{Accepted}(s,\pi) \setminus \mathit{ReqAgain}(s,\pi)) \tag{3.1}$$

where $\mathit{Accepted}(s,\pi) \subseteq \mathcal{L}$ are the landmarks that are true at some point along π and $\mathit{ReqAgain}(s,\pi) \subseteq \mathit{Accepted}(s,\pi)$ is a set of landmarks for which it can be induced that they are required again. A landmark is required again if it is part of the goal or greedy-necessarily ordered before a landmark that has not been accepted yet. The cost of an s-plan is estimated by the total number $|\mathcal{L}(s,\pi)|$ of landmarks which are not accepted in s or required again. This heuristic is inadmissible because multiple landmarks can be achieved at once. As the heuristic is used for sub-optimal planning, this is no problem.
In contrast to Hoffmann et al. (2004), Richter and Westphal (2010) do not force orderings upon the planner. Rather, landmark orderings are used to guide the search by trying out paths obeying them more frequently. To enforce this procedure, they implemented *anytime search* together with *multi-queue heuristic search*. The results of this implementation were astonishing. Richter and Westphal surprised themselves by outperforming all other contestants at the IPC 2008 with the LAMA planner. In their experimental setup, they compared different enhancements such as landmarks and cost-sensitive heuristics to previous state of the art planning systems and showed improvement of solution quality by a large margin.
In this thesis, we are interested in cost-optimal planning. Thus, we ignore all search enhancements except the generation of landmarks and landmark orderings. At the time of writing, Richter et al.'s method outmatched the other landmark ordering generators significantly. This makes their method particularly interesting for this thesis, as the number of cycles is likely to correlate with the number of found orderings.

3.5.2 Admissible Landmark Heuristic

Karpas and Domshlak (2009) have used landmarks to derive an admissible heuristic for cost-optimal planning. One of their contributions provides a way to make the landmark-count heuristic admissible; instead of counting the landmarks which are yet to be achieved, they sum up the costs required to achieve all these landmarks. The cost of a landmark is the cheapest option to achieve this landmark based on an action cost partitioning; the cost of each action is partitioned among all landmarks it achieves. This adaption ensures that the cost of an action is counted at most once, even if multiple landmarks exist which can be achieved by this action.
The choice of the cost partitioning is unrestricted. However, Karpas and Domshlak show that uniform cost partitioning is sub-optimal. Moreover, an optimal cost partitioning scheme is provided which can be computed in polynomial time using an LP. Besides being optimal, this cost partitioning outmatches uniform cost-sharing due to monotonically increasing costs for growing landmark sets; they show that the heuristic value may drop when considering more landmarks if uniform cost partitioning is used.

In a further step, they suggest using action landmarks to enhance their heuristic. If an action a which is a landmark has not been applied to reach a state s, its full cost can be accounted for the heuristic. All landmarks that can be achieved by a are then removed from the remaining landmarks before applying the cost partitioning scheme described above. They call this approach *action landmark covering* and the according heuristic dominates their previous approach. If solving LPs is too time-consuming, it can also be combined with uniform cost partitioning which renders it computationally efficient. The contribution of using action landmark covering is shown to be substantial by empirical results.
The landmarks for a planning task are only computed once before starting the search for a plan. Thus, the landmark set of a state s is based on the path(s) to s similar to Richter et al. (2008). A modified version of A* is introduced in order to handle the additional information which results from finding multiple paths to s. LM-A* computes the heuristic value of a state by approximating the remaining landmarks based on all paths to s. Whenever a new path to s is found, the heuristic is therefore reevaluated. An increased h-value is a more accurate estimate of the true cost of the optimal s-plan.

3.5.3 LM-Cut Heuristic

One of the most popular landmark heuristics is the LM-cut heuristic $h^{\text{LM-cut}}$ which was introduced by Helmert and Domshlak (2009). It is a highly accurate estimate of the optimal delete relaxation heuristic h^+. Unlike other methods, $h^{\text{LM-cut}}$ produces disjunctive action landmarks during its heuristic computation. The procedure repeatedly searches for cuts in justification graphs for a state in a planning task. The set of actions in each cut is a disjunctive action landmark. Action costs are eroded while accumulating the cheapest cost to resolve the landmarks. The sum of these costs then constitutes the heuristic estimate for a state.
The LM-cut heuristic can be interpreted as a cost-partitioning based on the cut landmarks (Bonet and Helmert 2010). Every iteration provides one component to the cost partitioning. The cost function of a component assigns $c = \min_{a \in L} cost(a)$ to all actions in a cut L and 0 to all other actions. Afterward, the original cost function is adapted by reducing the action costs of all actions in the cut by c. Simultaneously, the heuristic value of the state that is currently evaluated is increased by c. The evaluation for that state stops as soon as no more cuts of cost $c > 0$ can be found.
LM-cut provides a very accurate estimate of h^+ and is a strong heuristic for cost-optimal planning. Since h^+ ignores delete effects, all of its approximations have in common that they apply each operator at most once. Thus, the heuristic is prone to behave poorly where operators are necessary more than once.

4 Minimum Hitting Sets and Linear Programming

In the previous chapter we have presented methods that approximate the sets of landmarks in a planning task. Most of the discussed literature also provides a way to use the found landmarks for planning, although these problems can be approached independently. In this thesis we examine an unexplored way of dealing with landmarks. We present the concepts for our approach in this chapter.
Given a set of disjunctive action landmarks, we can ask the question "What is the cheapest way to satisfy all landmarks?" Landmarks could overlap, thus the answer to this question is not necessarily to pick the cheapest action from every landmark and combine them to a plan. Instead, we search for the cheapest set of actions so that every landmark contains at least one action from that set. This is called a *minimum hitting set* (MHS) problem[6] (Karp 1972).

4.1 Minimum Hitting Set

The following is a formal definition of hitting sets.

Definition 8 (Hitting set)**.** *Let X be a set, $\mathcal{F} = \{F_1, \dots, F_n\} \subseteq 2^X$ be a family of subsets of X, and $c : X \to \mathbb{R}_0^+$ be a cost function for X.*
A hitting set *is a subset $H \subseteq X$ that "hits" all subsets in $\mathcal{F}$, i.e., $H \cap F \neq \emptyset$ for all $F \in \mathcal{F}$.*
The cost *of H is $\sum_{x \in H} c(x)$.*
A minimum hitting set *(MHS) is a hitting set with minimal cost.*

Using hitting sets is the most accurate approach to define a heuristic when considering solely disjunctive action landmarks as a source of information. However, computing the MHS is NP-hard and thereby not feasible for complex problems (Karp 1972). By relaxing the problem as discussed in Section 4.2 we can still use the fundamental idea of MHS for our needs.
In a planning task $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A} \rangle$ where $G \not\subseteq s_0$ (i.e., the initial state is not a goal state), the set of all actions is trivially a landmark; at least one action needs to be applied to leave

[6] Also known as the *set cover problem.*

the non-goal initial state. This landmark might not be very relevant for most planning tasks. This is because it provides information that is also contained in landmarks of smaller cardinality. Any landmark that does not contain all actions further restricts which of the actions need to be considered for reaching a goal state from the initial state s_0.
Generally speaking, an element $F \in \mathcal{F}$ in a hitting set problem can be removed without changing the solution if another element $F' \subseteq F$ exists in $\mathcal{F}$. In terms of landmarks ℓ and ℓ', if $\ell' \subseteq \ell$, then every solution that hits ℓ' inevitably also hits ℓ; applying an action in ℓ' means applying an action in ℓ (but not vice versa).

4.2 Integer and Linear Programs

MHS problems can be solved using integer programming. An *integer program* (IP) consists of a linear objective function that needs to be maximized or minimized over a set of variables. These variables are constrained by a linear system of inequalities and their value must be integral in the solution. In this section we explain how the IP for an MHS is constructed.
Consider an MHS with X, $\mathcal{F}$, and c as in Definition 8. The objective is to find the subset of X that hits all $F \in \mathcal{F}$ and has minimal cost.

$$\min \sum_{x \in X} \mathsf{Y}_x c(x) \quad \text{s.t.} \tag{4.1}$$

$$\begin{aligned} \mathsf{Y}_x &\in \{0,1\} && \text{for all } x \in X \text{ and} \\ \sum_{x \in F} \mathsf{Y}_x &\geq 1 && \text{for all } F \in \mathcal{F} \end{aligned} \tag{4.2}$$

Solving this IP gives a solution for the MHS problem. The variables Y_x in Equations (4.1) and (4.2) denote whether x is in the hitting set H or not: $H = \{x \in X \mid \mathsf{Y}_x = 1\}$.
The variables Y_x are restricted to be either 1 or 0 in order to model the hitting set; an item x can either be in the hitting set or not. Same as the MHS problem, Karp (1972) has shown that the 0-1 integer problem is NP-complete. In order to approximate the solution, one can define a corresponding *linear program* (LP). LPs are computable in polynomial time (Khachiyan 1979; Aspvall and Stone 1980) since they relax the restrictions by allowing variables to have any real-numbered value greater or equal to 0 in a solution. The LP-relaxation of the above MHS has the following form.

$$\min \sum_{x \in X} \mathsf{Y}_x c(x) \quad \text{s.t.} \tag{4.3}$$

$$\begin{aligned} 0 \leq \mathsf{Y}_x &\leq 1 && \text{for all } x \in X \text{ and} \\ \sum_{x \in F} \mathsf{Y}_x &\geq 1 && \text{for all } F \in \mathcal{F} \end{aligned} \tag{4.4}$$

This adaption enables finding a solution to the minimization problem in polynomial time. However, it comes at the cost of possibly having an item n times in the solution where $0 < n < 1$. This does not make much sense given the original definition of the MHS problem. Still, the LP solution is an adequate approximation most of the time.
Now consider a minimization IP and its corresponding LP-relaxation with the same set of constraints. Every variable assignment that is a solution of the IP does inevitably adhere to

these constraints. This remains true if the same values are used in the LP setting, thus it is possibly also a solution for the LP. However, as the variables are not restricted to integers in the LP, a cheaper solution may exist. Hence, the LP solution never overestimates the IP solution. In terms of a planning heuristic, if it is based on an IP, the according LP solution is an admissible approximation. Moreover, the IP heuristic dominates the LP-relaxed version of the same heuristic.

4.3 Operator Counting

The operator-counting framework is a family of heuristics based on LPs which was introduced by Pommerening, Röger, Helmert, and Bonet (2014). As the name suggests, it counts how often each action is required to get from a state to the goal.
All operator-counting LPs share the following objective function for a planning task with action set $\mathcal{A}$.

$$\min \sum_{a \in \mathcal{A}} \mathsf{Y}_a \cdot \mathit{cost}(a) \tag{4.5}$$

A new LP is considered in every state that is encountered during the search of a planning task $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A} \rangle$. Let C be a function that maps the states of $\mathcal{T}$ to a set of operator-counting constraints. The LP variables include one variable Y_a for each action $a \in \mathcal{A}$. All constraints in $C(s)$ must be feasible in the following sense; if Y_a^π is the number of occurrences of a in a plan π for $\mathcal{T}$, then all constraints must hold for all plans $\pi \in \Pi$ if Y_a is replaced with Y_a^π for all $a \in \mathcal{A}$. Then, the objective value of Equation (4.5) subject to $C(s)$ is an admissible approximation of the optimal s-plan.
There is a variety of possible operator-counting constraints that capture different aspects of a state (Pommerening et al. 2014). One valuable property of the operator-counting framework is that different ideas can be combined into one strong heuristic. Take a set $\mathcal{H} = \{h^1, \dots, h^n\}$ of operator-counting heuristics and let $C_i(s)$ be the set of LP constraints for $h^i(s)$. We can combine all heuristics in $\mathcal{H}$ in one heuristic $h^\mathcal{H}$ by joining together all LP constraints in every state. The constraints for $h^\mathcal{H}(s)$ would be $\bigcup_{i=1}^n C_i(s)$. The resulting heuristic $h^\mathcal{H}$ dominates the maximum of all component heuristics (i.e., $h^\mathcal{H}(s) \geq \max_{h \in \mathcal{H}} h(s)$ for all s) (Pommerening et al. 2014).
For the sake of completeness we present another important finding by Pommerening et al. It is used in a proof later in this thesis.

Proposition 1 (Dominance)**.** *Let C, C' be functions that map states s of a planning task $\mathcal{T}$ to constraint sets for s such that $C(s) \subseteq C'(s)$ for all states s. Then the IP/LP heuristic for C' dominates the respective heuristic for C.*

4.4 Landmark Heuristic Based on Minimum Hitting Sets

One way of defining constraints for the operator counting framework uses the aforementioned MHS constraints for the LP. By definition, each landmark must occur at least once in every plan. In other words, if $\mathcal{A}_\pi \subseteq \mathcal{A}$ is the set of actions used in an s_0-plan π, then $\mathcal{A}_\pi \cap \ell$ must be non-empty for all landmarks $\ell \in \mathcal{L}(s_0)$. This is basically the definition of the hitting set problem. A solution of the MHS over all disjunctive action landmarks is a lower bound on the true cost of an optimal plan. Thus, we introduce a simple constraint for every landmark ℓ of the state s.

$$\sum_{a \in \ell} \mathsf{Y}_a \geq 1 \tag{4.6}$$

It denotes that at least one of the actions contained in ℓ must be used by each plan. (Since we use the LP-relaxation, a valid solution may for example indicate to apply two actions 0.5 times.) We define $h^{\mathrm{LM}}(s)$ to be the solution of an LP with the constraints from Equation (4.6) for all landmarks of state s. The resulting heuristic h^{LM} is a landmark heuristic that uses the MHS solution as an estimate for the optimal s-plan in a planning task.

All such LPs find solutions where $0 \leq \mathsf{Y}_a \leq 1$ for all $a \in \mathcal{A}$. On the one hand, setting the count of an operator to 1 already satisfies all constraints in which that operator occurs. On the other hand, concerning cost minimization, it is never beneficial to count the same operator more than once. A heuristic based on the MHS solution over a set of landmarks is thus bounded by the optimal delete relaxation heuristic h^{+} (Bonet and Castillo 2011). However, there are planning tasks that require applying the same operator multiple times to reach the goal. Cycles in the landmark graph may hint to this necessity by requiring that one landmark in the cycle must be achieved at least twice.

5 Cycle-Covering Heuristic

In this chapter we present the main contribution of this thesis. We explain how a generalized version of the cycle-covering heuristic based on the ideas of Paul et al. (2017) could be implemented. In the first part we present two alternatives for deciding which landmarks need to be considered in a state of a planning task. The second part contains explanations of how the heuristic is computed in each state.

5.1 Active Landmarks and Cycles

We introduce the term *active landmarks* referring to a set $\mathcal{L}(s)$ of landmarks for a state s. It serves two purposes which make the notation in the upcoming sections easier. On the one hand, given a set $\mathcal{L}$ of landmarks for state s, the set of active landmarks $\mathcal{L}(s) \subseteq \mathcal{L}$ denotes those landmarks which must hold along all s-plans, but are not true in s. This is only relevant when considering fact landmarks where all facts in s are trivial landmarks. One can use $\mathcal{L}(s) = \mathcal{L}$ if only disjunctive action landmarks are considered.

On the other hand, since the set of landmarks changes during the act of planning, the set of active landmarks changes with each state. Consider for example a logistics problem as shown in Figure 5.1. In the initial state s_0, $\ell = \{\mathit{load}(p_{B\to A}, t_1, B)\}$ is a landmark because the package must inevitably be loaded in order to bring it to its destination. Now consider a state s where the fact $p_{B\to A} \mapsto t_1$ holds. Obviously, ℓ is not a landmark for s because not every s-plan unloads and reloads the package at B. The landmark-count heuristic (Richter et al. 2008) uses the set of landmarks from the initial state removing landmarks which have already been true at some point on the path to s. While this method does not explicitly store active landmarks, we introduce an algorithm which does in Section 5.1.2.

Landmark orderings denote dependencies between the involved landmarks. In the landmark graph, such orderings might lead to cycles which denote that the involved landmarks are cyclically dependent on one another. If one of the landmarks in the cycle is not active, the cyclical dependency is not valid. A cycle c is active in a state s if all its landmarks are active in s: $\mathcal{L}(c) \subseteq \mathcal{L}(s)$. The set of active cycles is $\mathcal{C}(s)$. In this section we explain two techniques for deciding which landmarks and cycles are active in a state of a planning task. Both methods use a landmarks generation method according to Section 3.4.

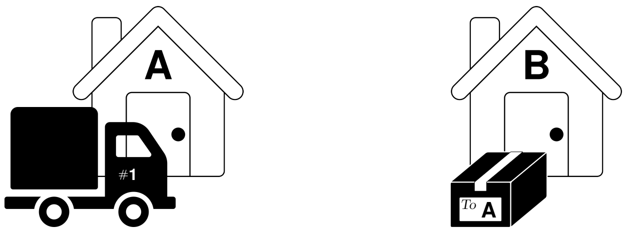


Figure 5.1: Small logistics task where $load(p_{B\to A}, t_1, B)$ is an action landmark for all states where $p_{B\to A}$ is located at B. It is not a landmark in all states where this is not the case.

5.1.1 Recomputing the Landmarks Graph

Our first approach is to compute the landmarks graph in every state s that is encountered during search. The active landmarks $\mathcal{L}(s)$ are recomputed from scratch repeatedly. However, we need to adapt the original planning task $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A} \rangle$ in order to get the landmarks for s because the landmark generators always compute the landmarks for s_0. The landmarks of any state $s \in S$ of $\mathcal{T}$ can be generated by computing the landmark graph for $\mathcal{T}_s = \langle \mathcal{V}, s, \mathcal{A}, G \rangle$. The only thing that differs from the original planning task is the initial state. If $\mathcal{G}_s = \langle \mathcal{L}_s, \mathcal{O}_s \rangle$ is the landmark graph computed with the chosen landmark generator, then $\mathcal{L}(s) = \mathcal{L}_s$ when using disjunctive action landmarks. Otherwise, if fact landmarks are used, then only those landmarks which do not hold in s are active: $\mathcal{L}(s) = \{\ell \in \mathcal{L}_s \mid \ell \setminus s \neq \emptyset\}$. If $\mathcal{C}_s$ is the set of elementary cycles induced by $\mathcal{O}_s$, then $\mathcal{C}(s) = \{c \in \mathcal{C}_s \mid \mathcal{L}(c) \subseteq \mathcal{L}(s)\}$.
Using this method, the time required to compute the heuristic value of a state depends on how long it takes to generate the according landmark graph. As the computation of landmarks might be expensive, we expect performance issues with this approach. On top of this, all computations based on the graph (e.g., finding cycles) must also be repeated. We present an alternative to surpass computing landmark graphs over and over again in the following section.

5.1.2 Tracking Unresolved Landmarks and Landmark Orderings

This approach computes the landmark graph once for the initial state at the beginning of the search procedure. Until termination of the planner it works with this graph and keeps track of which landmarks and landmark orderings have not been resolved since the beginning. In the following we explain how to keep track of active landmarks and cycles. Note that the provided algorithms are based on fact landmarks. The decision to refrain from using disjunctive action landmarks here was driven by the current implementation of the Fast Downward planner. It produces landmark graphs with fact landmarks together with a list of possible achievers. In our implementation we keep them as fact landmarks until after deciding whether they are active or not.
We track the active landmarks and cycles by maintaining two sets of information for each state s. The first is the set of active landmarks itself and it captures which landmarks have not been achieved in all found paths to s. The second is a list of landmark orderings which still need to be considered in s. If a cycle consists only of orderings that are not resolved, this means the cyclical dependency is still valid. These sets are propagated and updated along the paths during search. Consider a planning task $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A} \rangle$. The

Algorithm 1 Initialization of tracking which landmarks and landmark orderings need to be achieved. This procedure requires a set of landmarks $\mathcal{L}$ and a set of landmark orderings $\mathcal{O}$ based on the initial state s_0 of a planning task. The set $open_L[s_0]$ contains all landmarks in $\mathcal{L}$ which are not present in s_0. Similarly, $open_O[s_0]$ contains the landmark orderings for which the first element is not present in s_0.

```
1:  procedure InitializeInformation
2:      $open_L[s_0] \leftarrow \emptyset$
3:      $open_O[s_0] \leftarrow \emptyset$
4:      for all $\ell \in \mathcal{L}$ do
5:          if $\ell \notin s_0$ then
6:              $open_L[s_0] \leftarrow open_L[s_0] \cup \{\ell\}$
7:      for all $\ell_1 \rightarrow \ell_2 \in \mathcal{O}$ do
8:          if $\ell_1 \notin s_0$ then
9:              $open_L[s_0] \leftarrow open_L[s_0] \cup \{\ell_1, \ell_2\}$
10:             $open_O[s_0] \leftarrow open_O[s_0] \cup \{\ell_1 \rightarrow \ell_2\}$
```

landmark generator has produced the landmark graph $\mathcal{G} = \langle \mathcal{L}, \mathcal{O} \rangle$ for s_0. $\mathcal{L}$ denotes the set of landmarks and $\mathcal{O}$ denotes the set of landmark orderings. The graph $\mathcal{G}$ may contain elementary cycles which we collect in the set $\mathcal{C}$.

We now present our algorithms which makes it possible to track this information. For a state s we store active landmarks in $open_L[s]$ and the remaining landmark orderings in $open_O[s]$. Algorithm 1 is concerned with initializing these sets for s_0. All landmarks in $\mathcal{L}$ that are not true in the initial state are inserted in line 6. A landmark ordering $\ell_1 \rightarrow \ell_2$ remains unresolved for as long as ℓ_1 has not been made true, because it is still necessary to make ℓ_2 true after that. Thus, in line 9 all landmarks where ℓ is not true in s_0 are inserted into $open_L[s_0]$; even though ℓ_2 may be true in s_0, it must again be true after ℓ_1 has been true for the first time. The unresolved ordering itself is inserted in the according list in line 10.

During search, this information is updated for each state transition. The procedure provided in Algorithm 2 handles a transition from parent state p to another state s. If s is expanded for the first time, meaning no other path was found to lead to s previously, all landmarks which were unresolved in p and are not true in s remain unresolved. Should the search algorithm expand s again, it means another path to s has been found. The landmarks which were previously found to be unresolved remain in $open_L[s]$; even though they might be true along the new path, since they must be true along all paths they remain unresolved. The set is extended with all landmarks which were unresolved in the new p and are not true in s. Thus, we may gain information by reaching a state through multiple paths. For example, assume s can be reached through two paths: one where a landmark ℓ was true at some point, and another where it was never true. Since ℓ must hold at some point in all plans, but is not required to reach s, we deduce it must still occur in every s-plan.

In order to incorporate this additional information, we only initialize $open_L[s]$ and $open_O[s]$ as empty when s is expanded for the first time in lines 3 and 4. Otherwise we add unresolved landmarks and orderings on top of what we know from the last call of UpdateInformation for the same s. It is never necessary to remove something from the sets, because a new path to s can only prove that landmarks need not be achieved to reach s, but not vice versa. Independent of how often s has been expanded before, the procedure adds a landmark ℓ to $open_L[s]$ in line 7 if it is active in p and not true in s. Similarly, the procedure loops over

Algorithm 2 Updating the set of unfulfilled landmarks and landmark orderings based on a state transition from p to s. The set $open_L[s]$ is extended with all landmarks which were not achieved in the parent state p and are also not modeled by s. Similarly, $open_O[s]$ is extended with the unfulfilled orderings from the parent state where the first element is not modeled by s.

```
1: procedure UpdateInformation(State $p$, State $s$)
2:     if $s$ is expanded for the first time then
3:         $open_L[s] \leftarrow \emptyset$
4:         $open_O[s] \leftarrow \emptyset$
5:     for all $\ell \in open_L[p]$ do
6:         if $\ell \notin s$ then
7:             $open_L[s] \leftarrow open_L[s] \cup \{\ell\}$
8:     for all $\ell_1 \rightarrow \ell_2 \in open_O[p]$ do
9:         if $\ell_1 \notin s$ then
10:            $open_L[s] \leftarrow open_L[s] \cup \{\ell_1, \ell_2\}$
11:            $open_O[s] \leftarrow open_O[s] \cup \{\ell_1 \rightarrow \ell_2\}$
```

all orderings $\ell_1 \rightarrow \ell_2$ which are not resolved in p. If ℓ_1 does not hold in s, then ℓ_2 must still become true in the future and both landmarks as well as the ordering are added to the according sets.

In comparison to using the approach described in Section 5.1.1, a heuristic using this approach is multi-path-dependent. Consider the case where multiple paths are found to a state s. If the latest path brings new information, namely that a landmark ℓ must not be achieved to arrive at s, this could improve our knowledge about all successors of s. Previously, all successors had information that ℓ was achieved before state s and thus do not have it in their list of unfulfilled landmarks (unless an ordering requires so of course). Based on the new information, they could be updated accordingly up to the point where ℓ holds in such a successor. We did not implement this behavior but reevaluate states with our heuristic before they are expanded. If the h-value has changed since the insertion into the open list of A*, it is reinserted with the improved heuristic estimate. A state is expanded by A* only if this is not the case.

Furthermore, we expect tracking landmarks to be less informative than recomputing the landmark graph for every state. Consider again the example task from Figure 3.1. The path $\langle move(t_1, A, B), load(p_{B\rightarrow C}, t_1, B), move(t_1, B, A), unload(p_{B\rightarrow C}, t_1, A)\rangle$ is a valid sequence of actions. It brings us to a state where the package and the truck are at A. When recomputing the landmark graph in this state, we might find that now $\{load(p_{B\rightarrow C}, t_1, A)\}$ is a landmark. Since there exist s_0-plans that do not use $load(p_{B\rightarrow C}, t_1, A)$ (e.g., the optimal plan $\langle move(t_1, A, B), load(p_{B\rightarrow C}, t_1, B), move(t_1, B, C), unload(p_{B\rightarrow C}, t_1, C)\rangle$), it is not a landmark for s_0 and can thus never be in the set $open_L[s]$.

5.2 Decomposing Landmarks from Cycle-Covering

The MHS landmark heuristic from Section 4.4 provides a cost estimate for achieving each landmark once. With cycle-covering we aim for an increase of this value due to the cyclical dependencies between landmarks; if the orderings in the landmark graph induce a cycle, then one of the landmarks in that cycle must be achieved at least twice in every plan. In

$$\begin{tikzcd}
 & \{x, z\} \arrow[d, bend left] & \\
\{x\} \arrow[r] & \{y\} \arrow[u, bend left] \arrow[r] & \{z\}
\end{tikzcd}$$

Figure 5.2: Minimal example landmark graph to show that the MHS LP solution may include the costs to resolve deadlocks in the according problem.

this section we show that decomposing the cycle-covering problem and the landmarks MHS to solve two hypothetically simpler problems does not result in an admissible heuristic.
The cycle-covering problem appears to be another MHS also known as the *Minimum Feedback Vertex Set*; X is the set of all actions and $\mathcal{F}$ contains one element per cycle. Each such element consists of all the actions that achieve a landmark in that cycle. The solution of this MHS denotes the cheapest way to achieve (again) one landmark per cycle. Adding this cost to h^{LM} gives an estimate of the cost for achieving all landmarks plus an additional landmark per cycle. The following counterexample proves that this heuristic is not admissible.
In Figure 5.2 we see four landmarks, thus our landmark LP would be the following.

$$\begin{aligned}
\ell_x: &\quad \mathsf{Y}_x \geq 1 \\
\ell_y: &\quad \mathsf{Y}_y \geq 1 \\
\ell_z: &\quad \mathsf{Y}_z \geq 1 \\
\ell_{xz}: &\quad \mathsf{Y}_x + \mathsf{Y}_z \geq 1
\end{aligned} \tag{5.1}$$

The last constraint ℓ_{xz} is redundant and we get $h^{\text{LM}}(s_0) = 3$ for the planning task belonging to this landmark graph. The cycle LP has exactly one constraint.

$$c: \quad \mathsf{Y}_x + \mathsf{Y}_y + \mathsf{Y}_z \geq 1 \tag{5.2}$$

We must hit one element in cycle c an additional time, so this is simply the union over the actions in $\mathcal{L}(c)$. It is easy to see that the objective value of an MHS LP with only this constraint is 1. Hence, we add 1 to the estimate above and get $h(s_0) = 4$ as our heuristic estimate.
The plan $\pi = \langle x, y, z \rangle$ has a cost lower than the estimate of h. By analyzing the landmark graph we find that π satisfies all landmarks and landmark ordering restrictions.

1. Applying x satisfies ℓ_x and ℓ_{xz}.
2. Applying y satisfies ℓ_y.
    - Furthermore, this satisfies $\ell_x \to \ell_y$ and $\ell_{xz} \to \ell_y$.
3. Applying z satisfies ℓ_z and ℓ_{xz}.
    - This also satisfies $\ell_y \to \ell_z$ and $\ell_y \to \ell_{xz}$.
    - The cycle $\ell_{xz} \to \ell_y \to \ell_{xz}$ is thus completed.

This result proves the possibility that deadlocks are already resolved in the MHS solution for only the landmarks. Thus, it is inadmissible to add the solution of an MHS for hitting all cycles in the landmark graph to h^{LM}.

5.3 Additional Cycle-Covering Constraints for the Landmarks LP

Instead of a separate MHS problem, it is possible to enhance h^{LM} with additional constraints for the LP. The resulting heuristic is part of the operator-counting framework same as the MHS landmark heuristic. Besides the landmark constraints according to Equation (4.6), there is a constraint for each active cycle in s. While the landmark constraints are an immediate consequence of the hitting set characteristic, the cycle-covering constraints are not as straightforward.

5.3.1 LP Cycle Constraints

A constraint as presented in Equation (5.3) is added for every elementary cycle c which is active in state s.

$$\sum_{\ell \in \mathcal{L}(c)} \sum_{a \in \ell} \mathsf{Y}_a \geq n + 1 \tag{5.3}$$

In combination with the landmark constraints it states that all landmarks in the cycle must be achieved at least once (captured by the number n) and one of them must be achieved an additional time due to the cyclic dependencies. The generalized cycle-covering heuristic corresponds to the objective value of the described LP in each state.

We explain the idea further based on the logistics planning task in Figure 3.3; among others, it entails the landmarks $\ell_B = \{\mathit{move}(t_1, A, B), \mathit{move}(t_1, C, B)\}$ and $\ell_C = \{\mathit{move}(t_1, A, C), \mathit{move}(t_1, B, C)\}$. To keep the example small, we restrict it to only consider the *move* operators. (Assume that packages are loaded and unloaded automatically when arriving at their departure point or destination, respectively.) The last landmark concerned with moving to locations is $\ell_A = \{\mathit{move}(t_1, B, A), \mathit{move}(t_1, C, A)\}$. Since the truck is at A in the beginning, it is not active in the initial state of this planning task.

The landmarks ℓ_B and ℓ_C are cyclically dependent on one another due to the reasonable orderings $\ell_B \to_{\mathrm{r}} \ell_C$ and $\ell_C \to_{\mathrm{r}} \ell_B$. The landmark graph in Figure 3.3b shows all relevant information when only considering *move* operators. To simplify the notation, we write $\mathsf{Y}_{x \to y}$ for the operator-counting variable for $\mathit{move}(t_1, x, y)$. The complete set of LP constraints for landmarks and cycles is provided in Equation (5.4).

$$\begin{aligned}
\ell_B:&\quad \mathsf{Y}_{A\to B} + \mathsf{Y}_{C\to B} && \geq 1 \\
\ell_C:&\quad && \mathsf{Y}_{A\to C} + \mathsf{Y}_{B\to C} \geq 1 \\
c:&\quad \mathsf{Y}_{A\to B} + \mathsf{Y}_{C\to B} + \mathsf{Y}_{A\to C} + \mathsf{Y}_{B\to C} && \geq 3
\end{aligned} \tag{5.4}$$

The landmark heuristic h^{LM} solely considers the landmark constraints ℓ_B and ℓ_C but not the cycle constraint c. A solution to the according LP for $h^{\mathrm{LM}}(s_0)$ is $\mathsf{Y}_{A\to B} = \mathsf{Y}_{B\to C} = 1$ and all other variables are set to 0. In particular, the according actions can be applied in sequence; $\langle \mathit{move}(t_1, A, B), \mathit{move}(t_1, B, C)\rangle$ is a valid sequence of actions. However, not all

packages are at their destination at the end of this sequence, because $p_{C\to B}$ was only just loaded into t_1.
Enhancing the heuristic with the cycle constraint c yields a different solution to the LP. It is evident that the solution from before does not satisfy the cycle constraint. Hence, the LP solver must increase the variable values in order to satisfy it. While there are many ways to achieve this, the solution that also can be applied as an action sequence and is a plan would be to set $\mathsf{Y}_{A\to B} = \mathsf{Y}_{B\to C} = \mathsf{Y}_{C\to B} = 1$ and everything else to 0. In this example, the cycle-covering heuristic estimates that 3 move operators must be applied, while the landmark heuristic only accounts 2.

5.3.2 Overlapping Landmarks in Cycles

It may be the case that the same action appears in multiple landmarks within the same cycle. If so, the according LP variable occurs multiple times on the left-hand side of Equation (5.3). Consider an action a that occurs in x landmarks of cycle c where $x \in \mathbb{N}$. Then, x is the coefficient of Y_a in the LP constraint for c. The interpretation of this is that we can potentially make x of the landmarks in c true by applying one single operator.
In the extreme case, all landmarks in the cycle contain the same action. Generally speaking, two consecutive landmarks may have a non-empty intersection because of the way reasonable orderings are defined; in case of a reasonable ordering $\ell \to_{\mathrm{r}} \ell'$, it is possible and allowed for ℓ and ℓ' to be achieved simultaneously. If $a \in \bigcap_{\ell \in \mathcal{L}(c)} \ell \neq \emptyset$, this means that all landmarks in c can potentially be achieved at once by applying a. Hence, the LP constraint for c according to Equation (5.3) is incorrect; when removing all variables but Y_a from the constraint, only the following remains:

$$n\mathsf{Y}_a \geq n + 1 \tag{5.5}$$

where $n = |\mathcal{L}(c)|$. In order to satisfy this constraint, it is necessary to set $\mathsf{Y}_a \geq \frac{n+1}{n} > 1$, but actually, $\mathsf{Y}_a = 1$ resolves the cycle completely.
The following example illustrates how it is even possible that such cycles occur in landmark graphs. It is adapted from Hoffmann et al.'s (2004) Figure 2. They use the STRIPS formalism together with fact landmarks, similar to the way presented here. There are seven facts L, L', P_1, P_2, P'_2, P_3, and P'_3. Initially only P_1 is true, and the goal is to have $L \wedge L'$. Their actions are:

$$\begin{aligned}
&opL_1: && pre(opL_1) = P_1, && \mathit{eff}(opL_1) = L \wedge P_2 \wedge \neg P_1 \\
&opL'_1: && pre(opL'_1) = P_1, && \mathit{eff}(opL'_1) = L' \wedge P'_2 \wedge \neg P_1 \\
&opL_2: && pre(opL_2) = P'_2, && \mathit{eff}(opL_2) = L \wedge P_3 \wedge \neg L' \wedge \neg P'_2 \\
&opL'_2: && pre(opL'_2) = P_2, && \mathit{eff}(opL'_2) = L' \wedge P'_3 \wedge \neg L \wedge \neg P_2 \\
&opL_3: && pre(opL_3) = P'_3, && \mathit{eff}(opL_3) = L \wedge \neg P'_3 \\
&opL'_3: && pre(opL'_3) = P_3, && \mathit{eff}(opL'_3) = L' \wedge \neg P_3
\end{aligned} \tag{5.6}$$

Hoffmann et al. show with this example that sometimes it is not possible to find a plan that obeys every reasonable ordering. Fundamentally, this is the motivation of this thesis, because such situations are the source of cycles in the landmark graph.

$$\begin{tikzcd}
& L, P_2 \arrow[r, "opL_2'"] & L', P_3' \arrow[dr, "opL_3"] & \\
P_1 \arrow[ur, "opL_1"] \arrow[dr, "opL_1'"'] \arrow[rrr, dashed, "goal"] & & & L, L' \\
& L', P_2' \arrow[r, "opL_2"] & L, P_3 \arrow[ur, "opL_3'"'] &
\end{tikzcd}$$

Figure 5.3: State space of the example from Hoffmann, Porteous, and Sebastia (2004) extended by an operator *goal* marked as dashed. The landmark graph for this problem has a cycle of reasonable orderings which renders the estimate of h^{cycle} inadmissible when including the according constraint in the LP.

We introduce an additional action which achieves the top-level goals of this planning task directly:

$$goal: \quad pre(goal) = P_1, \quad eff(goal) = L \wedge L' \tag{5.7}$$

The corresponding state space is displayed in Figure 5.3 where *goal* is highlighted as a dashed arc. The goal facts L and L' are the only active landmarks in this problem. There are three plans: $\pi_1 = \langle opL_1, opL_2', opL_3 \rangle$, $\pi_2 = \langle opL_1', opL_2, opL_3' \rangle$, and $\pi^* = \langle goal \rangle$. The latter is the optimal plan in a uniform cost setting.
According to the definition, there is a reasonable ordering $\ell \to_{\mathrm{r}} \ell'$ if the following holds along all plans where ℓ' becomes true before ℓ:

- ℓ' must be destroyed in order to achieve ℓ, and
- it is necessary that ℓ' is true at the same time or after ℓ (e.g., ℓ' is part of the goal).

In the above example, π_1 is the only plan where L becomes true before L'. In order to achieve L' along π_1 it is necessary to delete L, but L is again necessary in the goal. Hence, there is a reasonable ordering $L' \to_{\mathrm{r}} L$. Equivalently, there is an ordering $L \to_{\mathrm{r}} L'$ because along all plans where L' is true before L (i.e., π_2), L' is removed to achieve L. These two landmark orderings form a cycle in the according landmark graph.
The disjunctive action landmarks corresponding to L and L' are $\ell_L = \{opL_1, opL_2, opL_3, goal\}$ and $\ell_{L'} = \{opL_1', opL_2', opL_3', goal\}$, respectively. According to Equation (5.3), they entail the following cycle constraint in the LP of the initial state of the above planning task:

$$\sum_{a \in \ell_L} \mathsf{Y}_a + \sum_{a \in \ell_{L'}} \mathsf{Y}_a = \mathsf{Y}_{opL_1} + \mathsf{Y}_{opL_2} + \mathsf{Y}_{opL_3} + \mathsf{Y}_{opL_1'} + \mathsf{Y}_{opL_2'} + \mathsf{Y}_{opL_3'} + 2\mathsf{Y}_{goal} \geq 3 \tag{5.8}$$

In the LP, the objective function $\sum_{a \in \mathcal{A}} \mathsf{Y}_a$ is minimized and the cheapest solution to fulfill this constraint is to set $\mathsf{Y}_{goal} = 1.5$. Consequently, $\sum_{a \in \mathcal{A}} \mathsf{Y}_a \geq 1.5 > 1 = cost(\pi^*)$ and hence the LP solution overestimates the true cost. It is thus necessary to exclude all cycles which can be resolved by a single action from the LP. Only then, the generalized cycle-covering heuristic is an admissible extension of the landmark heuristic h^{LM}.

5.3.3 Cycle-Covering Heuristic

We conclude Section 5.3 by defining the cycle-covering heuristic h^{cycle} and provide its theoretical implications.

Definition 9 (Generalized cycle-covering heuristic)**.** *Let $\mathcal{G} = \langle \mathcal{L}, \mathcal{O} \rangle$ be a landmark graph in a planning task $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A} \rangle$ and let $\mathcal{C}$ be the set of distinct elementary cycles in $\mathcal{G}$. Furthermore, let $\mathcal{C}_\emptyset \subseteq \mathcal{C}$ be the set of cycles with an empty intersection of their landmarks: $\mathcal{C}_\emptyset = \{c \in \mathcal{C} \mid \bigcap_{\ell \in \mathcal{L}(c)} \ell = \emptyset\}$.*
The cycle-covering LP for $\mathcal{G}$ is

$$\begin{aligned} \min \sum_{a \in \mathcal{A}} \mathsf{Y}_a \, cost(a) \quad & s.t. \\ \sum_{a \in \ell} \mathsf{Y}_a \geq 1 \quad & \text{for all } \ell \in \mathcal{L} \text{ and} \\ \sum_{\ell \in \mathcal{L}(c)} \sum_{a \in \ell} \mathsf{Y}_a \geq |\mathcal{L}(c)| + 1 \quad & \text{for all } c \in \mathcal{C}_\emptyset . \end{aligned}$$

The generalized cycle-covering heuristic h^{cycle} *is the objective value of the cycle-covering LP for the landmark graph in each state s of $\mathcal{T}$.*

Note that a feasible solution for the cycle-covering LP always exists in solvable planning tasks; setting $\mathsf{Y}_a = 2$ for all $a \in \mathcal{A}$ satisfies all constraints of the LP.
The landmark heuristic h^{LM} can never yield a higher heuristic estimate than h^{cycle}; it considers only a subset of the LP constraints of the cycle-covering heuristic and thus the LP is less constrained.

Theorem 1. *The cycle-covering heuristic h^{cycle} dominates the landmark heuristic h^{LM} for the same set of active landmarks and cycles.*

Proof. Let $\mathcal{T}$ be a planning task with states S and let C^{LM} be the function that maps a state to its landmark constraints. Similarly, let C^{cycle} be the function that maps the states of $\mathcal{T}$ to the according cycle-constraints.
The constraint set for the LP evaluating $h^{\mathrm{LM}}(s)$ is $C^{\mathrm{LM}}(s)$. The constraint set for the LP evaluating $h^{cycle}(s)$ is $C^{\mathrm{LM}}(s) \cup C^{cycle}(s)$.
We use Proposition 1: Since $C^{\mathrm{LM}}(s) \subseteq C^{\mathrm{LM}}(s) \cup C^{cycle}(s)$ holds for all $s \in S$, the cycle-covering heuristic h^{cycle} dominates the landmark heuristic h^{LM}. $\square$

5.4 Strengthening the Cycle-Covering Heuristic via Ordering Types

In Section 3.2 we provide a collection of different ordering types between landmarks. The constraints of h^{cycle} consider the cycles induced by these orderings but do not take the ordering types into account. The following observation suggests that valuable information is thereby neglected. Cycles can only occur where reasonable orderings are present; by definition, natural orderings cannot form cycles on their own. We can use this knowledge to define a stronger LP constraint for the cycles in the landmark graph. In this section we explain how the cycle-covering heuristic can be improved based on the ordering types occurring in a cycle.

$$\begin{tikzcd}
\{x\} \arrow[r, bend left, "\mathrm{n}"] & \{y\} \arrow[l, bend left, "\mathrm{r}"] \arrow[r, bend left, "\mathrm{r}"] & \{z\} \arrow[l, bend left, "\mathrm{r}"]
\end{tikzcd}$$

Figure 5.4: Small example landmark graph to demonstrate the superiority of h^{ord} over h^{cycle}. The natural ordering $\ell_x \rightarrow_{\mathrm{n}} \ell_y$ prevents ℓ_y to be achieved before ℓ_x. Thus, the according cycle must be resolved by achieving ℓ_x twice.

Consider a cycle $c : \ell_x \rightarrow_{\mathrm{n}} \ell_y \rightarrow_{\mathrm{r}} \ell_x$ as shown in Figure 5.4. The landmark ℓ_y cannot be resolved before ℓ_x because there is a natural ordering $\ell_x \rightarrow_{\mathrm{n}} \ell_y$. The definition of such a natural ordering states that it is impossible to achieve ℓ_y without achieving ℓ_x first. Thus, the only possibility to resolve c is to achieve ℓ_x twice. In other words, only the landmarks with an incoming reasonable ordering are candidates for being necessary multiple times. This observation allows for a stronger definition of the cycle constraints in the LP; only the landmarks with an incoming reasonable ordering should be considered.
Let $\mathcal{G} = \langle \mathcal{L}, \mathcal{O} \rangle$ be a landmark graph for a state s in a planning task and let c be an elementary cycle in $\mathcal{G}$. Then $\mathcal{L}_r = \{\ell' \mid \ell \rightarrow_{\mathrm{r}} \ell' \in \mathcal{O}(c)\}$ is the nonempty set of landmarks with an incoming reasonable ordering in c. Compared to Equation (5.3), the following is a stronger version of the LP constraint for cycle c.

$$\sum_{\ell \in \mathcal{L}_r} \sum_{a \in \ell} \mathsf{Y}_a \geq |L_r| + 1 \tag{5.9}$$

In order to clarify the idea, we explain it on the example from Figure 5.4. We provide the complete list of LP constraints according to Definition 9.

$$\begin{array}{rlllll}
\ell_x : & \mathsf{Y}_x & & & \geq 1 \\
\ell_y : & & \mathsf{Y}_y & & \geq 1 \\
\ell_z : & & & \mathsf{Y}_z & \geq 1 \\
c_{xy} : & \mathsf{Y}_x + & \mathsf{Y}_y & & \geq 3 \\
c_{yz} : & & \mathsf{Y}_y + & \mathsf{Y}_z & \geq 3
\end{array} \tag{5.10}$$

Setting $\mathsf{Y}_x = \mathsf{Y}_z = 1$ and $\mathsf{Y}_y = 2$ is a valid solution for the according LP. However, implementing the stronger constraints yields the following:

$$\begin{array}{rlllll}
c_{xy} : & \mathsf{Y}_x & & & \geq 2 \\
c_{yz} : & & \mathsf{Y}_y + & \mathsf{Y}_z & \geq 3
\end{array} \tag{5.11}$$

The constraint for c_{xy} is more restrictive while c_{yz} remains the same because it consists only of reasonable orderings. The above solution does not satisfy the constraint c_{xy}, because it requires Y_x to be at least 2. Reducing Y_y to 1 is not an option unless Y_z is also increased to 2 because otherwise the constraint c_{yz} is unsatisfied. Hence, considering the ordering types of the landmark orderings involved in the cycles increases the LP solution in this example.

We define the ordering-aware cycle-covering heuristic h^{ord}.

Definition 10 (Ordering-aware cycle-covering heuristic)**.** *Let $\mathcal{G} = \langle \mathcal{L}, \mathcal{O} \rangle$ be a landmark graph in a planning task $\mathcal{T} = \langle \mathcal{V}, s_0, G, \mathcal{A} \rangle$ and let $\mathcal{C}$ be the set of distinct elementary cycles in $\mathcal{G}$. Furthermore, let $\mathcal{L}_r(c) = \{\ell' \mid \ell \to_{\mathrm{r}} \ell' \in \mathcal{O}(c)\}$ be the set of landmarks with incoming reasonable orderings in all cycles $c \in \mathcal{C}$.*
The ordering-aware cycle-covering LP for $\mathcal{G}$ is

$$\begin{aligned} \min \sum_{a \in \mathcal{A}} \mathsf{Y}_a\, cost(a) &\quad s.t. \\ \sum_{a \in \ell} \mathsf{Y}_a \geq 1 &\quad \textit{for all } \ell \in \mathcal{L} \textit{ and} \\ \sum_{\ell \in \mathcal{L}_r(c)} \sum_{a \in \ell} \mathsf{Y}_a \geq |\mathcal{L}_r(c)| + 1 &\quad \textit{for all } c \in \mathcal{C}. \end{aligned}$$

The ordering-aware cycle-covering heuristic h^{ord} *is the objective value of the ordering-aware cycle-covering LP for the landmark graph in each state s of $\mathcal{T}$.*

The ordering-aware cycle-covering LP is feasible for the same reason than the cycle-covering LP.[7] The ordering-aware cycle-covering heuristic h^{ord} improves the heuristic estimates compared to the cycle-covering heuristic h^{cycle} presented in the previous section.

Theorem 2. *The ordering-aware cycle-covering heuristic h^{ord} dominates the cycle-covering heuristic h^{cycle} for the same set of active landmarks and cycles.*

Proof. Consider a cycle c consisting of $n = |\mathcal{L}(c)|$ landmarks. Let $\mathcal{L}_r \subseteq \mathcal{L}(c)$ be the set of landmarks with incoming reasonable orderings in c. The stronger constraint for the cycle-covering LP according to Equation (5.9) is

$$\sum_{\ell \in L_r} \sum_{a \in \ell} \mathsf{Y}_a \geq m + 1 \tag{5.12}$$

where $m = |\mathcal{L}_r|$. It forms an LP together with other cycle constraints as well as the basic landmark constraints. If we add the landmark constraints $\sum_{a \in \ell} \mathsf{Y}_a \geq 1$ for all landmarks $\ell \in \mathcal{L}(c) \setminus \mathcal{L}_r$ to the constraint in Equation (5.12), it becomes the following.

$$\begin{aligned} \sum_{\ell \in \mathcal{L}_r} \sum_{a \in \ell} \mathsf{Y}_a + \sum_{\ell \in \mathcal{L}(c) \setminus \mathcal{L}_r} \sum_{a \in \ell} \mathsf{Y}_a &\geq m + 1 + n - m \\ \sum_{\ell \in \mathcal{L}(c)} \sum_{a \in \ell} \mathsf{Y}_a &\geq n + 1 \end{aligned} \tag{5.13}$$

This is precisely the cycle constraint according to Equation (5.3). Thus, the weaker cycle-covering constraint is implied by the LP with the stronger constraint. □

[7] The ordering-aware cycle constraints are infeasible in the case of cycles consisting only of natural orderings. Such cycles can exist only in unsolvable planning tasks.

6 Experimental Evaluation

In order to empirically evaluate the proposed heuristics, we implemented them in the Fast Downward[8] planner version 19.06 (Helmert 2006; Helmert 2009). In the experiments we consider 122 domains with a total of 3298 planning tasks from the IPC benchmark set[9]. These domains do not include axioms and domains where the translator component of Fast Downward yields non-deterministic results are also not considered. All experiments are conducted on Intel Xeon E5-2660 CPU's running on 2.2GHz. Calculations were performed using Downward Lab (Seipp, Pommerening, Sievers, and Helmert 2017) at sciCORE[10] scientific computing center at the University of Basel. We allow the tasks to run up to 30 minutes and to require at most 4GB of memory. The heuristics based on linear programming are computed using IBM CPLEX[11] version 12.9.0 as an LP solver (IBM 2019).
Our contributions are not concerned with landmark generation. Thus, we use landmarks produced by the landmark generators already implemented in Fast Downward, of which some are described in Section 3.4. We refer to them as $\mathrm{LM}^{\mathrm{ZG}}$ (Zhu and Givan 2003), $\mathrm{LM}^{\mathrm{RHW}}$ (Richter et al. 2008), and LM^{h^m} (Keyder et al. 2010). For the latter, we use $m = 2$ and split conjunctive landmarks into two separate fact nodes in the landmark graph. Additionally, Fast Downward offers a method for *exhaustive* landmark generation $\mathrm{LM}^{\mathrm{exh}}$ that checks for each fact whether it is a landmark in the relaxed planning task. Lastly, $\mathrm{LM}^{\mathrm{merged}}$ combines the results of any number of landmark generators. Specifically, it merges the landmark graphs created by its components into one graph. In our experiments this configuration combines all other landmark generators $\mathrm{LM}^{\mathrm{ZG}}$, $\mathrm{LM}^{\mathrm{RHW}}$, LM^{h^m}, and $\mathrm{LM}^{\mathrm{exh}}$ into one.

6.1 Cycle Detection

Since the fundamental idea of this thesis is based on cyclical dependencies of landmarks, we start by showing that cycles appear regularly in landmark graphs. In order to do so, we changed the implementations of landmark generators in Fast Downward so that cycles

[8] http://www.fast-downward.org
[9] https://github.com/aibasel/downward-benchmarks
[10] http://scicore.unibas.ch/
[11] https://www.cplex.com

	Domains	Tasks	Maximum	Sum	Average	Timeout
$\mathrm{LM}^{\mathrm{exh}}$	0	0	0	0	–	0
$\mathrm{LM}^{\mathrm{ZG}}$	0	0	0	0	–	41
LM^{h^m}	27	183	194’754	665’804	3’638.3	750
$\mathrm{LM}^{\mathrm{RHW}}$	31	274	779’493	1’601’012	5’843.1	73
$\mathrm{LM}^{\mathrm{merged}}$	26	205	3’648’030	18’628’971	90’873.0	747

Table 6.1: Overview of the amount of cycles found by the different landmark generators. The presented numbers correspond to the cycles in the initial state of each planning task.

within the landmark graphs are not removed during their construction. In order to find all elementary cycles in these graphs, we use our own implementation of Johnson’s Algorithm (Johnson 1975). The different landmark generators compute distinct landmark graphs, making the amount of cycles dependent on these methods.

In our opinion, defining a measure to evaluate the susceptibility of planning tasks to cycles is difficult. In Table 6.1 give an overview of the number of cycles in the landmark graphs of the initial states. The columns “domains” and “tasks” show how many domains and tasks have cycles. The “maximum” refers to the amount of cycles found in the task with the most cycles. In “sum” we provide the sum of all cycles in the initial states of all tasks and “average” denotes the average amount of cycles in cyclical tasks (i.e., “sum” divided by “tasks”). Computing the landmark graph of the initial state is not possible within the time-limits for the amount of tasks provided under “timeout”.

The number of cyclical domains and tasks found by $\mathrm{LM}^{\mathrm{merged}}$ is lower than that of its components. This is because our time limit is exceeded before completing the computation and merger of all component landmark graphs in some cases. In particular, the computation of LM^{h^m} landmarks is the most problematic. We cannot explain why LM^{h^m} ran into more timeouts than $\mathrm{LM}^{\mathrm{merged}}$.

Despite the larger number of timeouts with $\mathrm{LM}^{\mathrm{merged}}$, its average amount of cycles indicates that combining the landmark graphs of the different methods is beneficial. A method may even be interesting if it does not find cycles on its own. For example, consider two landmark generators which find the same set of landmarks $\{\ell, \ell'\}$. The first method finds an ordering $\ell \to \ell'$ and the second method finds $\ell' \to \ell$. Independently, they are acyclic, but when combined, they produce the cycle $\ell \to \ell' \to \ell$. Hence, including $\mathrm{LM}^{\mathrm{exh}}$ and $\mathrm{LM}^{\mathrm{ZG}}$ in $\mathrm{LM}^{\mathrm{merged}}$ even though they never yield cycles can lead to an increased number of cycles in $\mathrm{LM}^{\mathrm{merged}}$.

For a domain-wise comparison of cyclicity we use $\mathrm{LM}^{\mathrm{RHW}}$ which finds the most tasks with cycles. Figure 6.1 displays the amount of cycles found for all problem instances. (Visualizing the numbers for the other landmark generators results in similar output but scaled accordingly.) The plotted functions $f_i(x) = f_{i-1}(x) + y_i$ (using $f_0(x) = 0$) depict that in the corresponding domain there are y_i problems with x or more cycles. Note that the plot is logarithmic in x. Domains, where no cycles are found at all, are not shown.

The above measure of cycles is flawed because it only considers one state from every planning task; if the landmark graph of the initial state is acyclic, this does not necessarily mean that the landmark graphs of all states are acyclic. A domain may have characteristics which

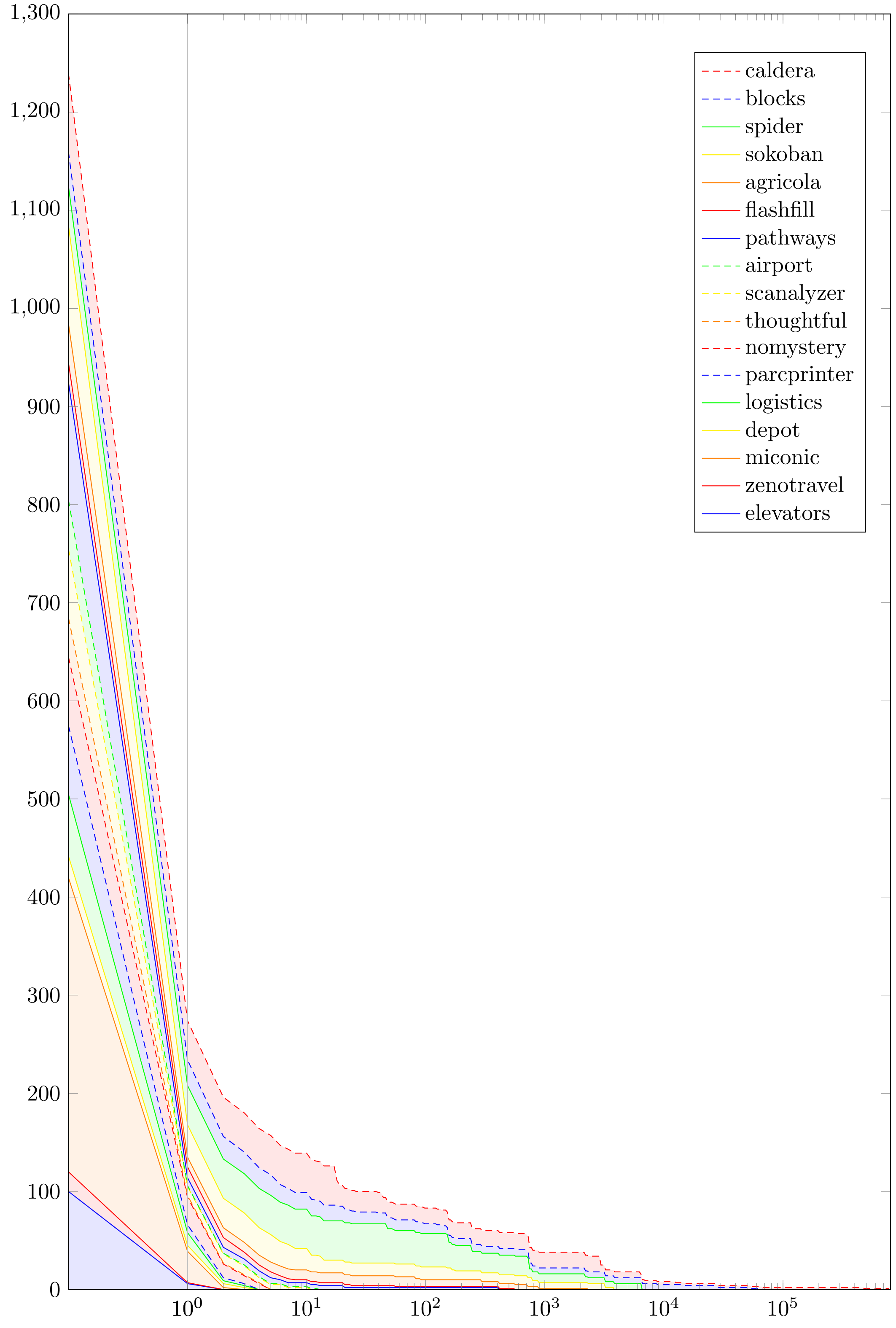


Figure 6.1: Amount of planning tasks with x or more cycles in the initial state stacked by domain. Displayed are the domains with at least 1 cycle in the initial state and domains with a shared prefix are combined. For example, logistics98 and logistics00 are displayed collectively as logistics.

	Domains	Tasks	Maximum	Sum	Average
LM^{exh}	0	0	0	0	–
LM^{ZG}	7	89	13	210	2.4
LM^{h^m}	44	443	11’001’628	36’021’787	81’313.3
LM^{RHW}	87	1’220	16’894’636	686’455’439	562’668.4

Table 6.2: Overview of the amount of cycles found by the different landmark generators. The presented numbers correspond to the state with the maximal number of cycles for each planning task.

never lead to cycles in the initial state, but this is not generally true for all states in that domain. By considering more states we get a more general assessment of cyclicity in the problem instances. Thus, we provide another evaluation in Table 6.2 where we consider the landmark graphs of all states encountered when searching for a plan. The numbers correspond to the maximal number of cycles found in any state of each planning task. Hence, “maximum” is the planning task which has the most cycles in any state. Similarly, “sum” adds up the maximal amount of cycles found for each problem. $\text{LM}^{\text{merged}}$ is missing in Table 6.2 because we found recomputing and merging four landmark graphs in every state to be infeasible.

The number of cyclical tasks is much higher if all encountered states of each planning task are considered. Again, a domain-wise overview using LM^{RHW} is provided in Figure 6.2. This indicates that only considering the initial state is not conclusive. In planning we may benefit from a cycle-covering landmark heuristic even if the landmark graph of the initial state is acyclic.

6.2 Recomputing vs. Tracking Active Landmarks

In the preceding section we show that cycles occur regularly in the landmark graphs of planning tasks. Now, we start using the according information in heuristic search. Two approaches to decide which landmarks are active in the states of a planning task are described in Section 5.1. The first method which we refer to as RECOMP recomputes the landmark graph in every encountered state. Our second approach called TRACK uses the landmark graph from the initial state and tracks which landmarks have since been achieved. According to the previous section, we should always use RECOMP; using TRACK if the initial state is acyclic entails that there are never cycle constraints in the LPs. However, there is more to the decision between these methods; in this section we discuss further advantages or disadvantages. In order to do so, we compare their performance using h^{LM}. The results when also considering cycles (e.g., with h^{cycle}) look similar.

In Table 6.3 we display the coverage (i.e., the amount of problems solved) along with the main error sources for unsolvability. It can be observed that the number of solved tasks is significantly smaller when recomputing the landmark graph in every state. The difference is mostly due to time restrictions terminating the search before finding a solution with RECOMP. Memory restrictions were never a problem for LM^{RHW} and LM^{exh}, and only rarely for LM^{ZG}. Over 600 planning tasks could not be solved using LM^{h^m} for both RECOMP and TRACK due to memory restrictions.

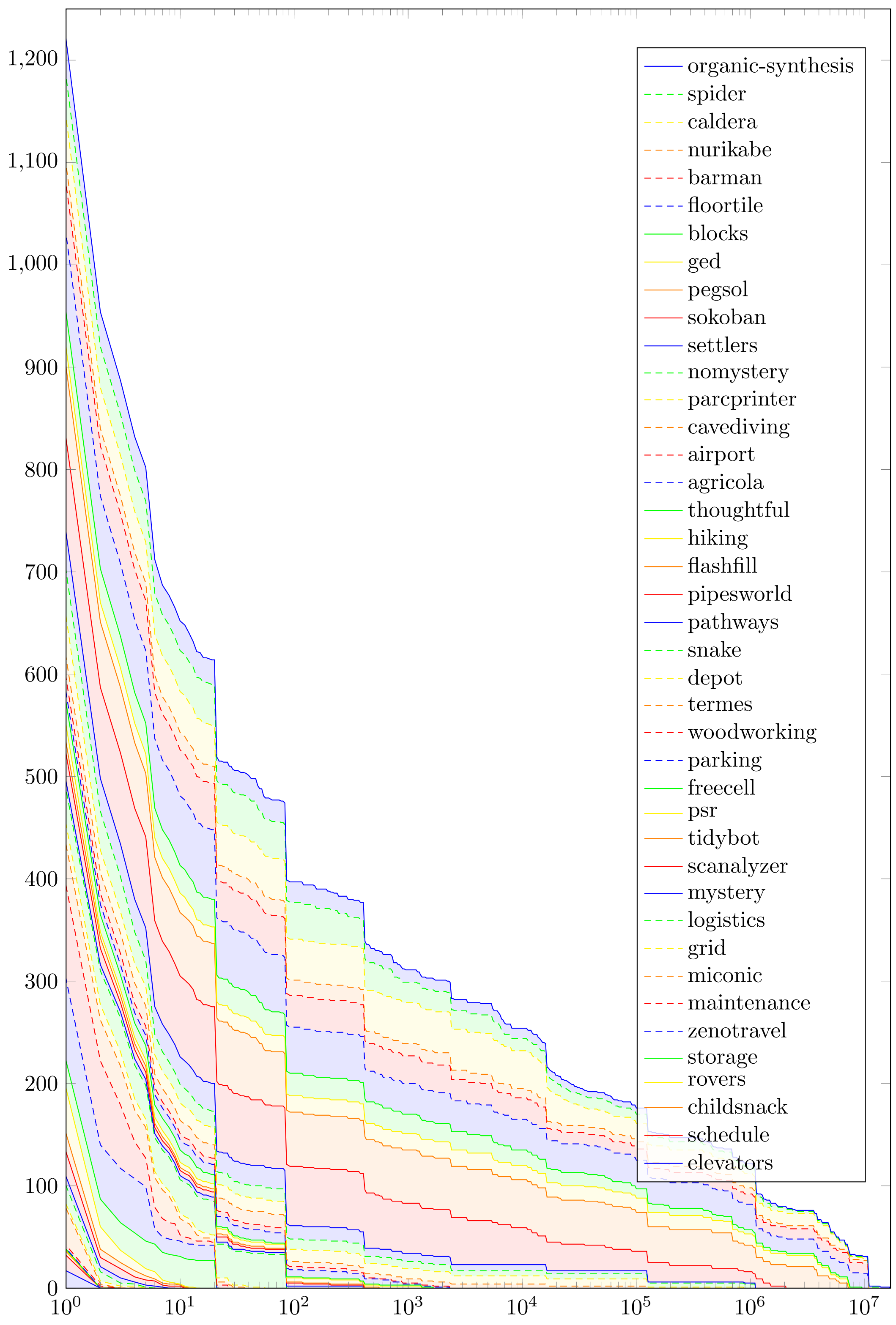


Figure 6.2: Comparing the maximal amount of cycles found in any state of each planning task. The amount of tasks with x or more cycles is stacked by domain and domains with a shared prefix are combined. There are 1'220 problems with at least one cycle which corresponds to roughly half of the planning tasks available in the listed domains.

		LM$^{\text{RHW}}$	LM$^{\text{ZG}}$	LM$^{h^m}$	LM$^{\text{exh}}$	LM$^{\text{merged}}$
Coverage						
	RECOMP	840	636	484	765	–
	TRACK	1000	1042	938	976	998
Errors						
Timeout	RECOMP	2402	2577	2135	2478	–
	TRACK	2240	2197	1685	2264	1627
Out of Memory	RECOMP	0	27	622	0	–
	TRACK	0	1	616	0	613

Table 6.3: Coverage results of the h^{LM} heuristic comparing two approaches to determine the set of active landmarks. We use RECOMP for the configuration which recomputes the landmark graph for every state and TRACK when updating the information from the initial state.

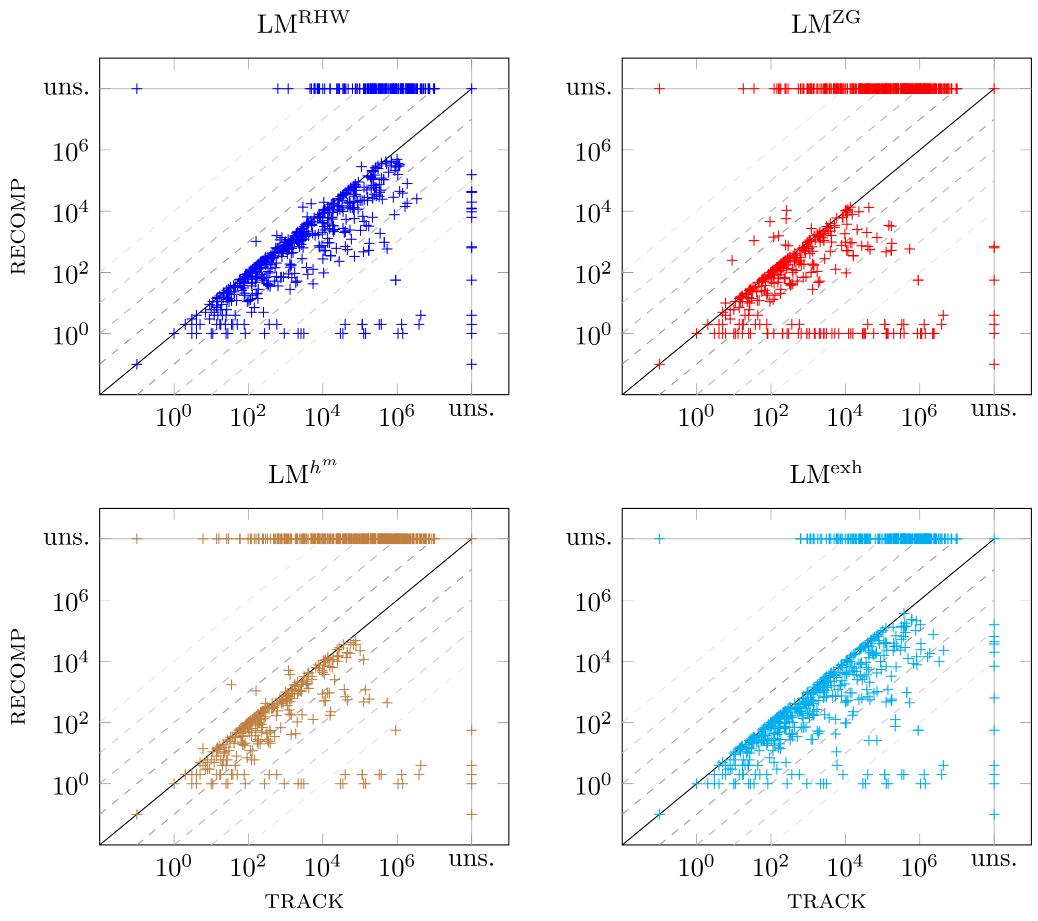


Figure 6.3: The number of expanded states before the last f-layer for each problem using the different landmark generators. Data points at the upper or right end of the scale were unsolved within the time- and memory-bounds. Recomputing the landmark graph in all evaluated states reduces the number of necessary expansions heavily.

We compare the number of expanded states before the last f-layer in Figure 6.3. There, the lower coverage of RECOMP can also be observed since all data points in the top rows correspond to unsolved tasks. At the same time, it is evident from these plots that in most cases RECOMP expands significantly fewer states. This indicates that RECOMP has more accurate information compared to TRACK. In particular, we assume that RECOMP recognizes landmarks which are required again or finds new landmarks. In contrast, the tracking approach cannot find heuristic estimates higher than in the initial state. On the contrary, some landmarks are likely achieved independently of whether search proceeds towards the goal. This leads to reduced h-values which in turn lead to a higher priority for such states to be expanded by A*.

The few instances where RECOMP expands more states than TRACK indicate that sometimes it is not beneficial to recompute the landmark graph when transitioning to a new state. Remember, that the set of landmarks found by the landmark generators is mostly an incomplete approximation of all landmarks. We suspect that sometimes landmarks are lost when transitioning between two states s and p through an action a. Consider a landmark $\ell \in \mathcal{L}(s)$ which is not resolved when applying a. It may be possible that ℓ is not found in the landmark graph for p and thus it is not considered as a landmark with RECOMP although it is still valid. A combination of RECOMP and TRACK which extends the recomputed landmarks with unfulfilled landmarks from the predecessor states could thus be beneficial.

While the number of expansions strongly speak for RECOMP, the data provided in Figure 6.4 has the opposite effect. It shows that recomputing the landmarks in all encountered states is infeasible most of the time. The benefit of having a more informed heuristic does not pay off in terms of the running time; recomputing the landmarks in each state requires exponentially more time than tracking the landmarks. The most negative effect is observed when using LM^{h^m}.

One more problem which occurs with RECOMP is the size of the LP when a lot of cycles are present. As shown in Table 6.2 there are problems with almost 17 million cycles. Since each cycle implies one LP constraint, LPs for such states become huge. While time requirements may be a problem, the main issue is the high memory consumption to store all these constraints. The results of an unreported experiment emphasize this assumption; the number of tasks running out of memory using $\mathrm{LM}^{\mathrm{RHW}}$ with RECOMP increases from 0 to 281 when using h^{cycle} instead of h^{LM}.

The upcoming sections analyze the performance of the cycle-covering heuristics. Since the results in this section are inconclusive, we do not exclude either RECOMP or TRACK from these experiments. Instead we show results for both methods or explain why it might not be interesting to consider both of them.

6.3 Cycle-Covering Heuristic

In Section 6.1 we have shown that cycles are not a rare occurrence in landmark graphs. An analysis of whether considering cycles in a heuristic pays off follows in this section. We first compare our implementation of the generalized cycle-covering heuristic h^{cycle} to the landmark heuristic h^{LM}. Moreover, we show how the stronger LP constraints for h^{ord} influence the performance of cycle-covering.

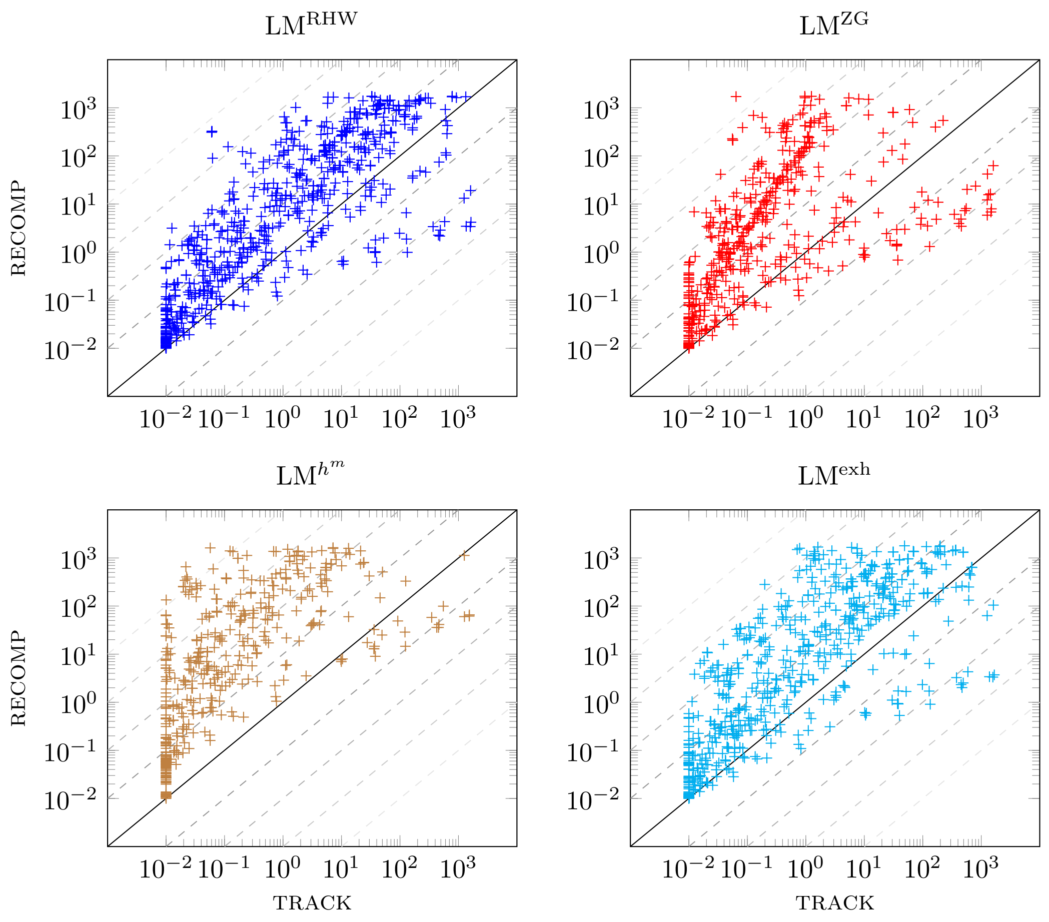


Figure 6.4: Search time in seconds comparing RECOMP and TRACK with h^{LM}. Points below the diagonal correspond to problem instances that find a solution faster when recomputing the landmark graph in every state. Similarly, TRACK is faster for all instances above the diagonal which happens more often with all landmark generators.

All presented results consider only the TRACK approach because its coverage is higher for h^{LM} and it is sufficient for this analysis. The unreported results for RECOMP look similar or even identical in some cases. For example, we compare the h-values in the initial state using the different heuristics; since TRACK and RECOMP compute the identical landmark graphs in the initial state, those values are equal for both methods.

Furthermore, only the landmark generators which yield cycles in the initial states are of interest; evidently, h^{LM} and h^{cycle} are identical in states where no cycles are present because they consider the same set of LP constraints in this case. While $\mathrm{LM}^{\mathrm{RHW}}$ and LM^{h^m} as well as $\mathrm{LM}^{\mathrm{merged}}$ produce cycles consistently, this is not the case for $\mathrm{LM}^{\mathrm{ZG}}$ and $\mathrm{LM}^{\mathrm{exh}}$. Not a single instance was found to have cycles in the landmark graph of any evaluated state when using $\mathrm{LM}^{\mathrm{exh}}$. The $\mathrm{LM}^{\mathrm{ZG}}$ landmark graphs never have cycles in the initial state and thus we exclude both $\mathrm{LM}^{\mathrm{exh}}$ and $\mathrm{LM}^{\mathrm{ZG}}$ from this analysis.

Lastly, we use a reduced benchmark set for this evaluation. Using TRACK we can never add cycle constraints if the landmark graph in the initial state is acyclic. Hence, we only consider domains where at least one problem has cycles. This leaves us with a total of 870 planning tasks in 26 domains.

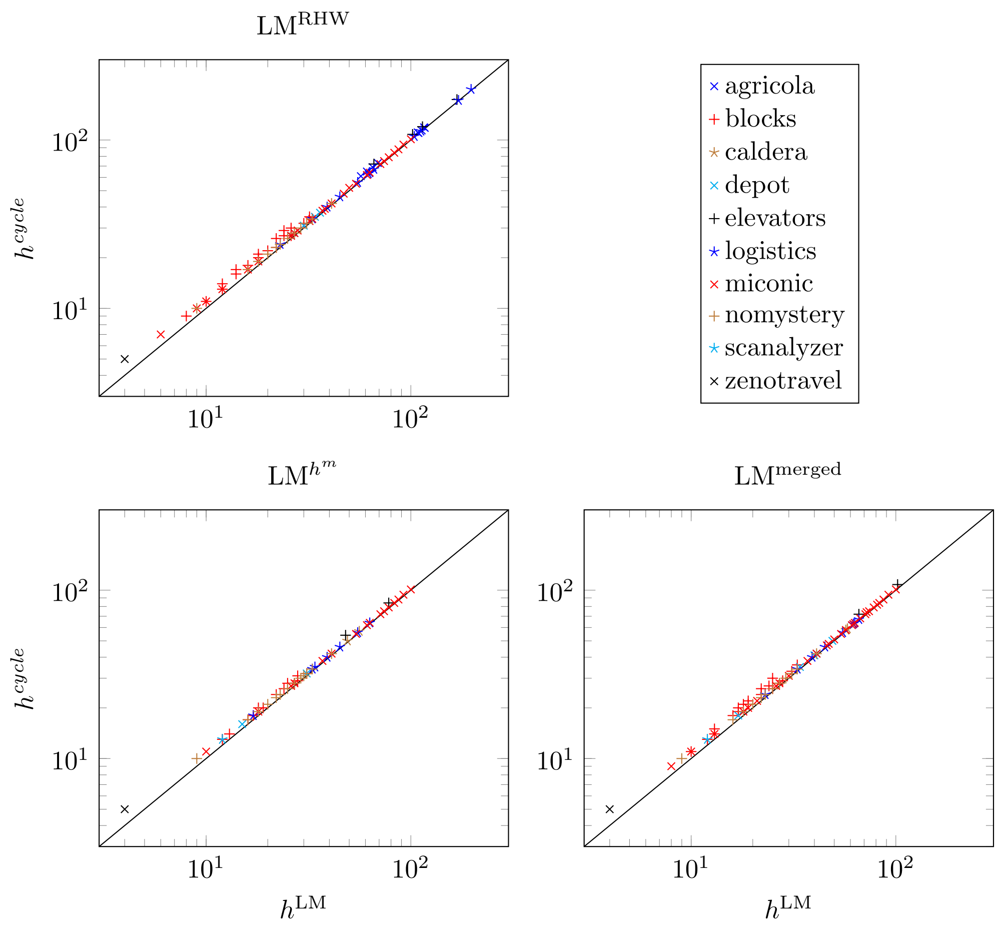


Figure 6.5: The initial state h-values of h^{LM} compared to h^{cycle}. Only problem instances where $h^{\mathrm{LM}}(s_0) \neq h^{cycle}(s_0)$ are displayed. An improvement of the initial h-value can be observed when using the cycle-covering heuristic.

6.3.1 h^{LM} vs. h^{cycle}

First of all, the initial h-values displayed in Figure 6.5 confirm Theorem 1. Each data point corresponds to a planning task where $h^{\mathrm{LM}}(s_0) \neq h^{cycle}(s_0)$. The diagonal line separates the instances where $h^{cycle}(s_0) > h^{\mathrm{LM}}(s_0)$ or rather $h^{cycle}(s_0) < h^{\mathrm{LM}}(s_0)$ (i.e., the upper and lower triangle, respectively). All data points lie above the diagonal which means that h^{cycle} yields higher or equal initial h-values in all problem instances. In turn, this means that h^{cycle} gives more accurate estimates of the cheapest plan cost since both heuristics are admissible. Generally speaking, we expect that a more accurate heuristic leads to fewer expansions due to its better heuristic estimates. Figure 6.6 shows the number of expanded states for each problem with either configuration. Problems which were unsolved by h^{LM} or h^{cycle} lie on the upper or right end of the scale, respectively.

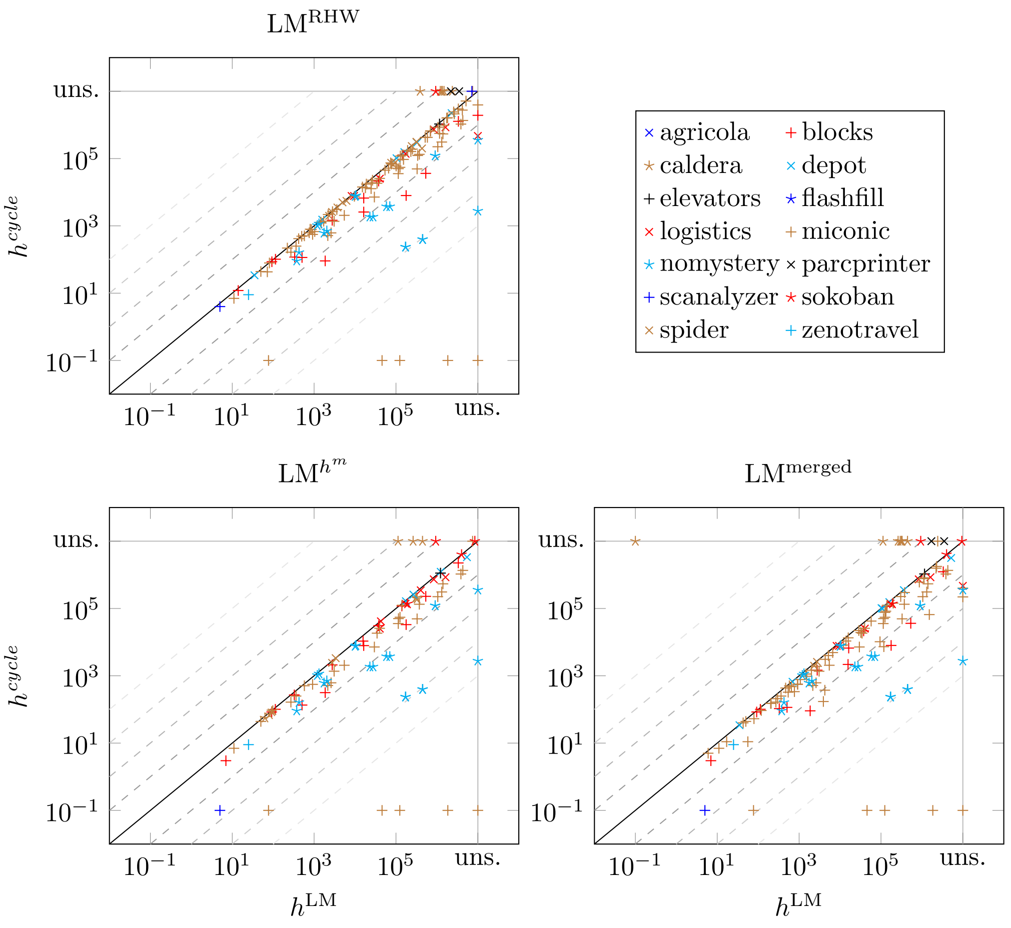


Figure 6.6: Expansions before the last f-layer comparing h^{LM} and h^{cycle}. Only problem instances where the number of expanded states differs are shown. h^{LM} expands more states in all solved problems which indicates that h^{cycle} has better information.

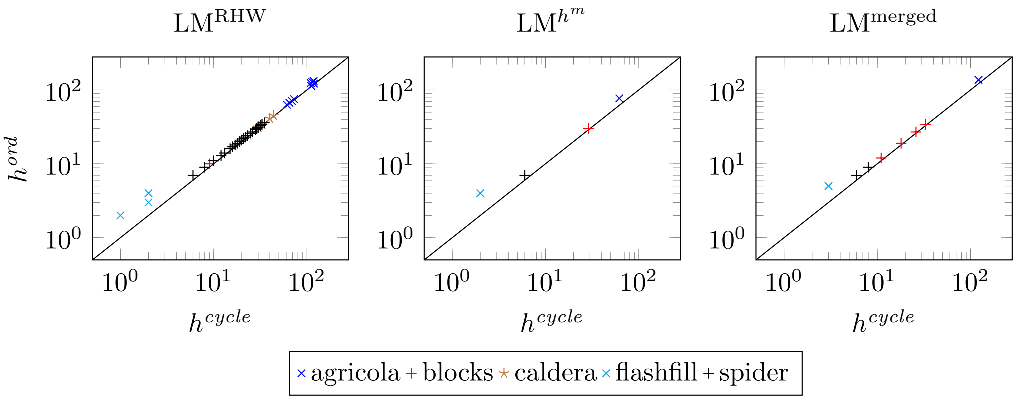


Figure 6.7: The initial state h-values of h^{cycle} compared to h^{ord}. Only problem instances where $h^{cycle}(s_0) \neq h^{ord}(s_0)$ are displayed. The stronger cycle-constraints in the LP lead to improved initial h-values.

$$\begin{tikzcd}
\ell_1 \arrow[r, "\text{gn}"] \arrow[drr, dotted, "\text{r}"] & \ell_2 \arrow[r, "\text{gn}"] \arrow[dr, dotted, "\text{r}"] & \cdots \arrow[r, "\text{gn}"] \arrow[d, dotted, "\text{r}"] & \ell_n \arrow[r, "\text{gn}"] \arrow[dl, dotted, "\text{r}"'] & \ell_{n+1} \\
 & & \ell \arrow[ull, dashed, bend left, "\text{t}"] \arrow[urr, bend right, "\text{gn}"'] & &
\end{tikzcd}$$

Figure 6.8: Structure in landmark graphs which may lead to a large number of cycles. All landmarks $\ell_1, \dots, \ell_n$ are reasonably ordered before ℓ according to Hoffmann, Porteous, and Sebastia (2004). If an ordering $\ell \rightarrow_t \ell_1$ is present, it induces n cycles but only ℓ is considered in the cycle constraints of h^{ord}.

6.3.2 h^{cycle} vs. h^{ord}

Similarly to the previous section, we compare the initial h-values of h^{cycle} to those of h^{ord} in Figure 6.7. The number of tasks influenced by the stronger cycle constraints is fairly small, but not negligible. Especially when using $\mathrm{LM}^{\mathrm{RHW}}$, there are several instances where this makes a difference. One interesting observation is that it seems to affect those domains with many cycles; they are all at the upper end of the scale in Figure 6.1 which means that these domains contain the problems with the most cycles. While we cannot pinpoint the reason for this, we assume a connection to the algorithm approximating reasonable orderings suggested by Hoffmann et al. (2004). Figure 6.8 may help to understand the following explanations.
Consider a set of landmarks $\{\ell, \ell_1, \dots, \ell_{n+1}\}$ such that $\ell_i \rightarrow_{\mathrm{gn}} \ell_{i+1}$ for all $1 \leq i \leq n$ and $\ell \rightarrow_{\mathrm{gn}} \ell_{n+1}$. Then, the algorithm by Hoffmann et al. adds reasonable orderings $\ell_i \rightarrow_{\mathrm{r}} \ell$ for all $1 \leq i \leq n$ because ℓ is in the aftermath of all ℓ_i. If there is also an ordering $\ell \rightarrow_t \ell_1$ of any type t, then each reasonable ordering from above induces a cycle. These are exactly the cases where the stronger constraints may affect the LP solution; all natural orderings are not considered in the cycle constraints of h^{ord}. In particular, this would explain why $\mathrm{LM}^{\mathrm{RHW}}$ is affected the most, because it uses this algorithm to approximate the set of reasonable orderings.
The number of expanded states before the last f-layer with h^{ord} is almost identical to the results of h^{cycle}. Thus, we see no point in visualizing them in a scatter-plot.

6.3.3 Coverage Results

While the results from the previous sections look promising, the change in coverage does not as clearly represent the dominance of h^{cycle} and h^{ord}, respectively. In Table 6.4 we compare the numbers to those of h^{LM} on the benchmark set of domains with cyclical initial states. Here, we also include the numbers of the RECOMP approach to give a broader overview.
The only landmark generation method which consistently improves from h^{LM} to h^{cycle} to h^{ord} is LM^{h^m}. Both $\mathrm{LM}^{\mathrm{RHW}}$ and $\mathrm{LM}^{\mathrm{merged}}$ decrease from h^{LM} to h^{cycle} and then regain some tasks from h^{cycle} to h^{ord}. The numbers for $\mathrm{LM}^{\mathrm{ZG}}$ are only provided for the RECOMP approach because it yields no cycles in the initial state; the results of h^{cycle} and h^{ord} are thus not influenced in this case.
We gain new insights on the comparison between RECOMP and TRACK in these results; when only considering tasks with cycles in the landmark graph of the initial state are considered,

		TRACK			RECOMP		
		LM^{RHW}	LM^{h^m}	$\text{LM}^{\text{merged}}$	LM^{RHW}	LM^{h^m}	LM^{ZG}
Coverage							
	h^{LM}	308	298	355	342	222	350
	h^{cycle}	305	305	351	336	228	350
	h^{ord}	306	308	355	340	231	350
Errors							
Timeout	h^{LM}	556	474	417	522	550	514
	h^{cycle}	554	465	406	447	529	514
	h^{ord}	555	463	406	439	526	514
Out of Memory	h^{LM}	0	92	92	0	92	0
	h^{cycle}	5	94	92	81	107	0
	h^{ord}	3	93	103	85	107	0

Table 6.4: Coverage results of h^{LM}, h^{cycle}, and h^{ord}. Considering cycles only pays off considering LM^{h^m} landmarks. The stronger cycle constraints of h^{ord} yield increased coverage using all methods.

RECOMP results in higher coverage considering LM^{RHW}. Computing the landmark graph in every state with LM^{h^m} however is still too time-consuming. Memory seems to be a limitation more often when using RECOMP especially for LM^{RHW} landmark graphs. Yet, the majority of the tasks are still unsolved due to time restrictions.

6.4 Aiming for Optimality

In cost-optimal planning we aim for a heuristic which approximates h^* as closely as possible. Paul et al. (2017) provide results of their cycle-covering heuristic[12] for logistics which estimates the cost of s_0-plans perfectly. Aiming to generalize their concepts we investigate how closely our domain-independent cycle-covering heuristic approaches h^*. In order to do so we compare the costs of the optimal plans in solved planning tasks with the estimates of h^{ord}. The according results are shown in Figure 6.9. We see some proximity to the diagonal where $h^{cycle}(s_0) = h^*(s_0)$. However, these results must be interpreted with caution since they are somewhat biased towards good initial estimates; we can only plot the results from solved instances because for all others, the true cost (i.e., $h^*(s_0)$) is unknown. One clear observation is that $\text{LM}^{\text{merged}}$ yields the best results concerning the closeness of h^{ord} compared to h^*.

6.5 Comparing to the State of the Art

In this last section of our experimental evaluation we compare our heuristics to related work. At the time of writing, LM-cut is one of the most successful non-portfolio heuristics for cost-optimal planning. Additionally, it also uses landmarks in order to compute its heuristic values. Our analysis is thus centered around a comparison with LM-cut. Since

[12] They call it "integrated cycle heuristic".

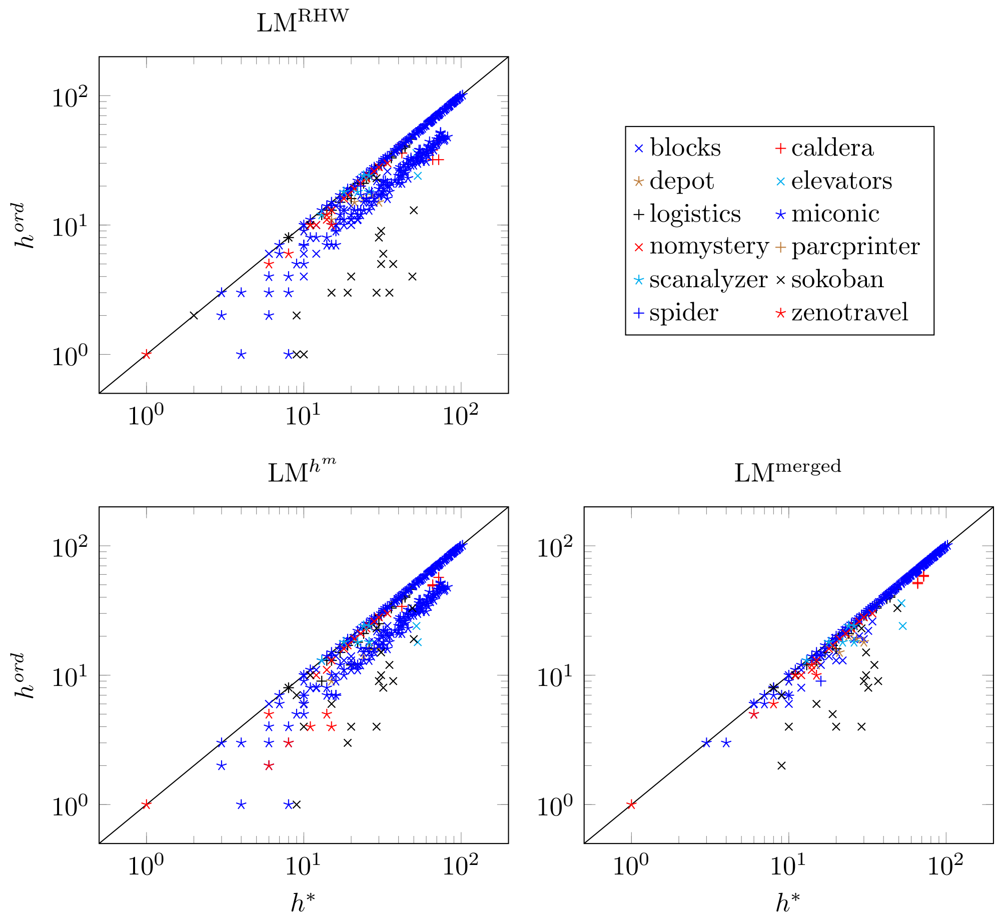


Figure 6.9: Comparison of the initial h-values of h^{ord} and h^*. The h^* values are generated by taking the plan-cost of solved problem instances.

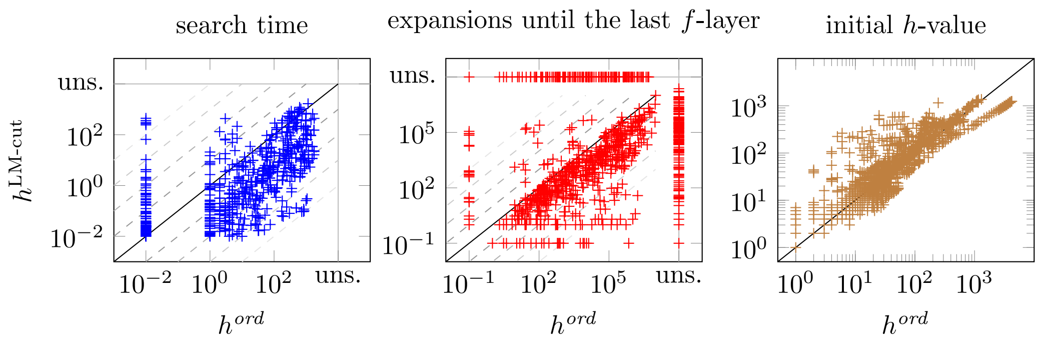


Figure 6.10: We show different aspects of $h^{\text{LM-cut}}$ which is a state of the art landmark heuristic compared to h^{ord}. The ordering-aware cycle-covering heuristic uses LM$^{\text{RHW}}$ landmarks with the TRACK approach which is the configuration resulting in the highest coverage.

LM-cut does not support conditional effects, we use yet another benchmark set consisting of 2937 planning tasks in 111 domains.

The results of this comparison are provided in Table 6.5 and Figure 6.10. They show that our heuristics cannot compete with $h^{\text{LM-cut}}$. All of our configurations solve fewer tasks and

			Coverage	Timeout	Memory-out
		$h^{\text{LM-cut}}$	1'165	1'733	0
TRACK	LM^{RHW}	h^{LM}	880	2'020	0
		h^{cycle}	873	2'024	3
		h^{ord}	874	2'024	2
	LM^{ZG}	h^{LM}	865	2'035	0
		h^{cycle}	860	2'040	0
		h^{ord}	861	2'039	0
	LM^{h^m}	h^{LM}	819	1'529	551
		h^{cycle}	829	1'519	551
		h^{ord}	829	1'521	551
	$\text{LM}^{\text{merged}}$	h^{LM}	826	1'524	548
		h^{cycle}	822	1'518	558
		h^{ord}	821	1'521	556
RECOMP	LM^{RHW}	h^{LM}	751	2'151	0
		h^{cycle}	721	1'949	232
		h^{ord}	730	1'938	234
	LM^{ZG}	h^{LM}	484	2'389	27
		h^{cycle}	484	2'389	27
		h^{ord}	484	2'389	27
	LM^{h^m}	h^{LM}	424	1'924	553
		h^{cycle}	433	1'904	564
		h^{ord}	434	1'904	564

Table 6.5: Comparing the different configurations of our landmarks and cycle-covering heuristics to the state of the art.

they generally need significantly more time to do so. The plots in Figure 6.10 are generated using h^{ord} together with TRACK and LM^{RHW} which is the most successful ordering-aware cycle-covering configuration. In a few tasks it can be observed that the number of expanded states is less for h^{LM}, but these are rare exceptions. The tasks where h^{ord} finds a solution quickly and $h^{\text{LM-cut}}$ does not (i.e., the leftmost instances in the first plot) correspond to unsolvable tasks from the IPC satisficing track. Based on the data of the rightmost plot it is inconclusive which of the methods gives better heuristic estimates. This leads us to the next analysis concerning delete-relaxation in general.

In LM-cut, like all delete-relaxation heuristics, it is never beneficial to apply the same action multiple times; delete-relaxation assumes that facts are available from when they are first achieved onward and they cannot be deleted. In terms of this thesis, this means that a landmark holds forever as soon as it is achieved for the first time. The motivation for the cycle-covering heuristic is different; it takes into account destructive interactions in order to gain additional information about the costs. Hence, h^{ord} has the potential to yield higher heuristic estimates than the perfect delete-relaxation heuristic h^+ if cycles are present in the landmark graphs.

We evaluate this possibility by using the Fast Downward feature to translate planning tasks such that they are delete-free. If solvable, the solution costs of these tasks correspond to the h^+ value of their initial states. In Figure 6.11 we compare these values to those of

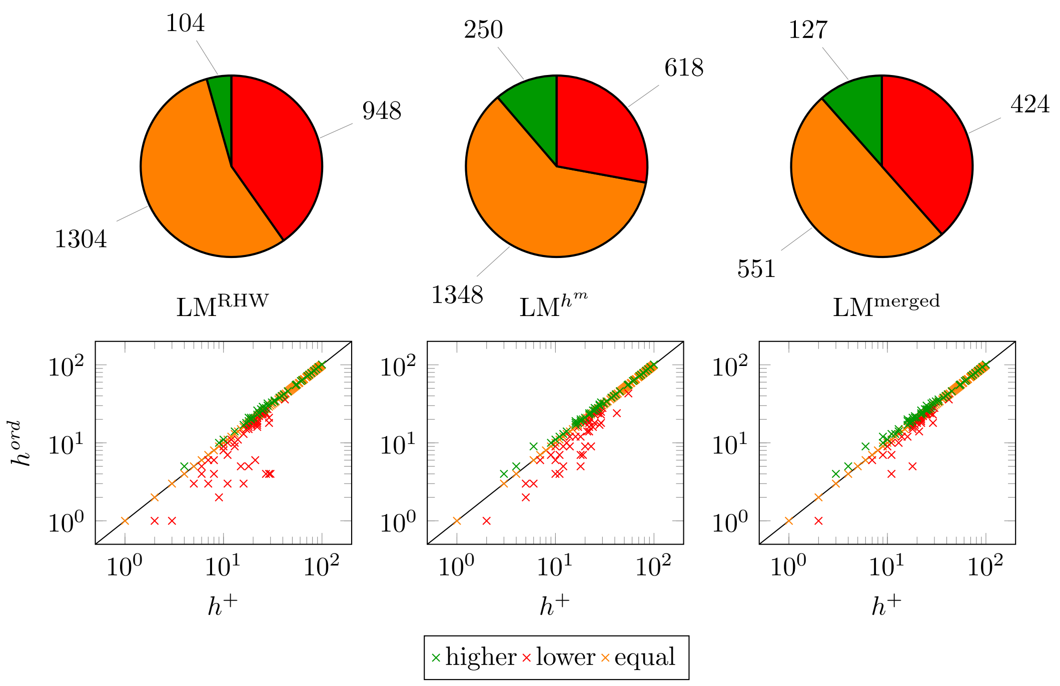


Figure 6.11: The heuristic values of the initial state comparing h^{ord} to the optimal delete-relaxation heuristic h^+. The pie charts show how many problem instances with higher, lower, and equal h-values are found, respectively. In the plots below, these categories are highlighted similarly for the landmark generators LM^{RHW}, LM^{h^m}, and $\text{LM}^{\text{merged}}$.

our ordering-aware cycle-covering heuristic h^{ord}. All instances where $h^{ord}(s_0) > h^+(s_0)$ are displayed in green, $h^{ord}(s_0) < h^+(s_0)$ in red, and $h^{ord}(s_0) = h^+(s_0)$ in orange. Since the scatter-plots themselves might be misleading due to overlapping data points, we also provide how often the different cases occur. The actual numbers, however, are not of great importance; what we take away from this analysis is that the heuristic estimates of h^{ord} can indeed surpass those of h^+.

In the following we give a more theoretical analysis of this topic. According to Bonet and Helmert (2010), the cost of an MHS solution for the complete set of delete-relaxation landmarks is equal to h^+. Hence, the landmark heuristic h^{LM} is equal to h^+ when computed using IPs. Since h^{ord} dominates h^{LM}, theoretically all of its estimates should be at least as good as h^+. In practice, however, this is not the case because of two reasons. On the one hand, the considered sets of landmarks are generally not complete. On the other hand, we only use the LP-relaxation in our experiments which enables computing heuristics in polynomial time. The fact that we find better heuristic estimates in polynomial time despite these reasons is worth mentioning; h^+ is considered a strong heuristic, but NP-hard to compute (Bylander 1994).

Our ordering-aware cycle-covering heuristic h^{ord} is bounded similar to the way $h^{\text{LM-cut}}$ is bounded by h^+; it never accounts for the cost of each operator more than twice. The argument here is similar to what we explain in Section 4.4 for h^{LM}; if the LP variable $\mathsf{Y}_a = 2$ for an action a in each cycle-constraint, then all of them are satisfied under the assumption that all the landmark constraints are already satisfied. Furthermore, the cycle-covering LPs are minimization problems; it is not beneficial to set the variable to something larger than 2 and thus, $\mathsf{Y}_a \leq 2$ for all a. This may become a problem for planning tasks where the same operators must be applied over and over again.[13] However, since this shortcoming also applies to delete-relaxation, the cycle-covering heuristic is still better suited for such planning tasks; this structure most likely yields cyclical dependencies in the landmark graph which may lead to an increased heuristic estimate when considered.

[13] Consider for example the following model of a binary counter: each bit has a corresponding operator which increments it applied. The preconditions of such an operator are that the according bit is 0 and all lower-ordered bits are 1. The effect is that the corresponding bit is set to 1 and all lower-ordered bits are set to 0. For a planning task to count to an arbitrary natural number, every second operator which is applied must be the one increasing the least significant bit.

7 Integer Programming

We have introduced the MHS problem applied to landmarks using IPs in Section 4.2. However, in all previous chapters we use LPs as an approximation because they can be computed in polynomial time. Since it is not impossible to find IP solutions in practice, we present some of our findings when experimenting with LP variables restricted to be integral. We isolate these results from the preceding chapter because they are not as conclusive. Our interpretations are rather speculative and thus we set them apart from the results concerning our generalized cycle-covering heuristic. In this chapter we only use the strongest heuristic h^{ord} in two versions. For reasons of simplicity, we write h^{LP} for the LP version and h^{IP} for the IP version of h^{ord}.

7.1 Experimental Evaluation

Independently of the chosen landmark generator, the results are identical. Thus, we only show the results for $\mathrm{LM}^{\mathrm{RHW}}$. The plots in Figure 7.1 provide an overview of our findings. We do not deem it sensible to distinguish the domains for this comparison.

As expected, more time is required to search for a plan due to the increased hardness of IPs compared to LPs. This is also reflected in the number of unsolved tasks which lie in the top row of the center plot in Figure 7.1. However, all problem instances solved by both methods, do not differ at all; they all lie on the diagonal line which indicates that their value is identical. The same holds for the initial h-values which are identical for LP and IP in all benchmark tasks.

This is unexpected because we assume that the integer solution should increase the initial h-value at least in some problem instances. However, there is not a single instance where the objective value of the IP is higher than its LP counterpart. When analyzing the LP solutions in more detail we find that there are only few problems with non-integer valued solutions. Although our initial assumptions suggested differently, it is actually to be expected that the IP objective values are identical in all cases where the LP solution is integral. There are two explanations for all cases where the LP solution has non-integer values: either there must be an integral solution with the same objective value or it is rounded up to the IP solution. The latter may happen because in practice we apply the ceiling function to LP

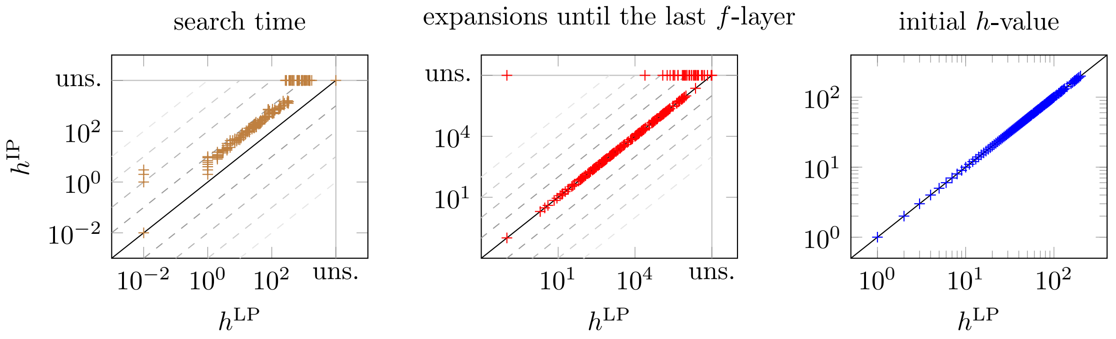


Figure 7.1: Comparison of computing our heuristics as IP rather than LP. The results show no increase in the heuristic values or number of expanded states, whereas running time is affected negatively. Both h^{LP} and h^{IP} correspond to the ordering-aware cycle-covering heuristic h^{ord} using LM$^{\mathrm{RHW}}$ landmarks.

solutions because we know that all costs are natural numbers. (We refer to Pommerening et al. (2014) for an explanation of why this is still admissible.) Still whenever this happens, the difference of the LP objective value is < 1 compared to the IP in all benchmark tasks.

7.2 Total Unimodularity

Questioning our findings about the initial h-values has brought up the hypothesis of having to deal with *totally unimodular* (TU) matrices in our LPs. Since the solution for an LP with a TU matrix must be integral it is possible to find the according IP solution in polynomial time (Hoffman and Kruskal 1956). In our case, the argument starts in the opposite direction: since our solutions for LPs are integral, we suspect the corresponding matrices to be TU. However, the LP matrix must not necessarily be TU if the LP solution is integral. In this section we present an argument that nevertheless supports this case.

Definition 11 (Totally unimodular matrix)**.** *A matrix $A \in \mathbb{R}^{m\times n}$ is called* totally unimodular *(TU) if all minors of A are in $\{-1,0,1\}$.*

In other words, all squared sub-matrices must be TU themselves. Moreover, it is a necessary (but not sufficient) condition that all $a_{ij} \in \{-1,0,1\}$ where a_{ij} is the entry of A in row i and column j.
We now explain the potential of our cycle-covering constraints to yield TU matrices. In order to do so, we consider the landmark graph $\mathcal{G}$ of an arbitrary state s in a planning task $\mathcal{T}$ for the remainder of this chapter.
Generally, the constraint set of the cycle-covering heuristic h^{cycle} does not allow for the assumption of TU. One example of this is if $\mathcal{G}$ contains a cycle c with overlapping landmarks. Let $\mathcal{L} = \{\ell_1,\dots,\ell_n\} \subset \mathcal{L}(c)$ be a set of $n \geq 2$ landmarks so that $a \in \bigcap_{\ell\in\mathcal{L}} \ell$ and $a \notin \bigcup_{\ell\in\mathcal{L}(c)\setminus\mathcal{L}} \ell$. The cycle constraint according to Equation (5.3) can be written as follows:

$$n\mathsf{Y}_a + \sum_{\ell\in\mathcal{L}(c)} \sum_{a'\in\ell\setminus\{a\}} \mathsf{Y}_{a'} \geq |\mathcal{L}(c)| + 1. \tag{7.1}$$

Hence, the LP matrix has an entry $n \notin \{-1, 0, 1\}$ and consequently it is not TU. However, if G has no such cycles, all entries of the matrix are either 0 or 1.

Proposition 2. *Let $\mathcal{G}$ be the landmark graph of an arbitrary state s in a planning task and let $\mathcal{C}$ be the set of elementary cycles in $\mathcal{G}$.*
All entries of the LP matrix A resulting from the constraint sets $C^{\mathrm{LM}}(s)$ and $C^{cycle}(s)$ are either 0 or 1 if the following condition holds for all cycles $c \in \mathcal{C}$:

$$\ell \cap \ell' = \emptyset \quad \textit{for all } \ell, \ell' \in \mathcal{L}(c), \ell \neq \ell'.$$

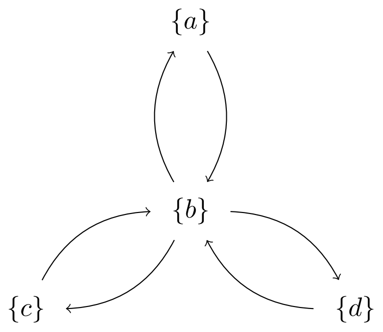
Proof. We distinguish between constraints in $C^{\mathrm{LM}}(s)$ and constraints in $C^{cycle}(s)$.

- Let $\ell \in \mathcal{L}(s)$ be a landmark for state s in a planning task. The corresponding constraint in $C^{\mathrm{LM}}(s)$ has the following properties: the coefficient for Y_a is 0 if $a \notin \ell$ and it is 1 if $a \in \ell$. Thus, the proposition holds for all landmark constraints.
- Now let $c \in C$ be a cycle in the landmark graph of state s. The coefficient of Y_a corresponds to the amount of landmarks $\ell \in \mathcal{L}(c)$ in which a occurs. Since $\ell \cap \ell' = \emptyset$ for all pairs $\langle \ell, \ell' \rangle \in \mathcal{L}(c) \times \mathcal{L}(c)$ where $\ell \neq \ell'$ we infer that a appears in at most one landmark. The according entry in A must thus either be 0 or 1.

□

Assume the matrix A of a state in $\mathcal{T}$ satisfies the above conditions and has only 0 and 1 entries. If it is possible to rearrange A so that the 1s in each row occur consecutively, this is a sufficient condition for A to be TU (Ghouila-Houri 1962). This is called the consecutive-ones property. We suggest an idea to decide whether this property is feasible further down. Rearranging the rows so that the columns have the consecutive-ones property is also sufficient because a matrix is TU if its transposed is TU. However, this is generally impossible if there are cycles consisting of three or more landmarks; the rows of the corresponding landmark constraints should all be neighboring the row of such a cycle but there are only two such spots available.
We now present our idea to achieve the consecutive-ones property for the columns. Consider a landmark graph $\mathcal{G} = \langle \mathcal{L}, \mathcal{O} \rangle$ and let $\mathcal{C}$ be the set of cycles induced by $\mathcal{O}$. The cycles in $\mathcal{C}$ do not overlap and hence all entries of the cycle-covering LP are 0 or 1 according to Proposition 2. For this idea, we interpret each action or rather its corresponding column in the LP matrix as an individual of a community. These individuals participate in *groups* according to the following rules:

- Each landmark $\ell \in \mathcal{L}$ induces a group g such that all actions $a \in \ell$ are in this group: $g = \ell$.
- Each cycle $c \in \mathcal{C}$ induces a group g' such that all actions $a \in \bigcup_{\ell \in \mathcal{L}(c)} \ell$ are in this group: $g' = \bigcup_{\ell in \mathcal{L}(c)} \ell$.
- Two groups g_1 and g_2 are *opposing* if $g_1 \setminus g_2 \neq \emptyset$ and $g_2 \setminus g_1 \neq \emptyset$. Otherwise, they are either identical or one is a sub-group of the other (i.e., $g_1 \subset g_2$ or $g_2 \subset g_1$).



(a) Landmark graph

	Y_a	Y_b	Y_c	Y_d	$\geq$
ℓ_a	1	0	0	0	1
ℓ_b	0	1	0	0	1
ℓ_c	0	0	1	0	1
ℓ_d	0	0	0	1	1
c_{ab}	1	1	0	0	3
c_{bc}	0	1	1	0	3
c_{bd}	0	1	0	1	3

(b) LP matrix

Figure 7.2: Example of a landmark graph for which the according LP matrix is TU but does not have the consecutive-ones property.

The rows of an LP matrix A of $\mathcal{G}$ correspond to the different groups present in the entire community. Each column of A corresponds to an action a and has a 1-entry for all groups in which a participates. The columns which participate in the same group must occur consecutively in A to fulfill the consecutive-ones property. If two groups are opposing, this means that they restrict the order in which groups must be aligned in the matrix. We propose the following conjecture for which an in-depth analysis exceeds the scope of this thesis.

Conjecture 1. *The columns of a matrix $A \in \{0,1\}^{m \times n}$ can be rearranged to have the consecutive-ones property if*

- *each group of A has at most two opposing groups, and*
- *the strings of opposing groups are acyclic.*

Our idea to construct a formal proof is to order the groups so that each of them is surrounded by its opposing groups; elements which occur in both groups build the transition between the groups. The restriction to two oppositions ensures that this is possible and also does not interfere with sub-groups. We assume that whether or not this criterion is satisfied highly depends on the domain of a problem.
This is by far not the only situation which may yield TU matrices for our LPs. For example, consider a landmark graph as displayed in Figure 7.2a. Since landmark $\{b\}$ is involved in three cycles, b occurs in three opposing groups; its neighbors in the LP matrix should be a, c, and d. The table in Figure 7.2b represents the according LP matrix. It goes without saying that making the Y_d column a neighbor of Y_b destroys the consecutive-ones property of c_{ab} or c_{ad}. Nonetheless, the matrix is TU and the optimal (integral) solution in this case is $\mathsf{Y}_a = \mathsf{Y}_c = \mathsf{Y}_d = 1$ and $\mathsf{Y}_b = 2$.

$$\begin{tikzcd}
 & \{x\} \arrow[dl, bend left=10] \arrow[dr, bend left=30] & \\
\{y\} \arrow[ur, bend left=40] \arrow[rr, bend left=15] & & \{z\} \arrow[ul, bend left=10] \arrow[ll, bend left=30]
\end{tikzcd}$$

(a) Landmark graph

	Y_x	Y_y	Y_z	$\geq$
ℓ_x	1	0	0	1
ℓ_y	0	1	0	1
ℓ_z	0	0	1	1
c_{xy}	1	1	0	3
c_{xz}	1	0	1	3
c_{yz}	0	1	1	3
c_{xyz}	1	1	1	4
c_{xzy}	1	1	1	4

(b)

Figure 7.3: A landmark graph consisting of 3 landmarks and 6 orderings, making all landmarks dependent on each other.

7.3 IP Can Influence the Heuristic Value

In the previous section, we explore why we only find identical objective values for LP and IP. However, if a matrix is not TU it is possible to show that different objective values are found. The following example is based on the landmark graph displayed in Figure 7.3a. In Equation (7.2) we present a squared sub-matrix of Figure 7.3b which represents the constraints for all cycles of exactly two landmarks.

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \Rightarrow \det(A) = -2 \tag{7.2}$$

As the minor of A is not in $\{-1, 0, 1\}$, the complete LP matrix cannot be TU although its entries are only 0 and 1. Indeed, the LP solution of this problem assigns $\mathsf{Y}_x = \mathsf{Y}_y = \mathsf{Y}_z = 1.5$ which are all non-integers. Solving this problem as an IP yields that two out of the three actions must be applied twice and the third only once; hence, the LP objective value is 4.5 and for the IP it is 5.

Based on this analysis we construct a logistics problem as presented in Figure 7.4. Arrows are added to better visualize the cycles induced by the packages. The structure of Figure 7.3a appears twice in this problem; once for locations B, C, and D as well as once for locations E, F, and G. In the landmark graph, this triangular structure can be found in independent sub-graphs. The involved disjunctive action landmarks are the unions of *move*-operators to each location. As an effect, the LP solution will find that each location must be driven to 1.5 times. Consequently, the *move*-operators contribute a total value of 9 to the heuristic estimate. If we use an IP instead, we find that in each of the location triangles, two out of the three must be driven to twice and the third only once. The *move*-operators thus contribute 10 to the heuristic, which improves the total heuristic value compared to the LP solution. We implemented the according planning task in PDDL and can confirm that the h-value in the initial state is different when comparing the LP with the IP computation.

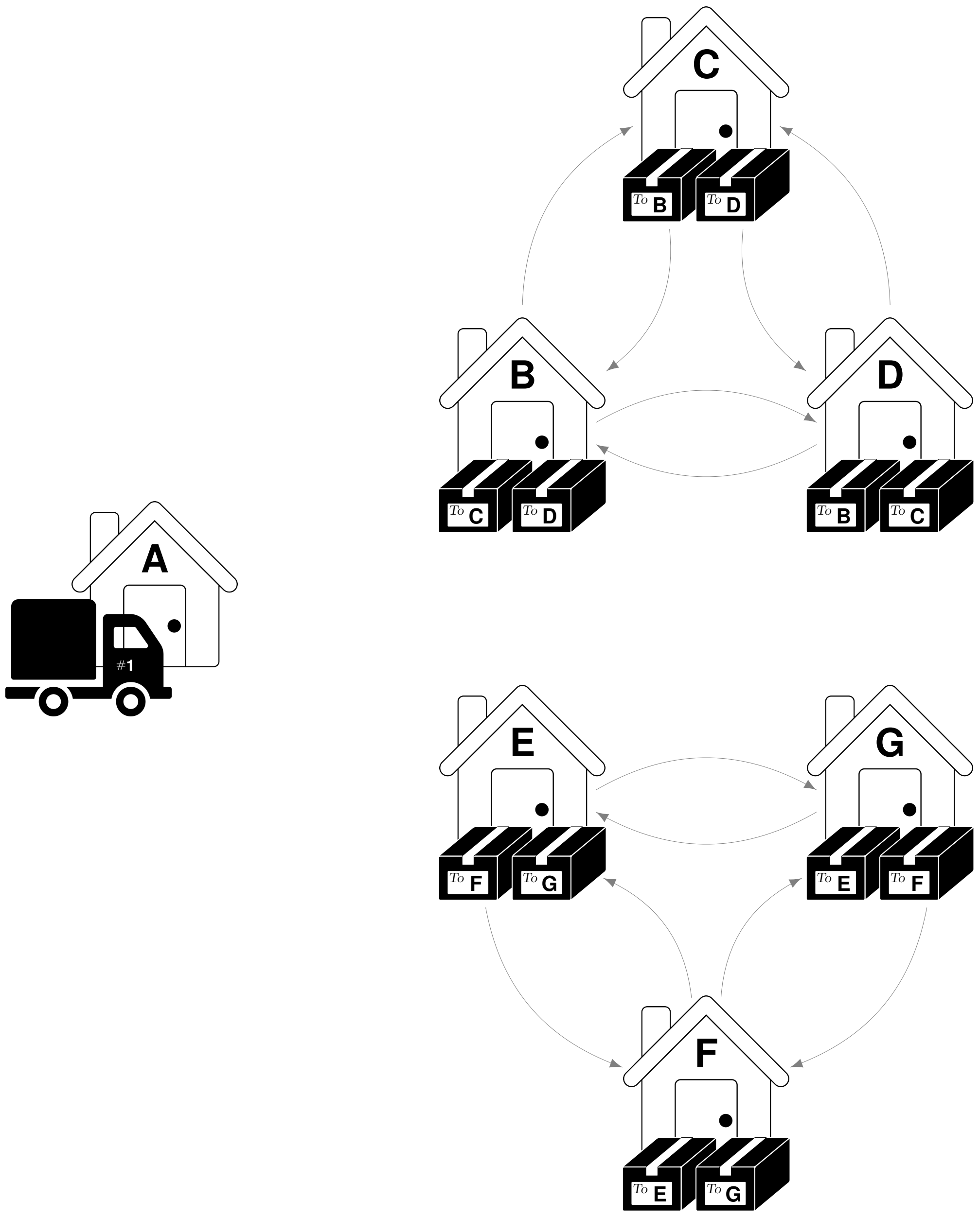


Figure 7.4: Example of a planning task where the solutions of LP and IP differ when using cycle-covering constraints. Arrows indicate the routes of the packages which yield the triangular structure also found in Figure 7.3a twice.

8 Conclusion

In this thesis, we use LPs to compute a landmark heuristic. Each landmark induces an LP constraint which states that this landmark must be part of every plan. The landmarks for this heuristic are approximated by landmark generators from recent literature. In addition, these landmark generators provide ordering constraints between the landmarks. Such landmark orderings may induce cycles that correspond to deadlocks in the according planning task. Deadlocks can only be resolved by applying an action that does not directly contribute to the goal; in other words, one landmark from each cycle must hold twice along all plans. This additional information can be used to enhance the landmark heuristic. We provide the theory for a generalized cycle-covering heuristic h^{cycle} inspired by Paul and Helmert (2016) and Paul et al. (2017). It dominates the landmark heuristic because it adds additional LP constraints which means that the estimated h-values cannot be worse. This dominance is also confirmed by an experimental evaluation.
Landmark orderings can be of different types; natural orderings must hold along all plans while there might be plans which do not obey reasonable orderings. By definition, natural orderings cannot induce cycles in the landmark graph of a solvable planning task. Thus, all cycles induced by landmark orderings consist of at least one reasonable ordering. Based on the definition of natural orderings, we infer that only landmarks with incoming reasonable orderings are candidates to resolve the deadlocks. The LP constraints for all cycles can be strengthened by this information and brings a stronger cycle-covering heuristic to light; we show that the ordering-aware cycle-covering heuristic h^{ord} dominates h^{cycle}.

8.1 Potential and Flaws of h^{ord}

This thesis mainly serves as a proof of concept; cycle-covering is a way to increase the information extracted from landmark graphs in heuristic search for cost-optimal planning. We have shown the theoretical dominance and provided experiments where this dominance is confirmed to some extent. At the moment, this is not represented in the problem coverage. However, we still see high potential in this information and list some of our reasons here.

8.1.1 Balancing Search Time and Heuristic Accuracy

We experiment with two approaches which we call RECOMP and TRACK. They describe whether the landmarks are recomputed in every state encountered during search or if they are computed once in the beginning and tracked along paths. While recomputing provides better guidance for the search algorithm, its running time highly depends on the time required to compute landmark graphs. This becomes an issue for most of the examined landmark generators.

We show that cycles are present in various planning domains. Finding all elementary cycles in a graph can be done efficiently using Johnson's Algorithm. Considering them as LP constraints in addition to the landmarks MHS increases the heuristic accuracy. However, solving an LP in every evaluated state also impairs the running time. Our heuristics need significantly more search time than for example LM-cut which is a state of the art landmark heuristic.

8.1.2 Surpassing Delete-Relaxation

With cycle-covering, we have the potential to exceed estimates from delete-relaxation heuristics. These heuristics apply each operator at most once because their delete-effects are ignored. Hence, the accumulated costs considers each operator at most once. In this setting, a landmark should never be necessary more than once because all its implications hold after achieving it for the first time. The cycle covering heuristic dominates the optimal delete-relaxation heuristic h^+ under the assumption of a complete set of delete-relaxation landmarks. In our results we show that already good approximations of these landmarks can yield heuristic estimates higher than h^+.

However, our heuristic is bounded similar to how delete-relaxation heuristics are bounded by h^+; while in delete relaxation it is never beneficial to apply the same operator more than once, with h^{ord} each operator will never be counted more than twice. The reason for this lies in the LP constraints which are satisfied when the according action count variables are set to 2. Perhaps analyzing the landmark graphs of planning tasks where operators are necessary more than twice can lead to interesting observations; common structural characteristics could be used to define additional LP constraints in a similar way to the cyclical dependencies used for cycle-covering.

8.1.3 Operator-Counting Framework

Our implementation of the cycle-covering heuristic is in accordance with the operator-counting framework. Thus, it may be combined with other operator-counting heuristics to create a stronger heuristic. For example, combining it with a network-flow heuristic may resolve the shortcoming outlined in the preceding section; repeated delete-effects when achieving different landmarks may be recognized by such an approach. In this fashion, each of the combined heuristics can compensate for the weaknesses of its counterparts. We leave it to future work to follow this path and try out synergistic combinations including the cycle-covering heuristic.

8.2 Future Work

The results presented in Chapter 6 support the claim that considering cycles in landmark graphs provides valuable additional information compared to only the landmarks. However, our research has many open ends which might be explored further in the future. In this section, we point out the discoveries which we found most interesting for further investigation.

8.2.1 Recomputing the Landmarks Graphs

We have shown that there are many planning domains where cyclical dependencies between landmarks occur. The number of domains with cycles in the landmark graphs of the initial state is significantly lower. However, recomputing the landmark graph in every encountered state is often too time-consuming using the considered landmark generators. Our alternative approach computes the landmark graph once in the initial state and tracks which landmarks are achieved along the searched paths. Hence, the resulting heuristic is path-dependent and cannot find new information (i.e., landmarks and cycles) when exploring the state-space. This especially leads to never evaluating a state to be farther from the goal than the initial state. The number of expanded states is likely rather high; this follows naturally because some landmarks might be achieved even though search moves away from the goal.
This is undesirable behavior and we would prefer the heuristic to recompute the landmark graph in every state. Maybe, the two approaches can be combined in one way or another to get the best from both worlds. For example, one could recompute the landmark graph not in every state, but regularly according to some criterion. The issue with this approach is that search expands different areas of the state space simultaneously; the landmark graphs for two states evaluated right after one another might differ significantly and thus there is not a commonly shared landmark graph for all states like the one of the initial state. This circumstance may require storing many landmark graphs in order not to have to recompute them again and again. Memory restrictions could bring this approach to a halt very fast.
A different idea may be to make the computation of a landmark graph more efficient. Some effort might be spared by not starting from scratch, but updating a previous version. Nodes are removed based on the action sequence from the last stored instance to the current state. Additionally, new landmarks may be added to the graph according to a criterion (which may or may not exist). This idea is of course again dependent on storing the landmark graph from time to time in order to keep these updating procedures tractable. The only way to avoid this is to recompute it in every encountered state. We found this to be too inefficient with the available landmark generators, which might just be unsuited for our purposes. Hypothetically, there might be faster methods which suffice for the needs of cycle-covering and enable recomputing the landmark graph in every state.

8.2.2 Landmark Generation

We limited our research to some extent by solely considering previously suggested landmark generation methods. Cyclical dependencies between landmarks are described as problematic in the studied literature. Thus, cycles in the landmark graphs are discarded by these methods; maybe they are even designed in a way that avoids encountering too many cy-

cles. While our experiments suggest that this is not the case, it could still be worthwhile investigating new approaches to generate landmarks. Or maybe the existing methods can be enhanced by new types of landmark orderings.
For example, Hoffmann et al. (2004) suggest an ordering relation similar to greedy-necessary orderings. As a reminder, greedy-necessary orderings denote conditions for making a fact true for the first time. Another approach could be to investigate conditions for making a fact true for the last time. In some sense, this also relates to Paul et al. (2017) who use a landmark ordering relation different from the ones used in this thesis. Consider two disjunctive action landmarks ℓ and ℓ' and an ordering $\ell \to \ell'$. In Paul et al.'s definition of landmark orderings this denotes that the first action from ℓ occurs before the last action from ℓ' in every plan.
The potential of cycles in the landmark graph increases with additional orderings between the landmarks. Furthermore, additional cycles can only improve the heuristic estimates of a cycle-covering heuristic. However, the complexity of the according LPs will also increase, and the required memory to store all these cycles might become a problem.

8.2.3 Reduce Excessive Memory Usage

In our experiments we observed that most of the unsolved tasks fail because of time-restrictions. However, the number of out-of-memory errors increases when using h^{cycle} or h^{ord} compared to h^{LM}. This is not surprising if we consult the numbers of cycles found in some encountered states; the maximal number of cycles found in any state is nearly 17 million according to Table 6.2. On the one hand, it might exceed the memory limits to store the LP constraints for all these cycles. On the other hand, it should not even be necessary to store them all.
In Figure 6.8, a single ordering $\ell \to_t \ell_1$ yields a lot of cycles. Translating all of them into ordering-aware cycle constraints for the LP results n times the identical constraint; all landmarks with greedy-necessary incoming orderings are discarded and what remains (i.e., only ℓ and maybe ℓ_1 if t is not reasonable) is identical for all cycles. This indicates that $(n-1)$ of these constraints are redundant.
This observation can be generalized further by going back to the example landmark graph in Figure 7.3a. The constraints for cycles c_{xyz} and c_{xzy} are redundant because they are implied by the other cycle and landmark constraints; adding the constraints

$$\begin{aligned} c_{xy}: &\quad \mathsf{Y}_x + \mathsf{Y}_y \qquad \geq 3 \\ \ell_z: &\quad \qquad\qquad \mathsf{Y}_z \geq 1 \end{aligned} \tag{8.1}$$

results in

$$\mathsf{Y}_x + \mathsf{Y}_y + \mathsf{Y}_z \geq 4 \tag{8.2}$$

which corresponds to the constraints of c_{xyz} and c_{xzy}. The constraints for these cycles can thus be discarded because they do not influence the solution; all solutions which satisfy c_{xy} and ℓ_z automatically also apply to c_{xyz} and c_{xzy}. In general, this deduction holds for any two cycles c and c' where $\mathcal{L}(c) \subseteq \mathcal{L}(c')$: it is redundant to add the constraint for c' on top of the constraint for c. We deduce that filtering out redundant cycles before setting up the LP may prevent our algorithm from running out of memory.

In our experiments we did not do this because CPLEX supports such optimizations automatically as pre-solve step of the LP procedure. We did not deem it sensible to compete with software designed for optimization by handling these redundancies in our implementation. Given the results, however, it seems sensible to consider this observation to reduce the number of LP constraints. Maybe the algorithm for finding cycles in the landmark graphs (i.e., Johnson's Algorithm) can be optimized in the sense that redundant cycles are discarded right away.

8.2.4 Integer Programming

In Chapter 7 we discuss the use of IPs instead of LPs in order to restrict the number of action applications to integer values. It would be interesting to learn why so many LPs have integral solutions. Currently, we cannot explain why IPs, if a solution is found despite their hardness, do not yield higher estimates than LPs. Analyzing the LP matrices concerning the characteristic of total unimodularity may provide important insights. If there are other reasons why the LP and IP solutions are equal so often, these might also be of great interest.

8.2.5 Temporal Interpretation of Landmark Orderings

Our consideration of landmark orderings is based on the structures they induce in the landmark graph. When first proposed, however, they were used to split the underlying planning task into smaller problems that are easier to solve (Hoffmann et al. 2004). This approach temporally interprets these relations; landmarks that are ordered before other landmarks may be achieved earlier in a plan. However, this is sub-optimal because the used set of orderings is incomplete.

In this context, our suggestion is a new approach for constructing LP constraints based on the landmark graph. It renders finding cycles redundant and adds constraints to the LP for all landmark orderings as well as the landmarks themselves. Additional LP variables that are ignored in the objective function serve the purpose of representing the temporal aspect of landmark orderings. For example, the constraint for a natural ordering $\ell_1 \rightarrow_{\mathrm{n}} \ell_2$ could be constructed as follows.

$$T^F_{\ell_1} + 1 \leq T^F_{\ell_2} \tag{8.3}$$

where the T^F_ℓ denotes the *time* at which ℓ was *first* achieved. A reasonable ordering $\ell_3 \rightarrow_{\mathrm{r}} \ell_4$ with $\ell_3 \cup \ell_4 = \emptyset$ can be formalized as

$$T^F_{\ell_3} + 1 \leq T^L_{\ell_4} \tag{8.4}$$

where T^L_ℓ is the time step in which ℓ is resolved for the *last* time. Then, if $T^F_\ell < T^L_\ell$ we can deduce that ℓ must be true at least twice in all plans. The LP constraint for ℓ should thus be conditional in order to distinguish this case from the case where $T^F_\ell = T^L_\ell$.

Although intrigued by this idea, we refrained from looking deeper into this as part of the present thesis. However, the approach would be more open to be extended with new ordering types compared to the cycle-covering heuristic; similarly to reasonable orderings in h^{ord}, they might imply the possibility of discoveries concerning their role in cycles. This may change the overall construction of cycle-constraints, whereas in the temporal approach they simply induce additional constraints.

8.3 Full Circle

This thesis started by declaring it necessary to take a step back in order to move forward from time to time. Indeed we have advanced in the understanding of cycle-covering for landmark heuristics by doing so. In this context, taking a step back corresponds to repeatedly achieving the same sub-goals which is required in order to solve certain planning tasks. Both our theoretical and practical results look promising. So now, we get to look forward to the next steps — in whichever direction they may go.

Bibliography

Aspvall, Bengt and Richard E. Stone (1980). “Khachiyan’s Linear Programming Algorithm”. In: *Journal of Algorithms* 1.1, pp. 1–13.

Bäckström, Christer and Bernhard Nebel (1995). “Complexity Results for SAS$^+$ Planning”. In: *Computational Intelligence* 11.4, pp. 625–655.

Bonet, Blai and Julio Castillo (2011). “A Complete Algorithm for Generating Landmarks”. In: *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*, pp. 315–318.

Bonet, Blai and Malte Helmert (2010). “Strengthening Landmark Heuristics via Hitting Sets”. In: *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pp. 329–334.

Bylander, Tom (1991). “Complexity Results for Planning.” In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI 1991)*, pp. 274–279.

Bylander, Tom (1994). “The Computational Complexity of Propositional STRIPS Planning”. In: *Artificial Intelligence* 69.1-2, pp. 165–204.

Fikes, Richard E. and Nils J. Nilsson (1971). “STRIPS: A new approach to the application of theorem proving to problem solving”. In: *Artificial intelligence* 2.3–4, pp. 189–208.

Ghouila-Houri, Alain (1962). “Caractérisation des matrices totalement unimodulaires”. In: *Comptes Redus Hebdomadaires des Séances de l’Académie des Sciences* 254, pp. 1192–1194.

Gupta, Naresh and Dana S. Nau (1992). “On the Complexity of Blocks-World Planning”. In: *Artificial Intelligence* 56.2–3, pp. 223–254.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Helmert, Malte (2004). “A Planning Heuristic Based on Causal Graph Analysis”. In: *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pp. 161–170.

Helmert, Malte (2006). “The Fast Downward Planning System”. In: *Journal of Artificial Intelligence Research* 26, pp. 191–246.

Helmert, Malte (2009). “Concise Finite-Domain Representations for PDDL Planning Tasks”. In: *Artificial Intelligence* 173.5–6, pp. 503–535.

Helmert, Malte and Carmel Domshlak (2009). “Landmarks, Critical Paths and Abstractions: What’s the Difference Anyway?” In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pp. 162–169.

Hoffman, Alan J. and Joseph B. Kruskal (1956). “Integral Boundary Points of Convex Polyhedra, in HW Kuhn and AW Tucker (Eds.) Linear Inequalities and Related Systems”. In: *Annals of Mathematical Studies* 38, pp. 223–246.

Hoffmann, Jörg, Julie Porteous, and Laura Sebastia (2004). “Ordered Landmarks in Planning”. In: *Journal of Artificial Intelligence Research* 22, pp. 215–278.

IBM (2019). *IBM® ILOG® CPLEX® Optimization Studio, v12.9.0.* URL: https://www.ibm.com/products/ilog-cplex-optimization-studio.

Johnson, Donald B. (1975). “Finding all the Elementary Circuits of a Directed Graph”. In: *SIAM Journal on Computing* 4.1, pp. 77–84.

Karp, Richard M. (1972). “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations*, pp. 85–103.

Karpas, Erez and Carmel Domshlak (2009). “Cost-Optimal Planning with Landmarks”. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pp. 1728–1733.

Keyder, Emil, Silvia Richter, and Malte Helmert (2010). “Sound and Complete Landmarks for And/Or Graphs”. In: *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pp. 335–340.

Khachiyan, Leonid Genrikhovich (1979). “A Polynomial Algorithm in Linear Programming”. In: *Doklady Akademii Nauk*, pp. 1093–1096.

Koehler, Jana and Jörg Hoffmann (2000). “On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm”. In: *Journal of Artificial Intelligence Research* 12, pp. 338–386.

McDermott, Drew M. (2000). “The 1998 AI Planning Systems Competition”. In: *AI Magazine* 21.2, pp. 35–55.

Paul, Gerald and Malte Helmert (2016). “Optimal Solitaire Game Solutions using A* Search and Deadlock Analysis”. In: *Proceedings of the 9th Annual Symposium on Combinatorial Search (SoCS 2016)*, pp. 135–136.

Paul, Gerald, Gabriele Röger, Thomas Keller, and Malte Helmert (2017). “Optimal Solutions to Large Logistics Planning Domain Problems”. In: *Proceedings of the 10th Annual Symposium on Combinatorial Search (SoCS 2017)*, pp. 73–81.

Pommerening, Florian, Gabriele Röger, Malte Helmert, and Blai Bonet (2014). “LP-based Heuristics for Cost-Optimal Planning”. In: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*, pp. 226–234.

Porteous, Julie, Laura Sebastia, and Jörg Hoffmann (2001). “On the Extraction, Ordering, and Usage of Landmarks in Planning”. In: *Proceedings of the 6th European Conference on Planning (ECP 2001)*, pp. 37–48.

Richter, Silvia (2010). “Landmark-Based Heuristics and Search Control for Automated Planning”. PhD thesis. Griffith University, Brisbane, Australia.

Richter, Silvia, Malte Helmert, and Matthias Westphal (2008). “Landmarks Revisited”. In: *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, pp. 975–982.

Richter, Silvia and Matthias Westphal (2010). “The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks”. In: *Journal of Artificial Intelligence Research* 39, pp. 127–177.

Seipp, Jendrik, Florian Pommerening, Silvan Sievers, and Malte Helmert (2017). *Downward Lab*. DOI: 10.5281/zenodo.790461. URL: https://doi.org/10.5281/zenodo.790461.

Zhu, Lin and Robert Givan (2003). “Landmark Extraction via Planning Graph Propagation”. In: *ICAPS 2003 Doctoral Consortium*, pp. 156–160.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

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

Author — Autor
Clemens Büchner

Matriculation number — Matrikelnummer
2015-059-603

Title of work — Titel der Arbeit
Generalization of Cycle-Covering Heuristics

Type of work — Typ der Arbeit
Master's Thesis

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

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

Basel, April 20, 2020

[illegible]

Signature — Unterschrift