# Hitting Set Heuristics for Overlapping Landmarks in Satisficing Planning

**Clemens Büchner, Remo Christen, Salomé Eriksson, Thomas Keller**

University of Basel, Switzerland
{clemens.buechner, remo.christen, salome.eriksson, tho.keller}@unibas.ch

## Abstract

Landmarks are a core component of LAMA, a state-of-the-art satisficing planning system based on heuristic search. It uses landmarks to estimate the goal distance by summing up the costs of their cheapest achievers. This procedure ignores synergies between different landmarks: The cost of an action is counted multiple times if it is the cheapest achiever of several landmarks. Common admissible landmark heuristics tackle this problem by underapproximating the cost of a minimum hitting set of the landmark achievers. We suggest to overapproximate it by computing suboptimal hitting sets instead if admissibility is not a requirement. As our heuristics consider synergies between landmarks, we further propose to relax certain restrictions LAMA imposes on the number of landmarks and synergies between them. Our experimental evaluation shows a reasonable increase in the number of landmarks that leads to better guidance when used with our new heuristics. Equipped with our heuristics, a LAMA-like configuration beats LAMA both in terms of problem coverage and plan quality.

## Introduction

Classical planning aims to find a sequence of actions leading from an initial state to a goal state in a deterministic transition system (Ghallab, Nau, and Traverso 2004). To this day, LAMA (Richter and Westphal 2010) is a competitive approach to finding suboptimal solutions for classical planning problems. This was recently demonstrated at the International Planning Competition (IPC) 2023, where LAMA was used as a baseline; it beat all competitors in the *agile* track (finding any solution quickly) and almost made the podium in the *satisficing* track (finding cheap solutions).

LAMA approaches planning as a series of explicit *heuristic searches*, guided by the $h^{\text{FF}}$ and $h^{\text{sum}}$ heuristics. The former sums up the costs of best achievers of relevant atoms in the delete-relaxation to estimate the goal distance (Hoffmann and Nebel 2001), the latter sums up the costs of the cheapest achiever of each *landmark* (Richter and Westphal 2010). The landmarks considered by LAMA are sets of atoms such that one atom from each set must hold in some state along all plans. Since an action may achieve multiple landmarks, adding up the costs of each individual cheapest achiever may overestimate the true cost to satisfy all landmarks. Common admissible landmark heuristics for optimal planning avoid this overcounting by underapproximating the cost of a *minimum hitting set* (Bonet and Helmert 2010; Pommerening et al. 2014; Büchner, Keller, and Helmert 2021), since an exact computation is NP-complete (Karp 1972). We instead study two approaches that *overapproximate* this cost by finding suboptimal hitting sets computable in polynomial time in the number of landmarks, and use their cost as an inadmissible heuristic for satisficing planning. Our first approach fixes one cheapest achiever for each landmark, making the set of these fixed achievers a hitting set. The second uses a known greedy approximation. Using an established approximation factor, the second approach can even be turned into an admissible heuristic, making it applicable to optimal planning as well.

Beyond proposing new landmark heuristics, we also revisit the algorithm LAMA employs to *find* landmarks. LAMA's landmark generation restricts landmarks to be disjoint sets of atoms with cardinality 4 or less (Richter, Helmert, and Westphal 2008). These restrictions ensure that the landmark generation algorithm is polynomial in the task representation. However, limiting the number of landmarks also limits the informedness of the heuristic. We thus relax both restrictions. Firstly, we remove the requirement for disjointedness, since our heuristics are designed to consider interactions between landmarks. Secondly, we increase the maximal cardinality. These changes still guarantee a polynomial bound, albeit with a larger factor. We discard dominated landmarks to keep the number of landmarks small and maintainable in practice.

Finally, we compare our heuristics and landmark generation changes against existing methods. We first compare our adapted landmark generation against the one in LAMA, focusing on how the method affects the different heuristics. Then, we analyze our heuristics in a LAMA-style planning system. While our first approach achieves higher coverage in the single search setting, it can also lead to higher plan costs. Our second approach, however, takes longer to compute but finds better plans. Based on this insight, we combine different approaches synergistically; in the spirit of LAMA, we push to find solutions fast with fast-to-compute heuristics and improve them with better-informed heuristics. This leads to a configuration that outperforms LAMA both in terms of coverage and IPC quality score. We also apply our greedy hitting set approach to the optimal setting, but cannot match the state of the art.

## Background

We consider classical planning in the SAS$^+$ formalism (Bäckström and Nebel 1995). A *planning task* is a 4-tuple $\Pi = \langle V, A, I, G\rangle$ where $V$ is a finite set of *finite-domain state variables*; $A$ is a finite set of *actions* (or *operators*); $I$ is a *state*; and $G$ is a *partial state*. Each variable $v \in V$ has an associated finite domain $dom(v)$. An *atom* (or *fact*) $v \mapsto d$ is an assignment mapping a variable $v \in V$ to a value in its domain $dom(v)$. A *partial state* $s$ is a set of atoms, each over a different variable. With $vars(s) = \{v \mid v \mapsto d \in s \text{ for some } d\}$ we denote the variables defined by $s$, and $s(v)$ denotes the value $d$ of $v$ in $s$. A partial state is called a *state* if $vars(s) = V$. Each *action* $a \in A$ is a triple $\langle pre(a), eff(a), cost(a)\rangle$ where *precondition* $pre(a)$ and *effect* $eff(a)$ are partial states, and $cost(a) \in \mathbb{R}_0^+$ denotes the *cost* of $a$.

An action $a \in A$ is *applicable* in state $s$ if $pre(a) \subseteq s$. Applying an applicable action $a$ in $s$ leads to the *successor state* $s' = s[\![a]\!]$ where $s'(v) = d$ if $v \mapsto d \in eff(a)$ and $s'(v) = s(v)$ otherwise. An *action sequence* $\pi = \langle a_1, \dots, a_n\rangle$ is applicable in $s$ if $a_1$ is applicable in $s$, $a_2$ is applicable in $s[\![a_1]\!]$ and so forth, and we denote the resulting state by $s[\![\pi]\!]$. If $s[\![\pi]\!] \supseteq G$, then $\pi$ is called an *s-plan*. The *cost* of $\pi$ is defined as $cost(\pi) = \sum_{i=1}^{n} a_i$. If $\pi$ is an $s$-plan and $cost(\pi)$ is minimal among all $s$-plans, then we call $\pi$ an *optimal plan*. The aim of classical planning is to find an $I$-plan (also just called plan). Likewise, the aim of optimal classical planning is to find an optimal $I$-plan.

A *landmark* for a state $s$ of a planning task $\Pi = \langle V, A, I, G\rangle$ is a property that needs to be satisfied along all $s$-plans. Related work often distinguishes landmarks based on atoms and landmarks based on actions. We rely on both concepts in this paper. In particular, we consider *disjunctive fact landmarks* and *disjunctive action landmarks*. A disjunctive fact landmark $l^F$ for $s$ is a set of atoms (facts) such that all $s$-plans $\pi = \langle a_1, \dots, a_n\rangle$ visit a state containing some atom in $l^F$, i.e., there exists an $s' \in \{s[\![\langle a_1, \dots, a_i\rangle]\!] \mid 0 \le i \le n\}$ with $s' \cap l^F \neq \emptyset$.[1] Similarly, a disjuncitve action landmark $l^A \subseteq A$ is a set of actions such that $l^A \cap \{a_1, \dots, a_n\} \neq \emptyset$ for all $s$-plans. We will use that a disjunctive fact landmark $l^F$ for a state $s$ that is not satisfied in $s$ (i.e., $s \cap l^F = \emptyset$) can be translated into a disjunctive action landmark for $s$; for example, we can use the set of actions that contain one of the atoms in $l^F$ as an effect for this purpose, that is $l^A = \{a \in A \mid eff(a) \cap l^F \neq \emptyset\}$.

## The LAMA Planner

Since our contributions were developed and evaluated in the context of the LAMA planner, we first give an overview on how LAMA works and has evolved over time. LAMA is a classical planning system designed by Richter and Westphal (2010) as an extension of Fast Downward (Helmert 2006). It aims to find a suboptimal plan fast and keeps searching for shorter plans while time permits. LAMA successfully participated in the satisficing track of the IPC since the 6th edition. It won the competition in the years 2008 and 2011 and was later used as a baseline planner. Most recently, in the IPC 2023 it turned out to still perform comparatively well: In the *satisficing track* the scores were determined by plan quality, cheaper plans leading to higher scores, and only 3 out of 22 competitors obtained higher overall scores than LAMA. In the *agile track* the scores were determined by how quickly a plan is found, shorter times leading to higher scores, and LAMA ended up with a higher score than all competitors.

### Core Features

The success of LAMA can be attributed to many of the various ideas embedded in the system. At its core, LAMA is a heuristic forward search algorithm making use of (1) an anytime search algorithm, (2) multiple heuristics, and (3) preferred operators among other techniques. One of its core features is the use of a landmark heuristic (Richter, Helmert, and Westphal 2008) which is the main link to this paper.

**Anytime Search** The search employed by LAMA does not stop once it finds a solution. Instead, it restarts the search from the initial state in a slightly different configuration, trying to find a cheaper solution while time remains. More specifically, LAMA starts with a greedy best-first search (Doran and Michie 1966) and heuristics agnostic to action costs to find any short (rather than cheap) solution as fast as possible. We sometimes call this the first iteration of LAMA. If the first iteration is successful within the given resource limits, LAMA stores the found plan and restarts the search from the initial state with a weighted A$^*$search (Pohl 1970) that takes action costs into account. Whenever a cheaper plan is found, it is again stored and another weighted A$^*$search starts with a lower weight than before. The search stops when no cheaper plan can be found, or when the time or memory limit imposed from the outside is exhausted.

**Multiple Heuristics** LAMA uses a so-called multi-heuristic search (Helmert 2006; Röger and Helmert 2010) where the open list of the search algorithm(s) is actually a set of queus, each ordered by a different heuristic. When choosing the next state to expand, the search alternates between the available queues. The generated successors are then evaluated with all heuristics independently and inserted into the corresponding queues. The heuristics used in LAMA are a cost-sensitive variant of the FF heuristic $h^{\text{FF}}$ (Hoffmann and Nebel 2001; Keyder and Geffner 2008) and a path-dependent heuristic counting the number of landmarks that remain to be achieved to reach the goal (Zhu and Givan 2003). With the introduction of action costs, later versions of LAMA considered a cost-sensitive variant of this heuristic that we call $h^{\text{sum}}$ in this paper because it sums up the costs of the cheapest landmark achievers (Richter and Westphal 2010).

**Preferred Operators** Some heuristics derive information about the underlying planning problem beyond the estimated cost to reach the goal from a given state. Helmert (2006) introduces the notion of *preferred operators* to flag successor states reached through such operators as interesting independent of their heuristic value. LAMA makes use of this idea

[1]The term "disjunctive" in the name indicates that the idea can be generalised to arbitrary formulas over atoms which must hold in some state along the state trace of each plan.

by maintaining additional queues in the open list containing only states reached through preferred operators. In particular, LAMA uses an additional queue for each heuristic in its open list and alternates between all of them. (Note that these additional queues contain *all* states reached through preferred operators, independent of which heuristic declared the operator preferred.) This makes it more likely to expand states reached through preferred operators. Furthermore, whenever a state is expanded that leades to progress in terms of the heuristic value, the corresponding queue is *boosted*: for a given number of turns, the choice of state to expand next is based exclusively on this queue within the open list (Richter and Helmert 2009).

Both heuristics used in LAMA can be used to decide preferredness of operators. In the case of $h^{\mathrm{FF}}$, operators are preferred if they occur in the relaxed plan used for the heuristic estimate. For $h^{\mathrm{sum}}$, LAMA prefers operators that are applicable and reach an unreached landmark for which all predecessors in the so-called *landmark graph* are already reached. If no such operator exists, then it prefers operators that occur in a relaxed plan to a "nearest acceptable landmark" (Richter and Westphal 2010). Unlike $h^{\mathrm{FF}}$, $h^{\mathrm{sum}}$ does not generate this information as a side product when computing the heuristic value for a given state. Computing preferred operators based on landmarks is rather an extra step that can be executed on top of the heuristic computation, iterating over all applicable operators and marking all that meet the above criteria.

## Landmark Generation

Given that landmarks lie at the core of LAMA and also at the intersection with this paper, we deem it useful to repeat how landmarks are computed in LAMA. It computes a set of disjunctive fact landmarks for the initial state along with orderings between them. (We ignore landmark orderings because they are not important for our contributions.) Then, it *progresses* which landmarks remain to be achieved along the paths expanded during search. The distance estimate of $h^{\mathrm{sum}}$ is then based on the landmarks that remain to be achieved in the future of any encountered state. We refer to Büchner et al. (2023) for a discussion of landmark progression and focus on the landmark generation here.

LAMA's landmark generator finds disjunctive fact landmarks in a backwards fashion, as illustrated in Algorithm 1 (Richter, Helmert, and Westphal 2008). It first creates a fact landmark for each goal atom (line 1), and then iteratively finds new landmarks (line 2): Given landmark $l$ (line 3), it tries to find a (small) set of atoms that contains at least one precondition of all actions achieving $l$ (line 5). Since any of these preconditions needs to be satisfied in order to reach $l$, this set of atoms forms a disjunctive fact landmark. Note that considering all possible sets $F$ is not feasible; LAMA only considers sets where all atoms stem from the same predicate symbol (Richter and Westphal 2010).

The number of landmarks is limited to be polynomial in the task representation through several mechanisms. Most importantly for us, the generator disallows overlap between the disjunctive fact landmarks (line 8), that is, every atom appears in at most one landmark. Note that this can still lead to overlap when translated to disjunctive action landmarks,

Algorithm 1: Simplified LAMA landmark generation without orderings and reasoning on domain transition graphs.

```
Input: planning task $\langle V, A, I, G \rangle$
Output: set of disjunctive fact landmarks $\mathcal{L}$
 1: $open, \mathcal{L} \leftarrow$ set of goal facts in $G$ as singletons
 2: while $open \neq \emptyset$ do
 3:   $l \leftarrow open.\mathrm{pop}()$
 4:   $\mathcal{A} \leftarrow$ first achievers of $l$
 5:   for all sets of facts $F \notin \mathcal{L}$ where $F \cap pre(a) \neq \emptyset$ for
      all $a \in \mathcal{A}$ do
 6:     if $|F| = 1$ then
 7:       $\mathcal{L} \leftarrow \{l' \in \mathcal{L} \mid F \cap l' = \emptyset\}$
 8:     if $|F| \leq 4$ and $F \cap l' = \emptyset$ for all $l' \in \mathcal{L}$ then
 9:       $\mathcal{L} \leftarrow \mathcal{L} \cup \{F\}$
10:       $open.\mathrm{insert}(F)$
11: return $\mathcal{L}$
```

since two atoms may be achieved by the same action. Furthermore, the generator limits the disjunction size of landmarks to 4 (line 8). This makes sense on an intuitive level since smaller disjunctions are harder to satisfy. Moreover, a larger landmark blocks more facts from being used in other landmarks, potentially resulting in fewer landmarks overall. If both of these conditions are satisfied, the newly found is added to the set of landmarks (line 9) and inserted into the open list (line 10).

In the special case where a newly found landmark consists of a single atom (line 6), instead of discarding it, LAMA discards the previously found landmark that overlaps with it (line 7). The reasoning behind this is again that the smaller disjunction is harder to satisfy, rendering the newly found landmark more informative.

LAMA further uses information from the *domain transition graph* of each variable (i.e., the atomic projection on each variable) to derive more landmarks (Richter and Westphal 2010). More specifically, whenever a landmark $F$ is added in line 9 such that $|F| = 1$, the following procedure is applied: If the singleton atom $v \mapsto d \in F$ cannot be reached in the domain transition graph without passing a state corresponding to another atom $v \mapsto d'$, then $F' = \{v \mapsto d'\}$ is also a (disjunctive) fact landmark. We leave this aspect out of Algorithm 1 for the sake of simplicity.

## Evolution

The code base of LAMA was originally based on Fast Downward, in which it has since been integrated. Since Fast Downward is under active development, what we consider to be the LAMA planner today also underwent some significant changes.[2] We briefly summarize some major differences between LAMA as it is described in the original paper by Richter and Westphal (2010) and the version underlying this paper.

One major change is the way landmark information is progressed along the search space. Büchner et al. (2023) show

[2]See in particular https://issues.fast-downward.org/issue987 concerned with landmarks.

that LAMA's original landmark progression applies incorrect inference in determining which landmarks remain to be achieved. They further suggest a principled and sound progression which has since been implemented in Fast Downward. Before this change, cycles in the landmark graphs (i.e., cyclic dependencies according to the orderings between landmarks) were an issue because LAMA would never consider the involved landmarks reached. LAMA therefore systematically removed orderings from the landmark graph until it was acyclic. With the sound progression, this issue no longer persists and Fast Downward no longer breaks up cycles in the landmark graph. While this may lead to more orderings, today's implementation no longer considers the so-called *obedient-reasonable orderings* leading to fewer orderings in some cases; on the one hand, Büchner et al. leave it an open question how to deal with obedient-reasonable orderings in a principled way, on the other hand their usefulness in practice is questionable.

We describe above how LAMA derives preferred operators. This procedure has also changed since the original version. Today, landmark heuristics in Fast Downward consider operators preferred if they achieve *any* landmark required in the future of a given state, not only when this landmark is achieved for the first time. Further, it is no longer a requirement that all predecessors in the landmark graph of this landmark are already achieved. If no applicable operator can achieve a landmark, then no operators are preferred. Since the main use case of preferred operators from landmark heuristics within Fast Downward is LAMA, and LAMA already considers preferred operators from relaxed plans through $h^{\text{FF}}$, computing relaxed plans to the nearest landmark and considering the corresponding operators preferred was found to not affect performance in any meaningful way. In general, experiments have shown that using preferred operators based on landmarks is not beneficial in the context of LAMA. Therefore, the recommended configuration of LAMA only uses $h^{\text{FF}}$ to determine preferred operators and does not apply the extra step to compute preferred operators from $h^{\text{sum}}$. Note that there are still four queues in the open list of LAMA, though, as the states preferred by $h^{\text{FF}}$ also end up in a queue ordered by the evaluation function based on $h^{\text{sum}}$ for these states.

## Hitting Set Heuristics

We now turn over to view landmarks as sets of actions rather than atoms. Recall that even though LAMA ensures that disjunctive fact landmarks are disjoint, translating them to disjunctive action landmarks may lead to overlaps because a single action may achieve several atoms occurring in different disjunctive fact landmarks. LAMA considers each landmark to have a cost that corresponds to the cheapest action in the landmark, i.e., $\mathit{cost}(l) = \min_{a \in l} \mathit{cost}(a)$. Now consider a state $s$ and an associated set of landmarks $\mathcal{L}$. The landmark heuristic $h^{\text{sum}}$ used in LAMA returns the sum of all landmark costs: $h^{\text{sum}}(s) = \sum_{l \in \mathcal{L}} \mathit{cost}(l)$. While $h^{\text{sum}}$ is fast to compute, it ignores the possibility that landmarks may overlap, meaning that several landmarks can be achieved by one action. For example, if we have two landmarks $l_1 = \{a_1, a_2\}$ and $l_2 = \{a_1, a_3\}$, with $\mathit{cost}(a_1) = 1$

Algorithm 2: Greedy Hitting Set Heuristic $h^{\text{ghs}}$

```
Input: set of landmarks $\mathcal{L}$
Output: heuristic estimate $h^{\text{ghs}}$
1: $h^{\text{ghs}} \leftarrow 0$
2: while $\mathcal{L} \neq \emptyset$ do
3:     select $a \in A$ with minimal $\frac{\mathit{cost}(a)}{|\{l \in \mathcal{L} \mid a \in l\}|}$
4:     $h^{\text{ghs}} \leftarrow h^{\text{ghs}} + \mathit{cost}(a)$
5:     $\mathcal{L} \leftarrow \mathcal{L} \setminus \{l \in \mathcal{L} \mid a \in l\}$
6: return $h^{\text{ghs}}$
```

and $\mathit{cost}(a_2) = \mathit{cost}(a_3) = 2$, $h^{\text{sum}}$ returns a heuristic value of $\mathit{cost}(a_1) + \mathit{cost}(a_1) = 2$, while a single application of $a_1$ with a cost of 1 satisfies both landmarks. Considering an alternative cost function $\mathit{cost}'$ with $\mathit{cost}' = \mathit{cost}$ except $\mathit{cost}'(a_1) = 3$, $h^{\text{sum}}$ computes a heuristic value of $\mathit{cost}'(a_2) + \mathit{cost}'(a_3) = 4$, even though we achieve both landmarks by applying $a_1$ with a cost of only 3.

In order to account for overlapping landmarks, we aim to find a *set of actions* that achieves all landmarks. This can be modeled as a *weighted hitting set problem*: Given a universe of elements $U$ and a set $\mathcal{S}$ consisting of sets of elements from $U$, along with a cost function $\mathit{cost} : U \to \mathbb{R}$, a hitting set is a set $H \subseteq U$ that "hits" each element of $\mathcal{S}$, that is $H \cap S \neq \emptyset$ for all $S \in \mathcal{S}$. The cost of $H$ is the sum of all its elements, i.e., $\mathit{cost}(H) = \sum_{u \in H} \mathit{cost}(u)$. By setting $U = A$, $\mathcal{S} = \mathcal{L}$, and *cost* as the action cost function, finding a set of actions that achieves all landmarks means finding a hitting set for $\mathcal{L}$. The cost of the hitting set can then be used as the heuristic value. Additionally, and in contrast to $h^{\text{sum}}$, the hitting set can be used to identify preferred operators with almost no overhead by considering all applicable actions in the hitting set as preferred.

Using hitting sets to compute landmark heuristics is not a new idea. Several approaches have been proposed to underapproximate the cost of a *minimum hitting set* for optimal planning (e.g., Bonet and Helmert 2010; Pommerening et al. 2014; Büchner, Keller, and Helmert 2021). We instead focus on satisficing planning where admissibility is not a concern for heuristic design. Finding a minimum hitting set is an NP-complete problem (Karp 1972), so we suggest to find suboptimal hitting sets and use their cost to overapproximate the cost of a minimum hitting set.

Starting from $h^{\text{sum}}$, a straightforward way to calculate a hitting set while maintaining computational efficiency is to start with the collection of actions that $h^{\text{sum}}$ would sum over, and then remove all duplicates. The resulting collection is a set because there are no duplicates, and it hits all landmarks because only duplicate actions are removed. We denote the resulting heuristic "hitting sum", or $h^{\text{hs}}$. Since $h^{\text{sum}}$ is based on landmark costs which are only implicitly linked to actions, we pick the first minimal cost action according to a fixed ordering for $h^{\text{hs}}$.

While this approach avoids the problem of counting the same action multiple times (illustrated by the example with cost function *cost*) the problem of summing multiple cheaper actions instead of a single expensive action (illustrated by the example with cost function $\mathit{cost}'$) still persists.

To also address this second issue, we instead use a well-known greedy approximation of small hitting sets, originally described for the *set cover* problem (Chvátal 1979), which is equivalent to the hitting set problem (Ausiello, D'Atri, and Protasi 1980). Algorithm 2 is an adaption written in the hitting set perspective and computes our greedy hitting set heuristic $h^{\text{ghs}}$. It iteratively selects actions such that the ratio between the cost of the action and the number of newly covered landmarks is minimal, until all landmarks are covered. In the example with cost function *cost*$'$, this approach selects $a_1$ in line 3 during the first iteration already because it appears in two landmarks, rendering its ratio $\frac{cost(a_1)}{2} = \frac{3}{2}$ lower than $\frac{cost(a_2)}{1} = \frac{cost(a_3)}{1} = 2$ for $a_2$ and $a_3$ which only appear in one landmark each.

## Derived Variables

Since the landmark generation method considered in this paper is based on disjunctive fact landmarks rather than disjunctive action landmarks, it is important to be able to transform the former into the latter. For planning tasks defined in SAS$^+$, this transformation is straightforward, as outlined in the background section. For other formalisms, computing the achievers of an atom can become difficult. In such cases, a valid alternative is to overapproximate the set of achievers. This ensures that the resulting disjunctive action landmark is still a landmark, but it might render the heuristic less informed than having the exact set of achievers.

One example where computing the set of achievers is difficult is the PDDL feature called derived variables (Edelkamp and Hoffmann 2004). A derived variable is a variable whose value is not changed by actions, instead it is uniquely derived from other state variables with the help of so-called axioms. For example, in the BLOCKSWORLD domain, where we stack blocks into towers, one could define a derived variable `above(x,y)` which is true if either `on(x,y)` is true or there is a `z` such that `on(x,z)` and `above(z,y)` is true. To extract the exact set of achievers of a landmark containing a derived variable, one would need to define the different constellations of non-derived variables that cause the derived variable to be true, which is a complex task. Fast Downward instead uses the set of all actions as a naive overapproximation; after all, in order to make a currently false derived variable true, we do know that we need to do *something*.

For $h^{\text{sum}}$ this overapproximation results in each such landmark losing a bit of information: instead of using the cost of the cheapest action of the real achiever set, the cost of the overall cheapest action is used. For $h^{\text{hs}}$ and $h^{\text{ghs}}$ on the other hand, almost all information from such landmarks is lost since *all* landmarks containing derived variables can now be reached by any single action. If we have at least one landmark without any derived variable we thus get the same heuristic value as if we would delete all such landmarks. Otherwise (if all landmarks contain derived variables), the heuristic value is equal to the cost of the overall cheapest action.

## Optimal Planning

As previously mentioned, approximating the cost of hitting sets has been used for optimal planning before. In this setting, we need to guarantee that the heuristic is *admissible*, that is that it never overestimates the cost to the goal. Since any path from the current state to the goal must achieve all landmarks that are yet to be achieved, a lower bound on the cost of a minimum hitting set is guaranteed to be admissible. While both $h^{\text{hs}}$ and $h^{\text{ghs}}$ instead compute an upper bound, we can derive a lower bound for $h^{\text{ghs}}$ as well, because the greedy hitting set algorithm has a guaranteed bound on the overapproximation: The cost of the hitting set found in Algorithm 2 is at most $H_d$ times as high as the cost of a minimum hitting set, where $d = \max_{a \in A} |\{l \in \mathcal{L} \mid a \in l\}|$ is the largest amount of landmarks an action can cover, and $H_n = \sum_{i=1}^{n} \frac{1}{n}$ is the $n$-th harmonic number (Chvátal 1979). This means that by dividing $h^{\text{ghs}}$ by $H_d$ we obtain an admissible estimate and can thus use it in optimal planning as well. We denote this variant by $h^{\text{ghs-opt}}$.

## Overlapping Landmark Generation

Since our new heuristics account for overlap between landmarks, we propose to adapt LAMA's landmark generation to allow overlapping disjunctive fact landmarks. While this change primarily targets our new heuristics, we remark that overlapping landmarks can contain valuable information in general. Consider for example a problem with atoms $x, y, z$ and the landmarks $\{x, y\}$, $\{x, z\}$, and $\{y, z\}$ (among others). No matter which of these landmark is found first in Algorithm 1, the other two will be discarded (line 8).

Furthermore, the generator disallows overlaps during the entire algorithm, not only in the final set of landmarks it returns. If a newly generated landmark $l$ overlaps with an existing landmark $l'$, then $l$ is immediately discarded unless $|l| = 1$, in which case $l'$ is discarded (line 6-7). Now consider the following example: The current set of landmarks is $\mathcal{L} = \{\{x, y\}\}$. The algorithm finds a new landmark $\{y, z\}$ but discards it since it overlaps with $\{x, y\}$. Next, it finds $\{x\}$, which replaces $\{x, y\}$ in $\mathcal{L}$, yielding $\mathcal{L}' = \{\{x\}\}$. Since $\{y, z\}$ is also a landmark and does not overlap with $\{x\}$ we could now add it to $\mathcal{L}'$, but the generator does no longer know about it and will only return $\mathcal{L}'$.

Finally, the argument behind special casing $|l| = 1$ is that $l$ contains strictly stronger information than any $l' \supset l$. We say that $l$ *dominates* $l'$ in this case. However, domination is not restricted to $|l| = 1$ but holds for arbitrary subset relations. Consider another case where $\mathcal{L} = \{\{x, y, z\}\}$ and the next landmark found is $\{x, y\}$. Clearly, $\{x, y\}$ is harder to satisfy than $\{x, y, z\}$, but in this case Algorithm 1 discards $\{x, y\}$ instead of replacing $\{x, y, z\}$ as it would in the case of $\{x\}$, $\{y\}$ or $\{z\}$. We do not see a good argument why domination should only be considered in the special case of $|l| = 1$.

Based on all these examples, we suggest to modify Algorithm 1 by dropping lines 6-7 and remove the second condition in line 8. This change leads us to also reconsider the restriction on the landmark size to at most 4 atoms, also imposed in line 8. As mentioned in the discussion of

LAMA's landmark generation, large landmarks are not only less informative, but they can also have a negative impact on the generation procedure when we disallow overlap. More specifically, larger landmarks block away more atoms from appearing in new landmarks. This problem cannot occur when we allow overlap, thus we have less reason to restrict landmark size. Nevertheless, we want to keep the total number of landmarks tractable, thus we still enforce a maximum size limit but increase the constant.

Theoretically, this change can lead to a significantly larger number of landmarks. Assuming a task with $n$ atoms and maximal landmark size $k$, up to $\sum_{i=1}^{k}\binom{n}{k}$ could be found (before the dominance elimination). At the same time, when disallowing overlaps, we get at most $n$ landmarks. However, we do not expect these changes to lead to significant runtime differences for two reasons:

1. We expect that the generation algorithm will only detect a tiny fraction of all $\sum_{i=1}^{k}\binom{n}{k}$ disjunctions of up to $k$ facts as landmarks. Firstly, many of those disjunctions are likely not landmarks at all. Secondly, we expect the algorithm to only find a small number of actual landmarks, because the number of landmarks found in each backtracking step is limited and landmark sizes tend to grow quickly with each step, limiting the overall amount of backtracking steps.
2. Disallowing overlap limits the *final* number of landmarks but not the number of landmarks generated during the procedure. Algorithm 1 may generate any number of landmarks in line 5, only to disregard them in line 8. This means that even when disallowing overlap, it is theoretically possible, albeit highly unlikely, to find all landmarks and discarding all but $n$ of them.

While allowing overlap and increasing the maximal landmark size enables us to consider more landmarks, this also introduces a new problem: Landmarks become increasingly redundant. We alleviate this issue by removing dominated landmarks in a post-processing step. We can safely disregard dominated landmarks because they are satisfied as soon as the dominating landmark is. We further adapt Algorithm 1 by adding the line

$$\mathcal{L} \leftarrow \{l \in \mathcal{L} \mid \nexists l' \in \mathcal{L} : l' \subset l\}$$

just before returning on line 11. Note that we additionally use these dominance relations to infer landmark orderings, but this discussion falls outside the scope of this paper.

## Experimental Evaluation

We implemented our approach on top of Fast Downward version 23.06 (Helmert 2006). In order to analyze our hitting set heuristics and changes to the landmark generation in isolation, we first run a simplified version of LAMA's first iteration, removing the $h^{\mathrm{FF}}$ heuristic and the use of preferred operators. In a second step we test them as fully fledged planning systems. Furthermore we tested a variant which uses different landmark heuristic in the different iterations of LAMA. Finally we looked into how the admissible $h^{\text{ghs-opt}}$ performs in an optimal setting.

| | | all domains | | | no derived variables | | |
|---|---|---|---|---|---|---|---|
| | | $h^{\mathrm{sum}}$ | $h^{\mathrm{hs}}$ | $h^{\mathrm{ghs}}$ | $h^{\mathrm{sum}}$ | $h^{\mathrm{hs}}$ | $h^{\mathrm{ghs}}$ |
| 4 | ○○ | 1985 | 1983 | 1958 | 1660 | 1718 | 1693 |
| | ⚭ | 1961 | 1974 | 1954 | 1636 | 1709 | 1689 |
| 10 | ○○ | **1998** | **2006** | 1975 | **1680** | **1742** | 1710 |
| | ⚭ | 1979 | **2006** | **1983** | 1662 | **1742** | **1718** |

Table 1: Coverage results between $h^{\mathrm{sum}}$, $h^{\mathrm{hs}}$, and $h^{\mathrm{ghs}}$. The left-most column indicates the maximal landmark size and the following column whether overlapping is allowed (⚭) or not (○○). The right side restricts from all 90 domains to those 81 without derived variables.

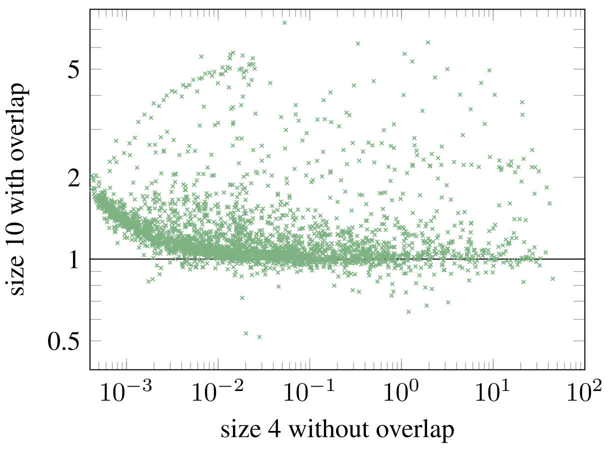

Figure 1: Relative landmark generation time in seconds.

Our experiments were conducted on AMD EPYC 7742 2.25GHz processors, using a time limit of 30 minutes and memory limit of 3.5 GiB. The results were evaluated with Downward Lab (Seipp et al. 2017). Our benchmark suite consists of 2882 planning tasks from the satisficing tracks of the IPCs 1998–2023. Due to the limitations of hitting set heuristics when dealing with landmarks containing derived variables, we sometimes additionaly report results restricted to domains without derived variables. See Büchner et al. (2024) for code, benchmarks and experimental data.

### Simplified First Iteration

To analyze the impact of our proposed landmark generation changes, we separately controlled disjunction size and overlap. For the disjunction size we tried values from 4 to 14 in increments of 2. When not allowing overlap, the number of landmarks across all tasks only increased slightly (from $\approx 484\,000$ for 4 to $\approx 511\,000$ for 14), but with overlap we observed two larger jumps from 6 to 8 ($\approx +90\,000$) and from 10 to 12 ($\approx +85\,000$). Table 1 reports the coverage results for 4 (baseline) and 10 (best overall performance).

While the time for landmark generation increases as we allow for larger sizes and overlaps, it does so within reason. Figure 1 compares size 4 without overlap to size 10 with overlap. For the vast majority of tasks the time spent on generation is less than doubled, but can increase up to a factor of 8. Yet, the generation time stays below 10 seconds for the majority of the problems, which we deem reasonable given the overall time limit of 30 minutes.

**Disjunction Size** The configurations using the larger disjunction size dominate across all used heuristics. We find that $h^{\text{sum}}$ solves the fewest additional tasks and $h^{\text{hs}}$ the most, and that configurations with overlap generally benefit more.

**Overlap** While allowing larger disjunctions is generally beneficial, not all configurations benefit from overlapping landmarks. The $h^{\text{sum}}$ heuristic consistently performs worse with overlap, confirming that it is undesirable to consider overlapping landmarks independently. Our new heuristics perform worse with overlap and the default disjunction size, while for disjunction size 10 overlap has no influence for $h^{\text{hs}}$ and a positive influence for $h^{\text{ghs}}$. We assume that overlap introduces some degree of inaccuracy for *all* considered heuristics. This results in a trade-off between the usefulness of the additional landmark information and the inaccuracies from the overlap. As we progress from $h^{\text{sum}}$ over $h^{\text{hs}}$ to $h^{\text{ghs}}$ the heuristics become better equipped to handle overlap. Similarly, as we progress to larger disjunction sizes, the overlap between landmarks becomes more likely to be small relative to the landmark size, harming the heuristic less.

**Heuristics** Finally, we compare the heuristics against each other by considering their respective best performing versions, namely size 10 without overlap for $h^{\text{sum}}$, and size 10 with overlap for both $h^{\text{hs}}$ and $h^{\text{ghs}}$. Taking $h^{\text{sum}}$ as a baseline, $h^{\text{hs}}$ improves coverage, solving more tasks in 16 domains and fewer in 15. Coverage drops for $h^{\text{ghs}}$ on the other hand, which solves more tasks in 14 domains compared to $h^{\text{sum}}$, and fewer in 21. On domains without derived variables, both $h^{\text{hs}}$ and $h^{\text{ghs}}$ achieve significantly higher coverage than $h^{\text{sum}}$.

Looking at the domains with the biggest difference, the hitting set heuristics lose 42 tasks on the two PSR domains but solve 65 tasks more in SCHEDULE. For PSR the results can be explained with the fact that roughly 70% of landmarks contain derived variables, highlighting the uninformedness of hitting set heuristics when dealing with derived variables. While not as impactful in absolute numbers, the OPTICALTELEGRAPH domain paints an even clearer picture: *all* landmarks contain derived variables, and coverage drops from 6 (for $h^{\text{sum}}$) to 2 (for $h^{\text{hs}}$ and $h^{\text{ghs}}$). The SCHEDULE domain on the other hand highlights the advantage of hitting set heuristics when having landmarks with synergies. It describes problems where we have several machines available to change properties of objects, such as shape, surface texture, color and holes. It contains actions for each machine to schedule the processing of one object, and one time-step action in which all scheduled tasks are executed and the machines and objects are free to use again. Many landmarks found in these problems are single fact landmarks that require an object or machine to be free. All these landmarks have the same set of achievers, namely the set containing only the time step action. While $h^{\text{hs}}$ and $h^{\text{ghs}}$ correctly realize that all these landmarks can be achieved in one step, $h^{\text{sum}}$ counts all of them separately, making the heuristic significantly less accurate.

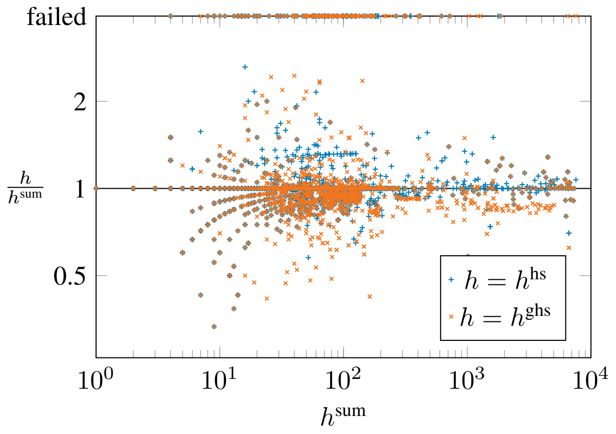

Figure 2: Relative cost of plans found in the first iteration by $h^{\text{sum}}$ compared against $h^{\text{hs}}$ and $h^{\text{ghs}}$. PARCPRINTER has plan costs in the order $10^6$ and is excluded to improve clarity.

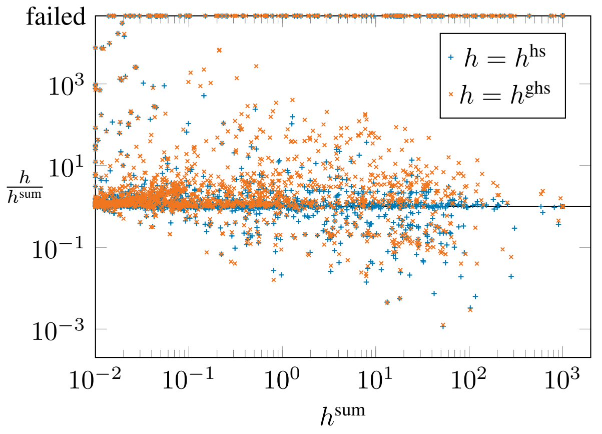

Figure 3: Relative runtime of $h^{\text{sum}}$ compared against that of $h^{\text{hs}}$ and $h^{\text{ghs}}$ in the first iteration.

Figures 2 and 3 compare the plan cost and runtime of $h^{\text{sum}}$ to that of $h^{\text{hs}}$ and $h^{\text{ghs}}$. While $h^{\text{hs}}$ often finds worse plans than $h^{\text{sum}}$, the opposite is true for $h^{\text{ghs}}$, suggesting that $h^{\text{ghs}}$ guides the search towards better plans. In terms of runtime, the results are mixed for $h^{\text{hs}}$ and generally worse for $h^{\text{ghs}}$, indicating that the improved guidance of $h^{\text{ghs}}$ comes with increased computation cost. This increase may be due to the more expensive heuristic, but it is also possible that more states need to be expanded in order to find a plan. The increased runtime also significantly affects the reasons for failure: While $h^{\text{sum}}$ and $h^{\text{hs}}$ time out in 5 tasks, $h^{\text{ghs}}$ does so in up to 200 tasks, depending on the configuration.

## Comparison against LAMA

For a full planner comparison we introduce LAMA-$h^{\text{hs}}$ and LAMA-$h^{\text{ghs}}$. They are identical to LAMA except for the heuristic used and changes to disjunction size and overlap. Since $h^{\text{sum}}$ improved with disjunction size 10, we further include LAMA-10 that differs only in the disjunction size.

| pref. op. | | all domains | | no derived var. | |
|---|---|---|---|---|---|
| | | score | cov. | score | cov. |
| FF | LAMA | 2357.25 | **2458** | 1957.81 | 2056 |
| | LAMA-10 | 2350.12 | 2455 | 1950.27 | 2052 |
| | LAMA-$h^{\mathrm{hs}}$ | 2324.18 | 2426 | 1952.02 | 2052 |
| | LAMA-$h^{\mathrm{ghs}}$ | 2346.74 | 2429 | 1974.48 | 2055 |
| FF + LM | LAMA | 2329.93 | 2439 | 1929.62 | 2037 |
| | LAMA-10 | 2320.71 | 2431 | 1922.60 | 2031 |
| | LAMA-$h^{\mathrm{hs}}$ | 2300.09 | 2401 | 1930.60 | 2030 |
| | LAMA-$h^{\mathrm{ghs}}$ | **2361.93** | 2444 | **1987.33** | **2068** |

Table 2: IPC scores and coverage of LAMA-like planners.

The top part of Table 2 shows the IPC quality score and coverage of each planner using the standard LAMA settings. The IPC quality score is the best plan cost[3] divided by the found plan cost. The coverage improvements for $h^{\mathrm{sum}}$ with larger disjunction size and $h^{\mathrm{hs}}$ do not carry over, most likely because $h^{\mathrm{FF}}$ compensates in domains where $h^{\mathrm{sum}}$ is weaker. Further, the negative impact of $h^{\mathrm{hs}}$ in terms of cost, as its IPC quality score is significantly worse. However, LAMA-$h^{\mathrm{ghs}}$ is much closer to LAMA in terms of IPC quality score, compensating its comparatively low coverage with better plans.

The default configuration of LAMA only considers preferred operators from the $h^{\mathrm{FF}}$ heuristic, because the computation of $h^{\mathrm{sum}}$ preferred operators is too expensive. Since the hitting set heuristics compute preferred operators as a side product, we tested the impact of taking the preferred operators from landmark heuristics into account. The results are shown in the bottom part of Table 2. They confirm that preferred operators from $h^{\mathrm{sum}}$ are harmful for standard LAMA. Similarly they negatively affect LAMA-$h^{\mathrm{hs}}$, presumably because the found hitting set is not a good approximation. For LAMA-$h^{\mathrm{ghs}}$ we however see a significant improvement, resulting in the highest IPC quality score despite lower coverage. This indicates that the hitting sets found by $h^{\mathrm{ghs}}$ are a good representation of which actions should be applied. When only considering domains without derived variables, LAMA-$h^{\mathrm{ghs}}$ with additional preferred operators from $h^{\mathrm{ghs}}$ achieves both the highest coverage and highest IPC quality score among all configurations.

**Combining Heuristics** As pointed out before, LAMA is designed to find an arbitrarily bad solution fast before gradually improving it. We can follow this path with our collected findings from the previous experiments. So far we have seen that $h^{\mathrm{sum}}$ and $h^{\mathrm{hs}}$ tend to find plans faster, but $h^{\mathrm{ghs}}$ finds plans of better quality. This suggests that a LAMA configuration using one of the former two heuristics in the earlier iterations and $h^{\mathrm{ghs}}$ in the later iterations might combine the best of two worlds. We thus evaluated two further LAMA-like configurations that use either $h^{\mathrm{sum}}$ or $h^{\mathrm{hs}}$ without preferred operators for the landmark heuristic in the first iteration, and $h^{\mathrm{ghs}}$ with preferred operators for all other iterations. The resulting planners behave similarly in terms of coverage: LAMA-$h^{\mathrm{sum}}$-$h^{\mathrm{ghs}}$ achieves a coverage of 2458 (identical to LAMA which also uses $h^{\mathrm{sum}}$ in its first iteration) and LAMA-$h^{\mathrm{hs}}$-$h^{\mathrm{ghs}}$ reaches 2427 (compared to 2426 for LAMA-$h^{\mathrm{hs}}$). This is to be expected since the first iteration is the only one affecting coverage. Regarding the IPC quality score however, LAMA-$h^{\mathrm{sum}}$-$h^{\mathrm{ghs}}$ improves on LAMA and LAMA-$h^{\mathrm{ghs}}$ with a score of 2378.08, confirming that finding any solution fast with a fast heuristic but then using a more sophisticated heuristic in later iterations pays off.

[3]The minimum of all found plans' costs and the upper bound from planning.domains (Muise 2016, accessed on April 5, 2024).

## Optimal Planning

In order to evaluate whether a greedy hitting set can also serve as a good underapproximation, we tested the admissible heuristic $h^{\text{ghs-opt}}$ in an optimal setting, comparing against the landmark cost partitioning heuristic (Karpas and Domshlak 2009; Domshlak et al. 2011). We ran three configurations, which all use A$^*$ search with a single heuristic and use our altered landmark generation with overlap and disjunction size 10. They only differ in the heuristic: $h^{\mathrm{ucp}}$ performs uniform cost partitioning, $h^{\mathrm{ocp}}$ optimal cost partitioning and $h^{\text{ghs-opt}}$ the admissible greedy hitting set heuristic. The configurations were evaluated on 1847 planning tasks from all STRIPS domains of the optimal IPC 1998–2023 tracks.

The highest coverage of 1007 is achieved by $h^{\mathrm{ucp}}$, while the other configurations trail behind with a coverage of 904 for $h^{\text{ghs-opt}}$ and 907 for $h^{\mathrm{ocp}}$. The reason is in both cases that the heuristic is too expensive to compute: $h^{\mathrm{ocp}}$ must solve an LP in each state, while $h^{\text{ghs-opt}}$ has to perform the iterative computation in each state. The uniform cost partitioning in $h^{\mathrm{ucp}}$, however, can be computed in time linear in the sum of the number of landmark achievers. While $h^{\mathrm{ocp}}$ dominates $h^{\mathrm{ucp}}$, $h^{\text{ghs-opt}}$ seems to result in worse heuristic values, as it almost always requires more expansions than $h^{\mathrm{ucp}}$.

## Conclusion

The landmark heuristic used in LAMA can be inaccurate due to ignoring synergies between landmarks. We propose two hitting-set-based heuristics that take synergies into account: $h^{\mathrm{hs}}$ creates a hitting set by fixing one action as the achiever for each landmark, while $h^{\mathrm{ghs}}$ uses a greedy hitting set algorithm that is more expensive to compute but yields higher quality hitting sets. Together with allowing overlaps in the disjunctive fact landmarks, increasing the maximal disjunction size, and utilizing the hitting set as a source for preferred operators, trading computation cost for better guidance pays off for $h^{\mathrm{ghs}}$: Its LAMA-like planner is on par with LAMA, and surpasses it on domains without derived variables. Furthermore, combining the fast computation of $h^{\mathrm{sum}}$ and accuracy of $h^{\mathrm{ghs}}$ by using the former in the first iteration and the latter afterwards further strengthens the planner, resulting in the overall highest coverage and score.

An important open question is how to improve our heuristics on domains with derived variables. To this end a more principled way to approximate the achievers of derived variables should be investigated. Furthermore, we believe that $h^{\mathrm{hs}}$ has the potential to offer better guidance without sacrificing runtime by developing more sophisticated tie-breaking criteria for deciding which achiever is selected.

## Acknowledgments

This research was supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme (grant agreement no. 952215) and by the Swiss National Science Foundation (SNSF) as part of the project "Lifted and Generalized Representations for Classical Planning" (LGR). A special thanks goes to Malte Helmert who supported this work with helpful recommendations and discussions.

## References

Ausiello, G.; D'Atri, A.; and Protasi, M. 1980. Structure Preserving Reductions among Convex Optimization Problems. *Journal of Computer and System Sciences*, 21(1): 136–153.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS$^+$ Planning. *Computational Intelligence*, 11(4): 625–655.

Bonet, B.; and Helmert, M. 2010. Strengthening Landmark Heuristics via Hitting Sets. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, 329–334. IOS Press.

Büchner, C.; Christen, R.; Eriksson, S.; and Keller, T. 2024. Code, Benchmarks and Experiment Data for the ICAPS 2024 HSDIP Workshop Paper "Hitting Set Heuristics for Overlapping Landmarks". https://doi.org/10.5281/zenodo.11148129.

Büchner, C.; Eriksson, S.; Keller, T.; and Helmert, M. 2023. Landmark Progression in Heuristic Search. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*, 70–79. AAAI Press.

Büchner, C.; Keller, T.; and Helmert, M. 2021. Exploiting Cyclic Dependencies in Landmark Heuristics. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 65–73. AAAI Press.

Chvátal, V. 1979. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3): 233–235.

Domshlak, C.; Helmert, M.; Karpas, E.; Keyder, E.; Richter, S.; Röger, G.; Seipp, J.; and Westphal, M. 2011. BJOLP: The Big Joint Optimal Landmarks Planner. In *IPC 2011 Planner Abstracts*, 91–95.

Doran, J. E.; and Michie, D. 1966. Experiments with the Graph Traverser program. *Proceedings of the Royal Society A*, 294: 235–259.

Edelkamp, S.; and Hoffmann, J. 2004. PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Technical Report 195, University of Freiburg, Department of Computer Science.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Karp, R. M. 1972. Reducibility among combinatorial problems. In Miller, R. E.; and Thatcher, J. W., eds., *Complexity of Computer Computations*, 85–103. Plenum Press.

Karpas, E.; and Domshlak, C. 2009. Cost-Optimal Planning with Landmarks. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1728–1733. AAAI Press.

Keyder, E.; and Geffner, H. 2008. Heuristics for Planning with Action Costs Revisited. In Ghallab, M.; Spyropoulos, C. D.; Fakotakis, N.; and Avouris, N., eds., *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, 588–592. IOS Press.

Muise, C. 2016. Planning.Domains. In *ICAPS 2016 System Demonstrations and Exhibits*.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1: 193–204.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based Heuristics for Cost-optimal Planning. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 226–234. AAAI Press.

Richter, S.; and Helmert, M. 2009. Preferred Operators and Deferred Evaluation in Satisficing Planning. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 273–280. AAAI Press.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 975–982. AAAI Press.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Röger, G.; and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In Brafman, R.; Geffner, H.; Hoffmann, J.; and Kautz, H., eds., *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 246–249. AAAI Press.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Zhu, L.; and Givan, R. 2003. Landmark Extraction via Planning Graph Propagation. In *ICAPS 2003 Doctoral Consortium*, 156–160.