ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230569

A Landmark-Cut Heuristic for Lifted Optimal Planning

Julia Wichlacz[a;*], **Daniel Höller**[a], **Daniel Fišer**[a] **and Jörg Hoffmann**[a,b]

[a]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[b]German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany

Abstract. Lifted planning – finding plans directly on the PDDL input model – has attracted renewed attention during the last years. This avoids the process of grounding, which can become computationally prohibitive very easily. However, the main focus of recent research in this area has been on *satisficing*, i.e., (potentially) suboptimal planning. We present a novel heuristic for *optimal* lifted planning. Our basic idea is inspired by the LM-cut heuristic, which has been very successful in grounded optimal planning. Like LM-cut, we generate cut-based landmarks via back-chaining from the goal, generating cuts of partially grounded actions. However, exactly mimicking the ground formulation is not feasible, this includes computing the $h^{\max}$ heuristic several times for one computation of the LM-cut heuristic (which is already NP-hard to compute). We show that our heuristic is admissible and evaluate it in a *cost optimal* setting.

1 Introduction

Heuristic search has been extremely successful in AI planning (e. g. [2, 12, 10, 22, 25]). But this research has almost exclusively been based on grounded (propositional) task representations, in contrast to the *lifted* PDDL input models, that employ variables ranging over a finite universe of objects. The grounded representation has – in worst-case – size exponential in the arity (number of arguments) of the PDDL predicates and action schemas. Hence, planning techniques at the grounded level are limited to planning tasks where that blow-up is not prohibitive. It has been frequently observed that this excludes a variety of applications (e. g. [11, 16, 17, 8, 19]).

Lifted planning methods do not require grounding as a preprocess. They have been considered throughout the history of AI planning (e. g. [20, 24, 27], but heuristic search planning has begun to develop lifted methods only recently. Research so far devised an effective lifted forward search mechanism [5], lifted delete relaxation [23, 6, 18, 7] and landmark (LM) heuristics [26]. However, most methods focused on *satisficing* planning, where no guarantee on plan quality is given and hence inadmissible heuristics can be used. The only *heuristic* for lifted optimal search is $h^{\max}$ [7], but there are two other systems for lifted optimal planning: $h^{\text{lmc}}_{\text{hom}}$ guides a (lifted) search with ground heuristics computed on a reduced model [14]; and LiSAT translates lifted planning problems into a propositional SAT problem [13]. The latter can – however – only do length-optimal and not cost-optimal planning.

A successful heuristic for *grounded* optimal planning is the *LM-cut* heuristic [10], which generates a set of disjunctive action LMs via back-chaining from the goal definition. It comes with interesting theoretical guarantees and has also been highly successful in empirical evaluations. However, it does not directly transfer to lifted planning. The problem is that – in the grounded setting – it relies on information from the $h^{\max}$ heuristic [2]. $h^{\max}$ needs to be computed repeatedly for one computation of the LM-cut heuristic, which is expensive (NP-hard) in the lifted setting. The information from $h^{\max}$ is needed in the *precondition choice function*, which is both important to determine good cuts (i.e., LMs), but also to show theoretical properties.

Here we present a novel heuristic for *lifted optimal* planning, which is inspired by the grounded LM-cut heuristic, generating disjunctive action LMs via back-chaining from the goal. In order to circumvent the problem described above, we present alternative precondition choice functions. While this means that we cannot show dominance of $h^{\max}$ (like in the grounded setting), we show that (1) our cuts form partially grounded disjunctive action LMs, and that (2) the resulting heuristic values are admissible.

We run experiments on recent benchmarks for lifted planning [18] which we extend with domain variants featuring non-unit action costs. We compare our methods to the competitors given above: lifted $h^{\max}$, $h^{\text{lmc}}_{\text{hom}}$, and LiSAT. LiSAT turns out to be dominant in length optimal planning, while cost optimal planning has more mixed results.

2 Preliminaries

We consider normalized PDDL tasks without conditional effects and negative preconditions, and with all formulas being conjunctions of atoms (represented as sets of atoms). To simplify the presentation, we also, w.l.o.g., assume that the initial state as well as the goal consists of a single ground atom. Every planning task can be transformed to this form simply by adding two new auxiliary (ground) actions.

A **normalized PDDL task** is a tuple $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}_S, c, \psi_I, \psi_G \rangle$ where $\mathcal{B}$ is a non-empty set of **objects**, $\mathcal{T}$ is a non-empty set of **types** containing a default type denoted by $t_0 \in \mathcal{T}$, objects and types are associated by a total function $\mathcal{D} : \mathcal{T} \mapsto 2^{\mathcal{B}}$ such that $\mathcal{D}(t_0) = \mathcal{B}$ and for every pair of types $t_i, t_j \in \mathcal{T}$ it holds that $\mathcal{D}(t_i) \subseteq \mathcal{D}(t_j)$ or $\mathcal{D}(t_i) \supseteq \mathcal{D}(t_j)$ or $\mathcal{D}(t_i) \cap \mathcal{D}(t_j) = \emptyset$. $\mathcal{V}$ is a denumerable set of variable symbols, each variable $v \in \mathcal{V}$ has a type $\tau_{var}(v) \in \mathcal{T}$.

$\mathcal{P}$ is a set of **predicate symbols**, each predicate $p \in \mathcal{P}$ has **arity** $\text{ar}(p) \in \mathbb{N}$ and an associated type $\tau_{pred}(p, i) \in \mathcal{T}$ for every $i \in \{1, ..., \text{ar}(p)\}$. An **atom** is of the form $p(s_1, \ldots, s_n)$, where $p \in \mathcal{P}$ is a predicate symbol, $n = \text{ar}(p)$ is the arity of p, and each s_i is either an object $o \in \mathcal{D}(\tau_{pred}(p, i))$, or a variable $v \in \mathcal{V}$ with $\mathcal{D}(\tau_{var}(v)) \subseteq \mathcal{D}(\tau_{pred}(p, i))$. For a given atom $\alpha = p(s_1, \ldots, s_n)$, $\mathcal{V}[\alpha] \subset \mathcal{V}$ denotes a set of variables appearing in the atom, i.e., $\mathcal{V}[\alpha] = \{s_1, \ldots, s_n\} \cap \mathcal{V}$, and $\mathcal{P}[\alpha] = p$ denotes the predicate of α. Given a set of atoms X, we define $\mathcal{V}[X] = \bigcup_{x \in X} \mathcal{V}[x]$

* Corresponding Author. Email: wichlacz@cs.uni-saarland.de.

and $\mathcal{P}[X] = \bigcup_{x \in X} \mathcal{P}[x]$. A **ground atom** is an atom α such that $\mathcal{V}[\alpha] = \emptyset$.

$\mathcal{A}_S$ denotes a set of **action schemas**, each action schema $a \in \mathcal{A}_S$ is a tuple $a = \langle \mathrm{pre}(a), \mathrm{add}(a), \mathrm{del}(a) \rangle$ where $\mathrm{pre}(a)$, $\mathrm{add}(a)$ and $\mathrm{del}(a)$ are sets of atoms, called **preconditions**, **add effects**, and **delete effects**, respectively. We, w.l.o.g., assume non-empty preconditions. By $\mathcal{V}[a] = \mathcal{V}[\mathrm{pre}(a) \cup \mathrm{add}(a) \cup \mathrm{del}(a)]$ we denote a set of variables appearing in the action. The **cost** function $c : \mathcal{A}_S \mapsto \mathbb{R}_0^+$ maps each action schema to a non-negative number.

ψ_I and ψ_G are ground atoms, ψ_I is called the **initial-state atom**, and ψ_G is called the **goal atom**.

A function $\sigma : \mathcal{V} \cup \mathcal{B} \mapsto \mathcal{V} \cup \mathcal{B}$ is called a **substitution** if $\sigma(o) = o$ for every object $o \in \mathcal{B}$, and for every variable $v \in \mathcal{V}$ it holds that either (i) $\sigma(v) \in \mathcal{D}(\tau_{var}(v))$, or (ii) $\sigma(v) \in \mathcal{V}$ and $\mathcal{D}(\tau_{var}(\sigma(v))) \subseteq \mathcal{D}(\tau_{var}(v))$. We write σx to denote $\sigma(x)$. A set of all substitutions is denoted by $\mathcal{S}$.

We extend σ element-wise to sets and tuples, i.e., if $X = \{x_1, \ldots, x_n\}$, then $\sigma X = \{\sigma x_1, \ldots, \sigma x_n\}$, and if $X = \langle x_1, \ldots, x_n \rangle$, then $\sigma X = \langle \sigma x_1, \ldots, \sigma x_n \rangle$.

Given a set of substitutions $\Sigma = \{\sigma_1, \ldots, \sigma_n\} \subseteq \mathcal{S}$ and an element X (a set X), $\Sigma(X)$ denotes the set $\{\sigma x \mid x \in X\}$ $(\bigcup_{\sigma \in \Sigma} \sigma(X))$.

A substitution γ is called **grounding** if $\gamma(v) \in \mathcal{B}$ for every $v \in \mathcal{V}$. A set of all groundings is denoted by $\mathcal{G}$.

$\mathcal{A} = \mathcal{S}(\mathcal{A}_S)$ denotes the set of all possible **actions**, and we assume that for every two distinct action schemas $a \in \mathcal{A}_S$ and $a' \in \mathcal{A}_S$ it holds that $\mathcal{S}(a) \cap \mathcal{S}(a') = \emptyset$. We also extend the cost function c over the set of actions $\mathcal{A}$ by setting $c(a) = c(a_S)$ whenever $a \in \mathcal{S}(a_S)$ for some $a_S \in \mathcal{A}_S$. A **ground action** is an action a such that $\mathcal{V}[a] = \emptyset$, i.e., $\mathcal{G}(\mathcal{A}_S) = \mathcal{G}(\mathcal{A}) \subseteq \mathcal{A}$ is the set of all ground actions.

A **state** is a set of ground atoms. An **initial state** is the state $\{\psi_I\}$ and every state s_G such that $\psi_G \in s_G$ is a goal state. A ground action a is **applicable** in a state s if $\mathrm{pre}(a) \subseteq s$. The **resulting state** of applying an applicable ground action a in a state s is the state $a[s] = (s \setminus \mathrm{del}(a)) \cup \mathrm{add}(a)$. A sequence of ground actions $\pi = \langle a_1, \ldots, a_n \rangle$ is applicable in a state s_0 if there are states $s_1, \ldots, s_n$ such that a_i is applicable in s_{i-1} and $s_i = a[s_{i-1}]$ for every $i \in \langle 1, \ldots, n \rangle$. The resulting state of this application is $\pi[s_0] = s_n$. The cost of the sequence π is defined as $c(\pi) = \sum_{i=1}^{n} c(a_i)$. A sequence of ground actions $\pi = \langle a_1, \ldots, a_n \rangle$ is called **plan** if it is applicable in the initial state, and $\pi[\{\psi_I\}] \ni \psi_G$ is a goal state.

To simplify the notation, we use $[n]$ for a natural number $n \geq 1$ to denote $\{1, \ldots, n\}$.

3 Lifted Disjunctive Action Landmarks

On the ground level, a disjunctive action landmark is a set of ground actions out of which at least one must be part of every plan [21, 15]. Given a disjunctive action landmark L, we can immediately infer that the minimum cost over ground actions from L, which we denote as $c(L) = \min_{a \in L} c(a)$, is a lower bound on the cost of optimal plans, i.e., it is an admissible estimate from the initial state. Similarly, if we have two disjoint landmarks L and L', then $c(L) + c(L')$ is a lower bound as every plan has to contain at least one ground action from L and another one from L'. This clearly generalizes to an arbitrary set of (pairwise disjoint) landmarks, but we can compute a lower bound on the cost of optimal plans also from a set of disjunctive action landmarks sharing some actions using cost partitioning.

Given a set of disjunctive action landmarks $L_1, \ldots, L_n$, we can associate cost functions $c_1, \ldots, c_n$ mapping ground actions to numbers with $L_1, \ldots, L_n$, respectively, and define $c_i(L_i) = \min_{a \in L} c_i(a)$ for every $i \in [n]$. Then $\sum_{i=1}^{n} c_i(L_i)$ is a lower bound on the cost of optimal plans, if the cost functions $c_1, \ldots, c_n$ are constructed in such a way that for every ground action $a \in \bigcup_{i \in [n]} L_i$ it holds that $\sum_{i \in [n], a \in L_i} c_i(a) \leq c(a)$. In other words, the sum over $c_i(L_i)$ is an admissible estimate as long as we spread the original cost $c(a)$ of each action a appearing in landmarks across the cost functions c_i.

One particular heuristic function utilizing this principle is the LM-cut heuristic [10]. It repeatedly builds the so-called justification graph and then extracts a landmark and the corresponding cost function from it by finding a cut in the graph separating the initial state and goal. The justification graph utilized in LM-cut is a labeled directed multi-graph having facts (ground atoms) as vertices and, for each ground action a, there is a set of edges labeled with a that connect a single selected precondition with all its add effects. Preconditions are selected using the so-called *precondition choice function* (pcf) mapping ground actions to facts, which selects the precondition with the highest $h^{\max}$ value (breaking ties arbitrarily).

Here, we build on the aforementioned LM-cut heuristic designed for ground planning. First, we generalize the notion of disjunction action landmarks to the lifted level. Second, we define justification graphs on the lifted level and we generalize the notion by showing it can be built much more freely than it was originally used without loosing its property that any cut separating the initial state and goal corresponds to a landmark. In particular, we show that we can choose preconditions arbitrarily. This is not an entirely new finding as Bonet and Helmert [3] already pointed out that an arbitrary pcf can be used. We expand on this by showing that we do not even need a pcf (as a function mapping (ground) actions to (ground) atoms), but for each add effect of each action, we can choose any precondition freely. Note, however, that choosing freely does not preserve the dominance of the original LM-cut over the $h^{\max}$ heuristic anymore. Third, we show how to extract a sequence of landmarks and the corresponding cost functions from the lifted justification graph in a similar manner as it is done in the ground LM-cut heuristic leading to a new admissible landmark-cut heuristic for lifted planning.

We start by introducing the notion of *lifted disjunctive action landmark* as a set of actions (ground or lifted, i.e., with or without variables) such that its grounding is a ground disjunctive action landmark. That is, $L \subseteq \mathcal{A}$ is a lifted disjunctive action landmark if every plan contains at least one ground action from $\mathcal{G}(L)$.

Definition 1. A set of actions $L = \{l_1, \ldots, l_m\}$ is called a **lifted disjunctive action landmark** (or a landmark for short) if for every plan $\pi = \langle a_1, \ldots, a_n \rangle$ there exist $i \in \langle 1, \ldots, n \rangle$, and $j \in \langle 1, \ldots, m \rangle$ such that $a_i \in \mathcal{G}(l_j)$.

Next, we define a *lifted justification graph* as a labeled directed multi-graph where vertices are atoms (ground or lifted) and edges are labeled with actions (also ground or lifted). Following the construction of the justification graph from [10], the edges connect actions' preconditions with their add effects and the graph contains the initial-state and goal facts. In contrast to [10], we do not require a pcf. So, we require that (C1) the initial-state and goal atoms have corresponding vertices in the graph; (C2) every edge is labeled with an action and leads from one of its preconditions to one of its add effects, i.e., we do not require that all edges labeled with the same action start in the same precondition; and (C3) we require that for every non-initial-state atom u there are incoming edges whose labels cover all possible ground actions leading to any ground atom in $\mathcal{G}(u)$, i.e., the lifted justification graph must be constructed so that it includes all possible actions leading to an atom appearing in the

graph. An s-t-path is then defined as a sequence of edges leading from the vertex (atom) s to the vertex (atom) t, and an s-t-cut is a set of edges separating s and t.

Definition 2. A **labeled directed multi-graph** is a tuple $\mathcal{K} = \langle U, E, L\rangle$ where U denotes a set of vertices, L denotes a set of labels, and E denotes a set of labeled edges $\langle u, u', l\rangle$, also denoted by $u \xrightarrow{l} u'$, where $u \in U$ and $u' \in U$ are start and end vertices, respectively, such that $u \neq u'$, and $l \in L$ is a label.

A sequence of edges $\pi = \langle s_1 \xrightarrow{l_1} t_1, \ldots, s_n \xrightarrow{l_n} t_n\rangle$ is called an s_1-t_n**-path** if for every $i \in [n-1]$ it holds that $t_i = s_{i+1}$.

Given two distinct vertices $s \in U$ and $t \in U$, a set of edges $C \subseteq E$ is called an s-t**-cut** if every s-t-path contains at least one edge from C.

Definition 3. Given a normalized PDDL task $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G\rangle$, a labeled directed multi-graph $\mathcal{K} = \langle U, E, L\rangle$ is called a **lifted justification graph** for $\mathcal{P}$ if all of the following hold:

(C1) U is a set of atoms, and $\psi_I \in U$ and $\psi_G \in U$, and
(C2) for every $s \xrightarrow{l} t \in E$ it holds that l is an action and $s \in \mathrm{pre}(l)$ and $t \in \mathrm{add}(l)$, and
(C3) for every $u \in U$ such that $\psi_I \notin \mathcal{G}(u)$, and every grounding $\gamma \in \mathcal{G}$, and every ground action a_g such that $\gamma u \in \mathrm{add}(a_g)$, there exists an edge $u' \xrightarrow{l} u \in E$ such that $a_g \in \mathcal{G}(l)$.

Note that any atom can be represented by at most one vertex in a lifted justification graph (cf. condition (C1)). Before we show that every ψ_I-ψ_G-cut of any lifted justification graph correspond to a lifted disjunctive action landmark, we prove two auxiliary lemmas. In Lemma 4, we show that every edge $u' \xrightarrow{l} u \in E$ of a lifted justification graph has its grounded counterparts $a \in \mathcal{G}(l)$ and $p' \in \mathrm{pre}(a)$ such that $p' \in \mathcal{G}(u')$ (and therefore also $p' \in \mathcal{G}(\mathrm{pre}(l))$).

In Lemma 5, we show that every plan has its corresponding ψ_I-ψ_G-path in every lifted justification graph. The correspondence is such that for every plan $\langle a_1, \ldots, a_n\rangle$ there is a sequence of labels (actions) $\langle l_1, \ldots, l_m\rangle$ induced by some ψ_I-ψ_G-path such that for every l_i there is some $a_j \in \mathcal{G}(l_i)$, and the labels have the same ordering as plans' actions, i.e., given l_i corresponding to a_j and $l_{i'}$ corresponding to $a_{j'}$, if $i > i'$ then $j > j'$.

For the rest of this section, let $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G\rangle$ denote a normalized PDDL task.

Lemma 4. *Let a denote a ground action, let $p \in \mathrm{add}(a)$ denote one of its add effects, and let $\mathcal{K} = \langle U, E, L\rangle$ denote a lifted justification graph for $\mathcal{P}$. If there exists $u \in U$ such that $p \in \mathcal{G}(u)$, then there exists $u' \xrightarrow{l} u \in E$ such that $a \in \mathcal{G}(l)$ and $p' \in \mathcal{G}(u')$ for some $p' \in \mathrm{pre}(a)$.*

Proof. From (C3) it follows that there exists $u' \xrightarrow{l} u \in E$ such that $a \in \mathcal{G}(l)$, and from (C2) we have that $u' \in \mathrm{pre}(l)$ and therefore there exists $p' \in \mathrm{pre}(a)$ such that $p' \in \mathcal{G}(u')$. □

Lemma 5. *Let $\pi = \langle a_1, \ldots, a_n\rangle$ denote a plan, and let $\mathcal{K} = \langle U, E, L\rangle$ denote a lifted justification graph for $\mathcal{P}$. Then there exists a ψ_I-ψ_G-path $p = \langle u_0 \xrightarrow{l_1} u_1, \ldots, u_{m-1} \xrightarrow{l_m} u_m\rangle$ with $u_0 = \psi_I$ and $u_m = \psi_G$ and an increasing function $f : [m] \mapsto [n]$ such that for every $i \in [m]$ it holds that $a_{f(i)} \in \mathcal{G}(l_i)$.*

Proof. (By induction) First, we show that there exists $u_{m-1} \xrightarrow{l_m} u_m = \psi_G$ and $a_i \in \{a_1, \ldots, a_n\}$ s.t. $a_i \in \mathcal{G}(l_m)$. Since π is a plan, we have that $\psi_G \in \pi[\psi_I]$, and from (C1) we have $\psi_G \in U$. Let $i \in [n]$ denote an arbitrary number such that $\psi_G \in \mathrm{add}(a_i)$. From Lemma 4, we have $u \xrightarrow{l} \psi_G \in E$ such that $a_i \in \mathcal{G}(l)$ and $p \in \mathcal{G}(u)$ for some $p \in \mathrm{pre}(a)$.

Next, we show that if there exists $u_{x-1} \xrightarrow{l_x} u_x$ for some $x \in [m]$ and $a_i \in \{a_1, \ldots, a_n\}$ s.t. $a_i \in \mathcal{G}(l_x)$, then either (i) $u_{x-1} = \psi_I$ (and therefore $u_0 = u_{x-1}$), or (ii) there exists $u_{x-2} \xrightarrow{l_{x-1}} u_{x-1}$ and $a_j \in \{a_1, \ldots, a_{i-1}\}$ s.t. $a_j \in \mathcal{G}(l_{x-1})$, i.e., $j < i$ and a_j precedes a_i in π. Since we assume non-empty preconditions and a singleton initial state, $\{\psi_I\} = \mathrm{pre}(a_1)$ so (i) eventually holds. So, assuming $u_{x-1} \neq \psi_I$, the precondition $p \in \mathrm{pre}(a_i)$ s.t. $p \in \mathcal{G}(u_{x-1})$ (C3) must be achieved by some action preceding a_i in π, therefore there exists $a_j \in \{a_1, \ldots, a_{i-1}\}$ with $p \in \mathrm{add}(a_j)$ s.t. $p \in \mathcal{G}(u_{x-1})$. Therefore (ii) follows from Lemma 4.

So, there exists a ψ_I-ψ_G-path $p = \langle u_0 \xrightarrow{l_1} u_1, \ldots, u_{m-1} \xrightarrow{l_m} u_m\rangle$ with each $l_i \in \{l_1, \ldots, l_m\}$ corresponding to a different $a_j \in \{a_1, \ldots, a_n\}$ in such a way that if l_i corresponds to a_j and $l_{i'}$ corresponds to $a_{j'}$, then $i > i'$ implies $j > j'$. □

Now we can finally prove that ψ_I-ψ_G-cut induces a lifted disjunctive action landmark. The reason is simple: As we know from Lemma 5, every plan π has its corresponding ψ_I-ψ_G-path p. Therefore, every ψ_I-ψ_G-cut contains at least one edge e from every such p, and the edge e in turn corresponds to some action in π. So, every ψ_I-ψ_G-cut goes over at least one action of every plan.

Theorem 6. *Let $\mathcal{K} = \langle U, E, L\rangle$ denote a lifted justification graph for $\mathcal{P}$, and let C denote an ψ_I-ψ_G-cut, and let $C_L = \{l \mid s \xrightarrow{l} t \in C\}$. Then C_L is a lifted disjunctive action landmark.*

Proof. If there is no plan, then any set of actions is a lifted disjunctive action landmark, so let us assume there are some plans. From Lemma 5 it follows that for every plan $\pi = \langle a_1, \ldots, a_n\rangle$ there exists an ψ_I-ψ_G-path $p = \langle u_0 \xrightarrow{l_1} u_1, \ldots, u_{m-1} \xrightarrow{l_m} u_m\rangle$ and a function mapping each l_i to some a_j in such a way that $a_j \in \mathcal{G}(l_i)$. Furthermore, since C is an ψ_I-ψ_G-cut, C contains at least one edge $u_{x-1} \xrightarrow{l_x} u_x$ from every such p. Therefore, we have that $l_x \in C_L$ and we also have some $a_y \in \mathcal{G}(l_x)$ for some $y \in [n]$, which concludes the proof. □

Theorem 6 shows how to extract landmarks from a lifted justification graph, but it does not tell us how to sum them up admissibly. Note that getting an admissible estimate from a set of *lifted* landmarks is more involved than doing the same from a set of *ground* landmarks. The reason is that lifted landmarks can contain lifted actions so instead of looking for intersections between landmarks, we need to deal with intersections between groundings of landmarks. For example, let us assume we have two lifted landmarks L_1 and L_2 and we would like to extract a lower bound on the cost of optimal plans from it. When can we take the sum $\min_{a \in L_1} c(a) + \min_{a \in L_2} c(a)$? Clearly, just checking the intersection between L_1 and L_2 is not enough as the actions in L_1 and L_2 can be lifted. Instead, we need to check the intersection $\mathcal{G}(L_1) \cap \mathcal{G}(L_2)$. If it is empty, we can use the aforementioned sum. If it is not empty, we need to apply some kind of cost partitioning.

It is easy to see that $\mathcal{G}(L_1) \cap \mathcal{G}(L_2) \neq \emptyset$ holds if and only if there exist $a \in L_1$ and $a' \in L_2$ such that $\mathcal{G}(a) \cap \mathcal{G}(a') \neq \emptyset$. In other words, we can apply cost partitioning over (lifted) actions by considering non-empty intersections $\mathcal{G}(a) \cap \mathcal{G}(a')$. That is, we can construct the cost function c_1 for L_1 and c_2 for L_2 so that for every

$a \in L_1$ and $a' \in L_2$ such that $\mathcal{G}(a) \cap \mathcal{G}(a') \neq \emptyset$ it holds that $c_1(a) + c_2(a') \leq \min\{c(a), c(a')\}$ (and $c_i(a) = c(a)$ for all other actions). Then the sum $\min_{a \in L_1} c_1(a) + \min_{a \in L_2} c_2(a)$ is a lower bound on the cost of optimal plans.

To generalize this approach over larger sets of landmarks, we introduce the notion of an *action-cost database*. It is defined as a set of pairs relating an action to its cost in such a way that every ground action is covered by some entry, and for every pair $\langle a, c_a \rangle$ of the action $a \in \mathcal{A}$ and its cost $c_a \in \mathbb{R}_0^+$ in the database it holds that $c_a \leq c(a)$.

Definition 7. A set $\mathcal{D} \subseteq \mathcal{A} \times \mathbb{R}_0^+$ is called an **action-cost database** if, for every action $a \in \mathcal{A}$, there exists $\langle b, c_b \rangle \in \mathcal{D}$ such that $\mathcal{G}(a) \cap \mathcal{G}(b) \neq \emptyset$ and $c_b \leq c(a)$.

Given an action-cost database $\mathcal{D}$ and an action $a \in \mathcal{A}$, we define $\mathcal{D}(a) = \min\{c_a \mid \langle a', c_a \rangle \in \mathcal{D}, \mathcal{G}(a) \cap \mathcal{G}(a') \neq \emptyset\}$.

Given an action-cost database $\mathcal{D}$ and a set of actions $A \subseteq \mathcal{A}$, we define $\mathcal{D}(A) = \min_{a \in A} \mathcal{D}(a)$.

We use action-cost databases instead of cost functions. Given an action-cost database $\mathcal{D}$ and an action $a \in \mathcal{A}$, we define $\mathcal{D}(a)$ as the minimum cost over all entries matching a. Such a definition allows us to update action-cost databases by simply adding more action-cost pairs. On one hand, this is guaranteed to only decrease $\mathcal{D}(a)$ values. On the other hand, if we update the database with the residual costs of (lifted) actions already encountered in previously considered landmarks, then the database will provide costs of actions that can be used for further landmarks admissibly.

Consider the following example: Let L_1 and L_2 denote two lifted landmarks and let $\mathcal{D}_1$ denote an action-cost database. Furthermore, let us assume that there is exactly one $a_1 \in L_1$ and exactly one $a_2 \in L_2$ such that $\mathcal{G}(a_1) \cap \mathcal{G}(a_2) \neq \emptyset$. Clearly, both $\mathcal{D}_1(L_1)$ and $\mathcal{D}_1(L_2)$ are (each individually) admissible estimates, but $\mathcal{D}_1(L_1) + \mathcal{D}_1(L_2)$ is *not* guaranteed to be an admissible estimate. Nevertheless, we could use a's cost up to $\mathcal{D}_1(L_1)$ and leave the residual cost $\mathcal{D}_1(a) - \mathcal{D}_1(L_1)$ for a' in L_2. In other words, we can construct another action-cost database $\mathcal{D}_2 = \mathcal{D}_1 \cup \{\langle a, \mathcal{D}_1(a) - \mathcal{D}_1(L_1) \rangle\}$ and then the sum $\mathcal{D}_1(L_1) + \mathcal{D}_2(L_2)$ must be an admissible estimate. This is because adding $\langle a, \mathcal{D}_1(a) - \mathcal{D}_1(L_1) \rangle$ to $\mathcal{D}_2$ makes sure that every action a' such that $\mathcal{G}(a) \cap \mathcal{G}(a') \neq \emptyset$ gets the cost of at most $\mathcal{D}_1(a) - \mathcal{D}_1(L_1)$, which induces a cost partitioning over actions in L_1 and L_2 – it is the same as creating cost functions c_1 and c_2 such that $c_i(a) = \min\{\mathcal{D}_i(a), \mathcal{D}_i(L_i)\}$ for every $a \in L_i$, and therefore $c_2(a') \leq c_1(a) - \min_{a'' \in L_1} c_1(a'')$.

In the following theorem, we prove this principle formally for any sequence of lifted disjunctive action landmarks as long as the first action-cost database in the sequence starts with the original costs of actions.

Theorem 8. *Let $L_1, \ldots, L_n$ denote a sequence of disjunctive action landmarks, let $\mathcal{D}_1, \ldots, \mathcal{D}_n$ denote a sequence of action-cost databases such that $\mathcal{D}_1 = \{\langle a, c(a) \rangle \mid a \in \mathcal{A}_S\}$ and, for every $i \in \{2, \ldots, n\}$, $\mathcal{D}_i = \mathcal{D}_{i-1} \cup \{\langle a, \mathcal{D}_{i-1}(a) - \mathcal{D}_{i-1}(L_{i-1}) \rangle \mid a \in L_{i-1}\}$, and let $c^\star$ denote the cost of an optimal plan. Then $\sum_{i=1}^n \mathcal{D}_i(L_i) \leq c^\star$.*

Proof. Before we get to the main claim, we need to verify that, for every $i \in [n]$, $\mathcal{D}_i$ is a well-defined action-cost database. For that, it is enough to show that for every $i \in [n]$, and every $a \in L_i$, it holds that $0 \leq \mathcal{D}_i(a) - \mathcal{D}_i(L_i) \leq c(a)$. This trivially holds for $i = 1$ as $\mathcal{D}_1(a) = c(a)$. For $i \geq 2$, it easy to see that $\mathcal{D}_i(a) \geq \mathcal{D}_i(L_i) \geq 0$ by Definition 7, therefore $\mathcal{D}_i(a) - \mathcal{D}_i(L_i)$ is non-negative, and also $\mathcal{D}_i(a) \leq \mathcal{D}_j(a)$ for any $j \in [i-1]$ because $\mathcal{D}_1(a) = c(a)$ and any consecutive $\mathcal{D}_i(a)$ can only decrease, concluding the first part of the proof.

Let $K_i = \mathcal{G}(L_i)$ for every $i \in [n]$, and let $A_K = \bigcup_{i \in [n]} K_i$. Next, we prove that $\sum_{i=1}^n \mathcal{D}_i(K_i) = \sum_{i=1}^n \mathcal{D}_i(L_i)$. From Definition 7, it easily follows that $\mathcal{D}(a) = \min_{a_g \in \mathcal{G}(a)} \mathcal{D}(a_g)$ for any action $a \in \mathcal{A}$ and any action-cost database $\mathcal{D}$. Therefore, for any action-cost database $\mathcal{D}$ and any set of actions $L \subseteq \mathcal{A}$ we have that $\mathcal{D}(L) = \min_{a \in L} \mathcal{D}(a) = \min_{a \in L} \min_{a_g \in \mathcal{G}(a)} \mathcal{D}(a_g) = \min_{a'_g \in \mathcal{G}(L)} \mathcal{D}(a'_g) = \mathcal{D}(\mathcal{G}(L))$. Therefore, we have that $\mathcal{D}_i(L_i) = \mathcal{D}_i(K_i)$ for every $i \in [n]$.

Finally, we prove that $\sum_{i=1}^n \mathcal{D}_i(K_i) \leq c^\star$. Since every L_i is a landmark, then also every K_i is a landmark (Definition 1). Therefore, it is enough to show that for every $a \in A_K$ it holds that $\sum_{i \in [n], a \in K_i} \mathcal{D}_i(K_i) \leq c(a)$. Keeping in mind that $\mathcal{D}_i(a) \geq \mathcal{D}_i(K_i)$ for every K_i s.t. $a \in K_i$, we prove an even stronger claim: For every $a \in A_K$ and every $i \in [n]$ such that $a \in K_i$ it holds that $\sum_{j \in [i-1], a \in K_j} \mathcal{D}_j(K_j) + \mathcal{D}_i(a) \leq c(a)$. Since $c(a) = \mathcal{D}_1(a) \geq \mathcal{D}_k(a) \geq \mathcal{D}_l(a)$ for every $1 \leq k \leq l \leq n$, it is easy to see that the claim holds for the smallest possible i. Now, we assume it holds for some $i \in [n]$, and we prove that it also holds for the smallest possible $k \in \{i+1, \ldots, n\}$, i.e., we assume $a \in K_i$ for some $i \in [n]$ and it holds that $\sum_{j \in [i-1], a \in K_j} \mathcal{D}_j(K_j) + \mathcal{D}_i(a) \leq c(a)$, and we prove that $\sum_{j \in [k-1], a \in K_j} \mathcal{D}_j(K_j) + \mathcal{D}_k(a) \leq c(a)$ for $k = \min\{x \in \{i+1, \ldots, n\} \mid a \in K_x\}$. Since $k > i$ and for every $x \in \{i+1, \ldots, k-1\}$ it holds that $a \notin K_x$, it follows that that $\mathcal{D}_k(a) \leq \mathcal{D}_i(a) - \mathcal{D}_i(K_i)$ holds by construction (as we already know that $\mathcal{D}_i(L_i) = \mathcal{D}_i(K_i)$). Moreover, we have that $\sum_{j \in [k-1], a \in K_j} \mathcal{D}_j(K_j) + \mathcal{D}_k(a) = \sum_{j \in [i-1], a \in K_j} \mathcal{D}_j(K_j) + \mathcal{D}_i(K_i) + \mathcal{D}_k(a)$. So, everything put together, we have that $\sum_{j \in [i-1], a \in K_j} \mathcal{D}_j(K_j) + \mathcal{D}_i(K_i) + \mathcal{D}_k(a) \leq \sum_{j \in [i-1], a \in K_j} \mathcal{D}_j(K_j) + \mathcal{D}_i(K_i) + \mathcal{D}_i(a) - \mathcal{D}_i(K_i) = \sum_{j \in [i-1], a \in K_j} \mathcal{D}_j(K_j) + \mathcal{D}_i(a) \leq c(a)$ which concludes the proof. □

4 Computing Lifted Landmark-Cuts

Moving from theory to practice, here we introduce a novel algorithm utilizing the ideas from the previous section. We start with the definition of (most general) unifiers as substitutions under which sets of atoms or actions become singletons.

Definition 9. Given a set of atoms or actions X, a substitution σ is called **unifier** for X if $|\sigma X| = 1$, and $\sigma v = v$ for every variable $v \notin \mathcal{V}[X]$.

A unifier σ for X is called a **most general unifier** (MGU) for X if, for every unifier τ for X, there exists another unifier ρ such that $\tau = \rho\sigma$.

To simplify the formalization, we tacitly assume than, unless explicitly specified otherwise, every unifier σ for X maps every variable $v \in \mathcal{V}[X]$ either to itself or to a fresh variable not used anywhere else, i.e., it never "recycles" any variable already used in any atom or other substitution.

It is well-known, that if there exists a unifier, then there also exists a most general unifier and it is unique up to renaming of variables [4]. It is also easy to see that $\mathcal{G}(a) \cap \mathcal{G}(a') \neq \emptyset$, for a pair of atoms or actions a and a', if and only if there exist an MGU for $\{a, a'\}$. Moreover, note that we assume unifiers are the identity mapping outside the considered set of atoms or actions. For example, given an action a, its add effect $p \in \mathrm{add}(a)$, some atom q, and a unifier σ for $\{p, q\}$,

σ can re-map only the variables appearing in p and q, i.e., applying σ on a preserves the variables $\mathcal{V}[a] \setminus \mathcal{V}[p]$.

The proposed algorithm, encapsulated in Algorithm 1, follows the schema laid out in Theorem 8 by looking for lifted landmarks one by one while maintaining the updated action-cost database. In each cycle, a lifted landmark with non-zero cost (according to the current action-cost database) is extracted as an ψ_I-ψ_G-cut of a lifted justification graph (Theorem 6). The algorithm terminates once it fails to find a non-zero lifted landmark which happens when there exists a zero-cost ψ_I-ψ_G-path.

The most noticeable difference to the theory is that we do not construct the full lifted justification graph. Instead, we iteratively build the smallest portion of the lifted justification graph connected to the goal vertex ψ_G containing a single non-zero ψ_I-ψ_G-cut. More precisely, we start with ψ_G and enumerate all possible actions having ψ_G in their add effects. For every such action a we select one of its preconditions $p \in \mathrm{pre}(a)$ (using some function `SelectPre`) and repeat this process for p as long as the explored actions have cost zero according to the current action-cost database (or ψ_I is reached). The non-zero actions encountered during this process then form a (non-zero) lifted disjunctive action landmark because we enumerate all possible actions achieving each explored atom (cf. `Achievers` and (C3)) and therefore we obtain a cut separating ψ_I and ψ_G.

Looking at Algorithm 1 in more detail, the algorithm maintains the resulting heuristic value h initialized on line 1 to zero and increased with each non-zero landmark (line 6). It also maintains the action-cost database $\mathcal{D}$ initialized at line 2 and updated on line 7 in the way as described in Theorem 8. For completeness, we also include the function `Cost` $(\mathcal{D},a)$ implementing $\mathcal{D}(a)$ as per Definition 7. The main part of the algorithm on lines 3 to 8 extracts landmarks using the function `FindCut` (line 3 and 8) one by one while updating the heuristic value h (line 5 and 6) and the action-cost database $\mathcal{D}$ (line 7) with the cost of the current landmark.

The function `FindCut` uses the function `ExploreGoalZone`, wich iteratively builds a portion of the lifted justification graph, extracts all atoms P_0 from which the goal vertex ψ_G is reachable with zero-cost paths, and a non-zero ψ_I-ψ_G-cut A_+ which consists of all actions having their add effects in P_0. Moreover, `ExploreGoalZone` returns empty sets (which eventually leads to the termination of the algorithm) whenever it connects ψ_I and ψ_G with a zero-cost path. Note that `ExploreGoalZone` can use an arbitrary function `SelectPre` for selecting one of the preconditions of the given action. It does not even have to be a function in a mathematical sense as it can have an internal memory and return a different precondition for the same action when called multiple times. Lastly, `FindCut` tries to reduce A_+ by removing actions whose selected precondition lies in the P_0 set. Since we are allowed to choose any precondition when constructing the lifted justification graph, we can reduce the cut A_+ by removing (non-zero cost) actions connecting vertices within P_0 because they are not necessary for separating ψ_I from ψ_G. Next, we sketch a proof showing that Algorithm 1 returns an admissible estimate for any function `SelectPre`.

Theorem 10. *Let $c^\star$ denote the cost of optimal plans. Then Algorithm 1 with any* `SelectPre` *function returns $h \leq c^\star$.*

Proof Sketch. Assuming `FindCut` eventually returns the empty set, the algorithm terminates. Assuming `FindCut` returns either the empty set or a valid ψ_I-ψ_G-cut, it follows from Theorem 8 that the main procedure on lines 1 to 8 results in an admissible estimate h. Furthermore, assuming `ExploreGoalZone` returns either empty sets or a ψ_I-ψ_G-cut A_+ and a set of all vertices P_0 from which ψ_G is reachable with a zero-cost path, the procedure on lines 11 to 16 clearly disregards only the actions from A_+ that are not necessary for separating ψ_I and ψ_G, therefore `FindCut` returns either the empty set or an ψ_I-ψ_G-cut.

Therefore, we need to prove that (1) `ExploreGoalZone` eventually returns empty sets, and (2) if A_+ and P_0 returned by `ExploreGoalZone` are non-empty, then (2a) A_+ is a ψ_I-ψ_G-cut and (2b) P_0 contains all vertices from the lifted justification graph from which the vertex ψ_G is reachable with zero-cost paths.

The main observations here are that `ExploreGoalZone` builds the portion of the lifted justfication graph starting from the ψ_G vertex and backchaining via actions add effects to their precondition which means that conditions (C1) and (C2) from Definition 3 hold for the explored portion of the graph. Anothoer crucial observa-

Algorithm 1: Lifted Landmark-Cut Heuristic

Input: Set of action schemas $\mathcal{A}_S$, initial-state atom ψ_I, goal atom ψ_G
Output: Heuristic value h

```
1  h ← 0;
2  D ← {⟨a, c(a)⟩ | a ∈ A_S};
3  C ← FindCut(D);
4  while |C| > 0 do
5      k ← min_{a∈C} Cost(D,a);
6      h ← h + k;
7      D ← D ∪ {⟨a, Cost(D,a) − k⟩ | a ∈ C};
8      C ← FindCut(D);
9  function FindCut(D)
10     A+, P0 ← ExploreGoalZone(D);
11     C ← ∅;
12     foreach a ∈ A+ do
13         q ← SelectPre(a);
14         if q is not unifiable with any p ∈ P0 then
15             C ← C ∪ {a};
16     return C
17 function ExploreGoalZone(D)
18     Q ← {ψ_G};
19     P0 ← {ψ_G};
20     A+ ← {};
21     while |Q| ≥ 0 do
22         p ← Pop(Q);
23         foreach a ∈ Achievers(A_S, p) do
24             if Cost(D,a) = 0 then
25                 q ← SelectPre(a);
26                 if there exists an MGU for {q, ψ_I} then
27                     return ∅, ∅;
28                 Q ← Q ∪ {q};
29                 P0 ← P0 ∪ {q};
30             else
31                 A+ ← A+ ∪ {a};
32     return A+, P0;
33 function Achievers(A_S, p)
34     A ← ∅;
35     foreach a ∈ A_S do
36         foreach q ∈ add(a) do
37             σ ← Find MGU for {p, q};
38             if σ exists then
39                 A ← A ∪ {σa};
40     return A;
41 function Cost(D, a)
42     k ← c(a);
43     foreach ⟨b, c_b⟩ ∈ D do
44         if c_b < k and there exists an MGU for {b, a} then
45             k ← c_b;
46     return k;
```

tion is that the function `Achievers` always enumerates all possible actions having the given atom p in their add effects. Therefore, `ExploreGoalZone` either reaches ψ_I with a zero-cost path, or the condition (C3) holds for all vertices (atoms) in P_0. Therefore, it immediatelly follows that (2b) holds, and (2a) holds because it also means that all non-zero actions having atoms from P_0 in their add effects are added to A_+. As for (1), it is easy to see that since we are iteratively reducing costs of actions in the action-cost database and since we are not skipping any action during the building of the justification graph, it must eventually happen that we connect ψ_I with ψ_G by a zero-cost path. □

As given before, `SelectPre` can be any function selecting a precondition of an action. Here, we evaluate the following functions:

- **LMC-random** selects a precondition randomly, i.e., it can select a different precondition every time it is called on the same action.
- **LMC-most-gr** selects the precondition having least variables, ties are broken arbitrarily. Fully lifted atoms are more likely unifiable with ψ_I, so avoiding those leads to a larger graph and potentially more cuts.
- **LMC-least** selects the precondition whose predicate was picked as a supporter least often so far, ties are broken arbitrarily. Picking predicates we have not seen in the graph so far is more likely to lead to new action schemas in the graph.
- **LMC-h^{Lmax}** selects the precondition with the highest $h^{\max}$ value breaking ties arbitrarily. This is the variant closest to the original LM-cut heuristic. For this purpose, we adapted the already existing h^{Lmax} heuristic [6] computing $h^{\max}$ values using a Datalog program on the lifted level.

5 Experimental Evaluation

We implemented[1] our approach on top of the Powerlifted (PWL) system [6]. We evaluate four configurations *LMC-random*, *LMC-most-gr*, *LMC-least*, and *LMC-h^{Lmax}*, described in the previous section. We compare our system against several systems from the literature. We use two grounded configurations of Fast Downward [9]:

- $h^{\max}$: ground A* search combined with the $h^{\max}$ heuristic [2].
- h^{lmc}: ground A* search combined with the LM-cut heuristic [10].

Then we compare against several lifted systems:

- h^{Lmax}: lifted PWL A* search with the $h^{\max}$ heuristic [6].
- $h^{\mathrm{lmc}}_{\mathrm{hom}}$: lifted A* with LM-cut computed on the task reduced with PDDL homomorphisms that was subsequently grounded [14], we use the best performing variant, as reported in [14], that preserves goal objects and reduces the planning task by 95% of objects.
- LiSAT: translation of the lifted problem into a propositional SAT problem [13]. We use the non-incremental, length-optimal configuration with the Kissat [1] solver.

Be aware that neither $h^{\mathrm{lmc}}_{\mathrm{hom}}$ nor LiSAT uses a heuristic computed on the lifted model, making h^{Lmax} the system closest to ours.

We evaluate our system on a benchmark set that has been introduced specifically for lifted planning [6, 18] and has been used in recent papers (see also [6, 18, 13, 26]).

However, the benchmarks have been introduced for satisficing planning and therefore most domains come with unit costs. So to make the cost-optimal planning evaluation more meaningful, we adapted the domains by adding cost values[1] in the following way:

- **GED** – already came with action costs.
- **Organic Synthesis** – since the costs represent the "effort" of a reaction, we assigned an action its number of effects as cost value.
- **Pipesworld** – here we distinguish unary/non-unary pipes; operations on unary pipes come with cost 1, those on non-unary cost 3.
- **VisitAll** – the version of *visitAll* that we build on comes with more than two dimensions; inspired by some kind of high bay warehouse, we gave moving on the first two dimensions cost 1, the third dimension cost 5 (since some kind of lift needs to be used), and further increased costs the higher the dimensions get.
- **Childsnack** – here we tried to reflect the temporal effort of the different actions in their costs and assigned costs 5 to making the sandwich, 1 to putting it on the tray, and 3 to serving it.
- **Blocks World** – here we tried to reflect the control effort (putting a block on some other block being more difficult than putting it on the table) and energy consumption (putting a block down being less effort than lifting it) in the costs and assigned 5, 3, 5 and 10 to *pickup*, *putdown*, *stack* and *unstack*, respectively.
- **Logistics** – operations of planes cost more than those of trucks, moving costs more than loading and unloading.
- **Rovers** – here we assigned costs based on the temporal effort and energy consumption of the actions (e.g., taking an image costing less than taking a rock sample).

We conduct two separate evaluations, one for the length-optimal and one for the cost-optimal setting. Since the SAT-based solver can only generate *length*-optimal solutions, we only use it in this setting. The other systems can plan both *length*- and *cost*-optimal.

	$h^{\max}$	h^{lmc}	LiSAT	$h^{\mathrm{lmc}}_{\mathrm{hom}}$	h^{Lmax}	LMC-random	LMC-most-gr	LMC-least	LMC-h^{Lmax}
ged (156)	16	18	**38**	18	16	22	7	21	18
ged-spl (156)	18	18	**30**	18	16	13	20	18	20
orgsy-alk (18)	15	15	**18**	17	**18**	6	6	**18**	**18**
orgsy-mit (18)	2	2	**18**	9	**18**	9	10	17	14
orgsy-org (20)	0	0	**20**	1	8	7	8	7	2
pipeswrl (50)	7	8	**20**	14	5	9	9	9	2
visitall (180)	70	62	**103**	56	66	31	31	36	26
childsnack (144)	5	4	**50**	4	0	0	0	1	0
blocks (40)	1	8	**40**	11	0	29	37	1	3
logistics (40)	2	6	**30**	25	4	0	0	0	0
rovers (40)	1	2	4	**5**	1	1	1	0	0
Total Sum (826)	137	143	**371**	178	152	127	129	128	103

Table 1: Coverage for length-optimal planning. The grounded systems are shown on the left, lifted systems on the right.

Figure 1 shows our coverage results for the unit-cost setting. It can be seen that in this setting, no search-based approach reaches the performance of the SAT-based approach on the given benchmark set. From the search-based systems, $h^{\mathrm{lmc}}_{\mathrm{hom}}$ has the hightest coverage, followed by h^{Lmax} and our configurations. As we expected *LMC-h^{Lmax}* falls behind on overall coverage. The overall coverage of *LMC-random*, *LMC-most-gr*, and *LMC-least* are close, but they perform very different across the domain set. The greatest difference is in the *Blocks World* domain, where the best of our configurations (*LMC-most-gr*) solves 37 out of 40 instances, while *LMC-least* solves only a single instance. Notably, this is the one of two domains where a search-based configuration reaches a performance close to the SAT-based system. The other is *Organic Synthesis* ("alk" and "mit"), but here several search-based systems are close together.

[1] The code can be found at: https://github.com/minecraft-saar/powerlifted

	$h^{\max}$	h^{lmc}	$h^{\text{lmc}}_{\text{hom}}$	h^{Lmax}	LMC-random	LMC-most-gr	LMC-least	LMC-h^{Lmax}
ged (156)	**18**	**18**	**18**	16	4	0	12	8
ged-spl (156)	**18**	**18**	**18**	16	4	0	16	8
orgsy-alk (18)	15	15	17	**18**	6	6	**18**	**18**
orgsy-mit (18)	2	2	9	**18**	9	10	16	12
orgsy-org (20)	0	0	1	**8**	6	5	7	1
pipeswrl (50)	8	8	**13**	5	8	8	7	2
visitall (180)	**70**	58	44	60	26	13	18	9
childsnack (144)	11	**18**	6	1	0	0	1	0
blocks (40)	1	8	11	0	**36**	**36**	1	3
logistics (40)	2	6	**19**	6	0	0	0	0
rovers (40)	1	2	**10**	1	2	1	0	0
Total Sum (826)	146	153	**166**	149	101	79	96	61

Table 2: Coverage for cost-optimal planning. The grounded systems are shown on the left, lifted systems on the right.

Another domain with large differences between our configurations is *Organic Synthesis (alk)*, where *LMC-least* and *LMC-*h^{Lmax} perform better then the others. In *GED*, *LMC-most-gr* does not perform well.

Compared to the other search-based systems, our configurations perform worse mainly in two domains: *VisitAll* and *Logistics*. However, the latter seems to be difficult for all PWL-based configurations, the only search-based system performing well is $h^{\text{lmc}}_{\text{hom}}$.

Let us next have a look at the cost-optimal setting (Table 2). When we first compare our configurations, we see that there are – again – differences in the performance among the domains: *LMC-least* performs well in *GED* and *Organic Synthesis*, *LMC-random* in *VisitAll* and both *LMC-random* and *LMC-most-gr* in *Blocks World*. We also see the expected drop in overall coverage by LMC-h^{Lmax}.

Comparing all systems, $h^{\text{lmc}}_{\text{hom}}$ performs best. Using ground heuristics on the transformed model seems to work well on the given benchmark set. Compared to the other lifted heuristic (*LMC-*h^{Lmax}), our configurations perform worse especially in *visitAll* and *Logistics*. We will discuss the latter domain in more detail in the next section. Our system shows especially good results in the *Blocks World*. Here it even beats $h^{\text{lmc}}_{\text{hom}}$.

6 Discussion and Conclusion

In this paper, we presented a lifted version of the admissible LM-cut heuristic. We also showed that the justification graph used in LM-cut for the extraction of landmarks can be built much more freely than previously used. Moreover, we introduced an algorithm that does not need to construct the whole justification graph to infer a landmark, but it constructs just the minimal part of the graph connecting a non-zero cost landmark to the goal.

A main question is how to come up with an informative pcf that can be computed on the lifted model. As we expected, using *LMC-*h^{Lmax} seems to be too costly and resulted in the worst overall coverage of our pcfs. Further, there is also not a single domain where it outperformed the other pcfs. The other pcfs perform similar when looking at overall coverage, but there are large differences between the domains. While this is not good for the comparison with other systems as done in the evaluation, it opens several directions for future work. The most elegant solution would be to find a different pcf combining the advantages of the ones presented here. But from a practical perspective, there are also other ways to combine the advantages. One would be to compute several landmark sets and do a cost partitioning or hitting set approach based on them. Computationally, this might be no problem, since the different steps like successor generation or heuristic computation are more costly in lifted planning than in the grounded setting. If it is too costly, it might be an option to select the best pcf for a given planning instance based on the initial heuristic value. Since the heuristics are admissible, a simple criterion might be to pick the one with highest value on the initial node.

One detail in the evaluation made us especially interested: we did not expect the poor result in the *Logistics* domain, both in terms of absolute performance (coverage of 0 across all pcfs), nor in comparison to h^{Lmax} (the latter is different e.g. in *Childsnack*). A look in the data revealed that all 6 instances solved by h^{Lmax} in the cost-optimal setting have a single goal, resulting in very informed h^{Lmax} values.

While this is an extreme example, it seems that the benchmark set has been constructed in a way to get large, but simple problems, resulting in problems with very few goals. Another example is *VisitAll*, which comes with 1 to 3 goals.

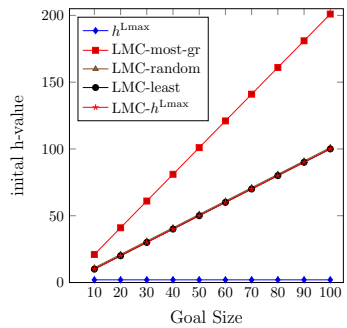

Figure 1: Plot showing different initial heuristic values for logistics problems with larger goals. Configurations LMC-random, LMC-least, LMC-h^{Lmax} all have identical values.

Based on this insight, we want to get an impression about scaling behavior of the heuristics across goals and are especially interested in the comparison to h^{Lmax}, since these are the only lifted heuristics in the evaluation ($h^{\text{lmc}}_{\text{hom}}$ uses grounded heuristics on a reduced model).

We created a small *Logistics* problem with a single *city*, a single *truck*, and an increasing number of *packages*, all of which are placed at a single position and need to be brought to the same goal position. Since h^{Lmax} and our configurations are admissible, we can meaningfully compare the initial heuristic values and we know that a larger value is always better. Figure 1 shows the number of packages on the x axis and the initial heuristic value on the y axis. In this example, we observe the behavior that we expected when comparing h^{Lmax} with other heuristics: while h^{Lmax} cannot incorporate costs of more than one sub-goal (resulting in a constant heuristic value shown at the bottom), our heuristics accumulate costs over all sub-goals, resulting in a heuristic value scaling with the number of goals. Be further aware that not only one, but all our different pcf functions scale reasonably well with the packages. In this example, *LMC-most-gr* returns the best heuristic values.

While this discussion points towards a problem that might be present in the benchmark set, we want to underline that we do not have a good solution for it: the lifted systems are – yet – not as powerful as the grounded ones. So instances in the benchmark set need to be relatively simple to solve. At the same time, they shall be large enough to break the systems needing a grounded model. This results in a set like the one used in recent work (and here).

Acknowledgments

„Gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 232722074 – SFB 1102 / Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 232722074 – SFB 1102“

References

[1] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger, ‘CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020’, in *Proceedings of the 2020 SAT Competition – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pp. 51–53. University of Helsinki, (2020).

[2] Blai Bonet and Héctor Geffner, ‘Planning as heuristic search’, *Artificial Intelligence*, **129**(1–2), 5–33, (2001).

[3] Blai Bonet and Malte Helmert, ‘Strengthening landmark heuristics via hitting sets’, in *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI’10)*, pp. 329–334. IOS Press, (2010).

[4] Samuel R. Buss, ‘An introduction to proof theory’, in *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, chapter I, 1–78, Elsevier, (1998).

[5] Augusto B. Corrêa, Florian Pommerening, Malte Helmert, and Guillem Francès, ‘Lifted successor generation using query optimization techniques’, in *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS’20)*, pp. 80–89. AAAI Press, (2020).

[6] Augusto B. Corrêa, Florian Pommerening, Malte Helmert, and Guillem Francès, ‘Delete-relaxation heuristics for lifted classical planning’, in *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS’21)*, pp. 94–102. AAAI Press, (2021).

[7] Augusto B. Corrêa, Florian Pommerening, Malte Helmert, and Guillem Francès, ‘The FF heuristic for lifted classical planning’, in *36th AAAI Conference on Artificial Intelligence (AAAI’22)*, pp. 9716–9723. AAAI Press, (2022).

[8] Patrik Haslum, ‘Computing genome edit distances using domain-independent planning’, in *Proceedings of the SPARK Workshop*, (2011).

[9] Malte Helmert, ‘The Fast Downward planning system’, *Journal of Artificial Intelligence Research*, **26**, 191–246, (2006).

[10] Malte Helmert and Carmel Domshlak, ‘Landmarks, critical paths and abstractions: What’s the difference anyway?’, in *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS’09)*, pp. 162–169. AAAI Press, (2009).

[11] Jörg Hoffmann, Stefan Edelkamp, Sylvie Thiébaux, Roman Englert, Frederico Liporace, and Sebastian Trüg, ‘Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4’, *Journal of Artificial Intelligence Research*, **26**, 453–541, (2006).

[12] Jörg Hoffmann and Bernhard Nebel, ‘The FF planning system: Fast plan generation through heuristic search’, *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).

[13] Daniel Höller and Gregor Behnke, ‘Encoding lifted classical planning in propositional logic’, in *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS’22)*, pp. 134–144. AAAI Press, (2022).

[14] Rostislav Horčík, Daniel Fišer, and Álvaro Torralba, ‘Homomorphisms of lifted planning tasks: The case for delete-free relaxation heuristics’, in *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI’22)*, pp. 9767–9775. AAAI Press, (2022).

[15] Erez Karpas and Carmel Domshlak, ‘Cost-optimal planning with landmarks’, in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI’09)*, pp. 1728–1733, (2009).

[16] Alexander Koller and Jörg Hoffmann, ‘Waking up a sleeping rabbit: On natural-language sentence generation with FF’, in *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS’10)*, pp. 238–241. AAAI Press, (2010).

[17] Alexander Koller and Ronald Petrick, ‘Experiences with planning for natural language generation’, *Computational Intelligence*, **27**(1), 23–40, (2011).

[18] Pascal Lauer, Álvaro Torralba, Daniel Fišer, Daniel Höller, Julia Wichlacz, and Jörg Hoffmann, ‘Polynomial-time in PDDL input size: Making the delete relaxation feasible for lifted planning’, in *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI’21)*, pp. 4119–4126. IJCAI organization, (2021).

[19] Rami Matloob and Mikhail Soutchanski, ‘Exploring organic synthesis with state-of-the-art planning techniques’, in *Proceedings of the SPARK Workshop*, pp. 52–61, (2016).

[20] J. Scott Penberthy and Daniel S. Weld, ‘UCPOP: A sound, complete, partial order planner for ADL’, in *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR’92)*, pp. 103–114. Morgan Kaufmann, (1992).

[21] Silvia Richter, Malte Helmert, and Matthias Westphal, ‘Landmarks revisited’, in *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI’08)*, pp. 975–982. AAAI Press, (2008).

[22] Silvia Richter and Matthias Westphal, ‘The LAMA planner: Guiding cost-based anytime planning with landmarks’, *Journal of Artificial Intelligence Research*, **39**, 127–177, (2010).

[23] Bram Ridder and Maria Fox, ‘Heuristic evaluation based on lifted relaxed planning graphs’, in *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS’14)*, pp. 244–252. AAAI Press, (2014).

[24] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1995.

[25] Jendrik Seipp, ‘Pattern selection for optimal classical planning with saturated cost partitioning’, in *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI’19)*, pp. 5621–5627, (2019).

[26] Julia Wichlacz, Daniel Höller, and Jörg Hoffmann, ‘Landmark heuristics for lifted planning’, in *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI’22)*, pp. 4665–4671. IJCAI organization, (2022).

[27] Håkan L. S. Younes and Reid G. Simmons, ‘VHPOP: versatile heuristic partial order planner’, *Journal of Artificial Intelligence Research*, **20**, 405–430, (2003).