Pattern Database Heuristics for Lifted Planning

Mika Skjelnes[1], Dominik Drexler[1], Jordan Thayer[1], Daniel Gnad[2, 1], Jendrik Seipp[1]

[1]Linköping University, Sweden
[2]Heidelberg University, Germany
firstname.lastname@liu.se, daniel.gnad@uni-heidelberg.de

Abstract

Pattern database (PDB) heuristics are among the strongest abstraction heuristics for grounded planning, but they have so far not been available in the lifted setting. We present a method for computing PDBs and for generating interesting patterns directly for lifted planning tasks. We construct patterns by lazily growing a causal graph backwards from the goal atoms, which avoids enumerating all ground atoms upfront. Given a pattern, we compute its goal distances in two phases: first, we build the abstract state space over the pattern atoms by grounding only the relevant projected actions; second, we run Dijkstra's algorithm to obtain the optimal goal distances. We evaluate our lifted PDBs on hard-to-ground (HTG) benchmarks, admissibly combining several abstractions with the canonical heuristic. Our results show that lifted PDBs can solve tasks faster than other admissible lifted heuristics. We see this as a first step toward bringing the strengths of PDB heuristics to lifted optimal planners.

1 Introduction

Grounding has been the standard first step in automated planning for decades, because the grounded representation is well suited to heuristic search: successor states can be generated efficiently (often millions per second), and a wide range of heuristics is available for both guidance and pruning. Its drawback is well known: grounding can cause a combinatorial blow-up in the size of the task representation (Areces et al. 2014; Masoumi, Antoniazzi, and Soutchanski 2015; Gnad et al. 2019; Wichlacz, Torralba, and Hoffmann 2019). Most IPC benchmarks are not affected by this, but several industrial applications, such as organic synthesis and Airbus logistics, contain tasks that state-of-the-art grounded planners cannot even ground, let alone solve.

This has led to renewed interest in lifted planning (Corrêa and De Giacomo 2024), with successor generators built on top of CSP solvers (Francès 2017), conjunctive queries (Corrêa et al. 2020), SQLite queries (Horčík and Fišer 2021) and maximum-clique enumeration (Ståhlberg 2023). A parallel line of work develops lifted heuristics: delete-relaxation heuristics computed via Datalog (Corrêa et al. 2021, 2022; Lauer et al. 2025), the k-ary relaxation (Lauer et al. 2021), lifted landmarks (Wichlacz, Höller, and Hoffmann 2022), lifted versions of $h^{\max}$ and LM-cut (Wichlacz et al. 2023), and homomorphism-based heuristics (Horčík and Fišer 2021; Horčík, Fišer, and Torralba 2022; Horčík and Fišer 2023). Beyond these, there are also compilation approaches, such as the SAT-based LiSAT planner (Höller and Behnke 2022). Lifted heuristics bring their own challenges: they are often less informed than their grounded counterparts, and exact computation of some of them, such as the lifted versions of $h^{\max}$ and LM-cut, is significantly harder and has to be approximated (Wichlacz et al. 2023).

In this work, we tackle the problem of computing pattern database heuristics (PDBs) (Edelkamp 2001; Haslum et al. 2007; Rovner, Sievers, and Helmert 2019; Seipp 2019) in a lifted way. We present a method for computing lifted PDBs that is designed to closely match the heuristic quality of the grounded version, while having the potential to be more efficient to compute since it avoids grounding the entire task. We begin by selecting sets of state atoms and systematically compute all so-called *interesting* patterns up to a given size (Pommerening, Röger, and Helmert 2013; Pommerening et al. 2021). To do so, we lazily construct the causal graph backwards, starting from the goal atoms. We then compute the projection for each pattern, instantiating only the relevant state-altering action labels. Collections of PDB heuristics are admissibly aggregated with the canonical heuristic (Haslum et al. 2007).

We evaluate our method on an established hard-to-ground benchmark set (Corrêa et al. 2020), demonstrating the potential of lifted PDBs and pinpointing the next steps for making their computation more efficient. Our lifted PDBs also open the door to integrating these heuristics with cost partitioning methods, which are state-of-the-art for optimal grounded planning (Seipp, Keller, and Helmert 2017).

2 Background

PDDL is the most common planning formalism. It defines a planning task through predicates and action schemas (McDermott et al. 1998). We call this formalism *lifted*, since it allows variables in the definition of actions and atoms. For a broader overview of lifted classical planning we refer to the survey by Corrêa and De Giacomo (2024).

A *lifted planning task* is a tuple $\sqcap = \langle \mathcal{B}, \mathcal{V}, \mathcal{P}, \mathcal{A}, I, G \rangle$ that combines a *domain* and a *problem instance*. The domain fixes a set of predicate symbols $\mathcal{P}$ and a set of action schemas $\mathcal{A}$. The problem instance provides a set of objects $\mathcal{B}$, a set of variable symbols $\mathcal{V}$, an initial state I and a goal

condition G. An *atom* α is a predicate symbol p applied to a tuple of terms, where each term is either a variable symbol or an object. α is *ground* if all of its terms are objects, and *lifted* otherwise. We write $\mathcal{V}[\alpha]$ for the set of variables occurring in α, and $\mathcal{P}[\alpha]$ for the predicate symbol of α.

Each action schema $a \in \mathcal{A}$ is a pair $\langle pre(a), \mathit{eff}(a)\rangle$. $pre(a)$ is a set of atoms (the *preconditions*) and $\mathit{eff}(a)$ is a set of literals (the *effects*). We write $\mathit{eff}^+(a)$ for the set of atoms added by a and $\mathit{eff}^-(a)$ for the set of atoms deleted by a. The *initial state* I and the *goal condition* G are both sets of ground atoms.

A *substitution* σ is a mapping $\sigma\colon \mathcal{V} \cup \mathcal{B} \to \mathcal{V} \cup \mathcal{B}$ with $\sigma(o) = o$ for every object $o \in \mathcal{B}$. We call σ a *grounding* if $\sigma(v) \in \mathcal{B}$ for every variable $v \in \mathcal{V}$. We write $\alpha\sigma$ for the application of σ to an atom α, $A\sigma = \{\alpha\sigma \mid \alpha \in A\}$ for its extension to a set of atoms, and $a\sigma = \langle pre(a)\sigma, \mathit{eff}(a)\sigma\rangle$ for its extension to an action schema. A *ground action* is the result of applying a grounding to an action schema.

A lifted planning task is *grounded* by grounding the predicates $\mathcal{P}$ and the action schemas $\mathcal{A}$ with the objects $\mathcal{B}$. Grounding the predicates yields the set of ground atoms $\mathcal{C}$, and grounding the action schemas yields the set of ground actions $\mathcal{A}_G$. We write a grounded planning task as $\Pi = \langle \mathcal{C}, \mathcal{A}_G, I, G\rangle$. A *state* of Π is a subset of $\mathcal{C}$. A ground action $a \in \mathcal{A}_G$ is *applicable* in state s if $pre(a) \subseteq s$, and applying a yields the successor state $(s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$. A *plan* is a sequence of ground actions whose successive application transforms I into a state s with $G \subseteq s$. We assume unit-cost actions throughout.

Grounded Pattern Database Heuristics

The induced transition system $\mathcal{T}$ of Π can have up to $2^{|\mathcal{C}|}$ states, so enumerating it is infeasible. Pattern database (PDB) heuristics sidestep this by projecting Π syntactically, without ever materializing $\mathcal{T}$.

A *pattern* P is a set of ground atoms. It induces an *abstract transition system* $\mathcal{T}_P$, obtained by projecting the atoms in each action of Π onto P and keeping only those actions that have effects on atoms in P. Let $h^*_{\mathcal{T}_P}$ denote the optimal goal-distance function in $\mathcal{T}_P$, which maps each abstract state t to the cost of a cheapest path from t to an abstract goal state. Tabulating $h^*_{\mathcal{T}_P}$ for all abstract states yields a *pattern database (PDB) heuristic* h^{PDB}_P, which for a state s of the original task is defined as $h^{\text{PDB}}_P(s) = h^*_{\mathcal{T}_P}(s|_P)$, where $s|_P$ is the restriction of s to P (Edelkamp 2001).

Definition 1 (ground projected task)**.** *Given a grounded planning task $\Pi = \langle \mathcal{C}, \mathcal{A}_G, I, G\rangle$ and a pattern $P \subseteq \mathcal{C}$, P induces a* ground projected task $\Pi_P = \langle \mathcal{C}_P, \mathcal{A}_{G,P}, I_P, G_P\rangle$ *with $\mathcal{C}_P = P$, $\mathcal{A}_{G,P} = \bigcup_{a\in\mathcal{A}_G} \langle pre(a) \cap P, \mathit{eff}(a) \cap P\rangle$, $I_P = I \cap P$ and $G_P = G \cap P$.*

Two patterns P_1 and P_2 *conflict* if some action has effects on atoms in both P_1 and P_2. A set of non-conflicting patterns can be combined *additively* by summing the heuristic values of their PDBs. The *canonical heuristic* is the strongest such additive combination (Haslum et al. 2007): given a pattern collection $\mathcal{P}$, let M be the set of all maximal subsets of $\mathcal{P}$ that contain no conflicting patterns. Then

$$h^{\text{CAN}}(s) = \max_{\mathcal{P}'\in M} \sum_{P'\in\mathcal{P}'} h^{\text{PDB}}_{P'}(s).$$

Choosing good patterns is crucial for the performance of PDB heuristics. The most prominent selection methods rely on the notion of *interesting* patterns, which capture how each atom is relevant for the goal and for the other atoms in the pattern. Relevance is determined via the *causal graph*.

Definition 2 (causal graph)**.** *The causal graph of a grounded planning task Π is a directed graph $CG = (V, E)$, where V is the set of atoms in Π and, for any two distinct atoms $v, v' \in V$, $(v, v') \in E$ whenever some action has*

- *a precondition on v and an effect on v' (*pre-eff link*), or*
- *effects on both v and v' (*eff-eff link*).*

We say that v has a *causal link* to v' if there is a path of pre-eff links from v to v' (including the empty path when $v = v'$). Using the causal graph, we define interesting patterns (Pommerening, Röger, and Helmert 2013).

Definition 3 (interesting pattern)**.** *Pattern P is* interesting *if the subgraph $CG|_P$ of the causal graph induced by P satisfies the following two conditions (Pommerening, Röger, and Helmert 2013):*

- *$CG|_P$ is weakly connected, and*
- *from every node of $CG|_P$ there is a directed path of pre-eff links inside $CG|_P$ to a goal atom in P.*

3 Computing Lifted PDBs

We compute lifted PDBs by working with a *lifted projected task*. For a set of atoms P, we write $\mathcal{P}[P] = \{\mathcal{P}[p] \mid p \in P\}$ for the set of predicate symbols occurring in P. A predicate is *static* if it does not occur in the effect of any action schema in $\mathcal{A}$, and an atom is *static* if its predicate is static. We write $\mathit{Static}(\sqcap)$ for the set of static atoms of $\sqcap$.

Definition 4 (lifted projected task)**.** *Let $\sqcap = \langle \mathcal{B}, \mathcal{V}, \mathcal{P}, \mathcal{A}, I, G\rangle$ be a lifted planning task and let P be a set of ground atoms. Set $\hat{P} = P \cup \mathit{Static}(\sqcap)$. The* lifted projected task induced by $\hat{P}$ *is the lifted task $\Pi_{\hat{P}} = \langle \mathcal{B}_{\hat{P}}, \mathcal{V}_{\hat{P}}, \mathcal{P}_{\hat{P}}, \mathcal{A}_{\hat{P}}, I_{\hat{P}}, G_{\hat{P}}\rangle$ with $\mathcal{B}_{\hat{P}} = \mathcal{B}$, $\mathcal{V}_{\hat{P}} = \mathcal{V}$, $\mathcal{P}_{\hat{P}} = \mathcal{P}[\hat{P}]$ and*

$$\begin{aligned}
\mathcal{A}_{\hat{P}} = \bigcup_{a\in\mathcal{A}} \langle &\{\ell \in pre(a) \mid \mathcal{P}[\ell] \in \mathcal{P}[\hat{P}]\},\\
&\{\ell \in \mathit{eff}(a) \mid \mathcal{P}[\ell] \in \mathcal{P}[\hat{P}]\}\rangle,\\
I_{\hat{P}} = \{&\ell \in I \mid \mathcal{P}[\ell] \in \mathcal{P}[\hat{P}]\},\\
G_{\hat{P}} = \{&g \in G \mid \mathcal{P}[g] \in \mathcal{P}[\hat{P}]\}.
\end{aligned}$$

The abstract states of $\mathcal{T}_{\hat{P}}$ are the subsets of P; static atoms are fixed to their values in $I_{\hat{P}}$ and thus do not enlarge the abstract state space, but they prune substitutions violating static preconditions, both here and in the causal-graph construction below. A direct way to obtain $\mathcal{T}_{\hat{P}}$ is to run an off-the-shelf lifted successor generator on $\Pi_{\hat{P}}$, restricted to the pattern atoms. Modern generators cast successor generation as enumerating, for each action schema a and state s, all

groundings σ with $pre(a)\sigma \subseteq s$, which corresponds to a conjunctive query whose body atoms are the preconditions of a and whose distinguished variables are the free variables of a (Corrêa et al. 2020). Applying any such generator to $\Pi_{\hat{P}}$ exposes two scalability issues.

First, the generators enumerate all groundings of a, including those whose effect leaves every atom of $\hat{P}$ unchanged. These groundings correspond to self-loops in $\mathcal{T}_{\hat{P}}$. Their number is bounded by $|\mathcal{B}|^{|\mathcal{V}[a]|}$ and thus grows exponentially with the arity of a. Filtering self-loops out requires comparing the source state to its successor, information an enumeration-style generator does not expose. We avoid the enumeration by iterating over abstract state *pairs* $\langle s, s' \rangle$ instead and answering, for each pair, an existence question: is there a substitution σ that realizes the transition $s \xrightarrow{(a,\sigma)} s'$ in $\mathcal{T}_{\hat{P}}$? Because s' pins down the effect, this query has fewer free variables than the standard precondition query and never returns self-loop answers.

Second, action schemas may contain variables that are not bound by any atom in $\hat{P}$; we call these *irrelevant variables*. Atoms in $pre(a)$ that contain only irrelevant variables can be dropped without affecting which transitions are reachable in the abstract state space. Atoms containing *indirectly relevant* variables, which are those that co-occur in a precondition atom with a relevant variable, directly or transitively, cannot be pruned, as dropping their constraints may widen the set of applicable groundings. Our current implementation prunes all atoms outside $\mathcal{P}[\hat{P}]$ and so treats indirectly relevant variables as irrelevant. This is a relaxation: it may admit spurious transitions, leaving the resulting heuristic less informed but still admissible. In some domains no variables can be pruned this way and the scaling issues persist.

Algorithm 1 captures the core of this procedure: given a schema a and abstract states s, s', it returns the transitions $s \xrightarrow{(a,\sigma)} s'$ that are realized in $\mathcal{T}_{\hat{P}}$ by some substitution σ. We implement the existence check as a backtracking search: we first enumerate the substitutions σ that match $\mathit{eff}^+(a)$ to δ^+ and $\mathit{eff}^-(a)$ to δ^-,[1] then test whether any such σ extends to satisfy $pre(a)\sigma \subseteq s$. To build $\mathcal{T}_P$, we invoke this primitive on every pair $s, s' \in \mathcal{S}|_P$ and every schema $a \in \mathcal{A}$ whose predicates overlap with $\mathcal{P}[P]$; the remaining schemas cannot produce any non-self-loop transition in $\mathcal{T}_{\hat{P}}$ and are skipped. Once $\mathcal{T}_P$ is built, we run Dijkstra's algorithm backwards from the abstract goal states to obtain the optimal goal-distance function $h^*_{\mathcal{T}_P}$.

Algorithm 1: Transitions $s \xrightarrow{(a,\sigma)} s'$ realized by some σ.

Require: Action schema a, abstract states s and s'
1: $\delta^+ \leftarrow s' \setminus s$; $\delta^- \leftarrow s \setminus s'$
2: **return** $\{s \xrightarrow{(a,\sigma)} s' \mid \mathit{eff}^+(a)\sigma = \delta^+,\ \mathit{eff}^-(a)\sigma = \delta^-,\ pre(a)\sigma \subseteq s\}$

4 Generating Pattern Collections

We show next how to efficiently come up with informative pattern collections, which is crucial for search guidance.

Interesting Patterns

We adapt the systematic pattern generation of Pommerening, Röger, and Helmert (2013) to the lifted setting. For a goal atom g, let $\mathcal{P}^k_g$ denote the set of all interesting patterns of size at most k that contain g. The original procedure inspects the full causal graph, which presupposes a grounded task. Instead, we build only the part of the causal graph that can witness membership in $\mathcal{P}^k_g$, expanding it lazily backwards from g by computing substitutions that bind effects of action schemas to atoms already in the graph (Algorithm 2). Limiting the chaining of these substitutions to depth k yields the smallest subgraph that still contains every interesting pattern of size at most k rooted at g.

To reduce the number of substitutions considered, we only follow substitutions that respect the static preconditions of the action schema. We call an extension σ' of σ *statically consistent* if every static precondition atom of a holds under σ', i.e., $\{\ell \in pre(a) \mid \mathcal{P}[\ell] \in \mathit{Static}(\sqcap)\}\sigma' \subseteq \mathit{Static}(\sqcap)$.

By construction, every directed path along pre-eff links of length at most k that ends at a goal atom g induces an interesting pattern: its node set is weakly connected, and every node has a pre-eff path to g inside the pattern. We therefore enumerate all such prefixes for each $g \in G$, which yields every chain-shaped interesting pattern of size at most k rooted at a goal atom. For any k, this construction misses patterns whose goal atoms are connected only by eff-eff links. We leave enumerating such patterns for future work.

Algorithm 2: Lazy construction of the lifted causal graph.

Require: Lifted action schemas $\mathcal{A}$, goal atom g
$C \leftarrow [g]$; $CG \leftarrow$ graph with single node g
while C is not empty **do**
  $v \leftarrow C$.pop()
  for all (a, σ') with $a \in \mathcal{A}$, $v \in \mathit{eff}(a)\sigma'$ and σ' statically consistent **do**
    for all non-static $v' \in pre(a)\sigma'$ **do**
      add pre-eff link $v' \to v$ to CG
      if $v' \notin CG$ **then** C.push(v')

Canonical Heuristic

We aggregate our lifted PDBs into the canonical heuristic h^{CAN} following the grounded definition without modification (Haslum et al. 2007). For each pattern, we record the action labels that label any non-self-loop transition in its abstract transition system; two patterns conflict if their label sets intersect. We then build the *compatibility graph*, one node per pattern, one edge per non-conflicting pair, and enumerate its maximal cliques to obtain the additive subcollections that h^{CAN} maximizes over.

[1] This exact matching disregards groundings that leave pattern atoms unchanged and may miss transitions, threatening admissibility. Empirically, all our plan costs coincide with optimal costs proven by the admissible baselines.

Domain	LiSAT	$h^{\text{L-hmax}}$	$h^{\text{LMC-least}}$	$h^{\text{CAN}}_{\text{GR-sys1}}$-fdr	$h^{\text{CAN}}_{\text{GR-sys2}}$-fdr	$h^{\text{CAN}}_{\text{GR-sys1}}$-bin	$h^{\text{CAN}}_{\text{GR-sys2}}$-bin	h^{blind}	$h^{\text{CAN}}_{\text{L-sys1}}$	$h^{\text{CAN}}_{\text{L-sys2}}$
Blocksworld (40)	**40**	0	1	12	12	2	4	0	5	4
Childsnack (144)	**48**	2	2	9	6	7	6	2	2	2
Labyrinth (40)	0	**40**	0	1	3	28	29	1	1	1
Logistics (40)	**28**	5	0	13	18	0	0	5	5	2
Organic (56)	**55**	44	42	18	18	18	18	36	0	0
Pipesworld (50)	**18**	7	10	16	17	16	16	13	14	0
Rovers (40)	**3**	1	0	2	**3**	2	2	2	**3**	0
Visitall (180)	**98**	68	36	42	42	15	15	36	53	0
Total (590)	**290**	167	91	113	119	88	90	95	83	9

Table 1: Number of tasks solved by different lifted planners.

5 Experiments

We implemented our lifted PDB heuristics in the Tyr planner. We evaluate two configurations of lifted PDBs, $h^{\text{CAN}}_{\text{L-sys1}}$ and $h^{\text{CAN}}_{\text{L-sys2}}$, which use interesting patterns up to size 1 and 2, respectively. Our baselines consist of grounded PDB heuristics, common lifted heuristics and a lifted SAT planner.

- $h^{\text{CAN}}_{\text{GR-sys}\{1,2\}}$-{bin,fdr}: the canonical heuristic over grounded PDBs with interesting patterns up to size 1 and 2, respectively, implemented in Scorpion (Seipp, Keller, and Helmert 2020), an extension of Fast Downward (Helmert 2006). We use a binary variable encoding (bin)[2] or finite-domain representation (fdr) variables.
- $h^{\text{LMC-least}}$ and $h^{\text{L-hmax}}$: lifted LM-cut (in the LMC-least configuration) and lifted $h^{\max}$, both implemented in the minecraft-saar fork of Powerlifted (Wichlacz et al. 2023).
- h^{blind}: We use our own implementation in the Tyr planner.
- LiSAT: The LiSAT fork of the Powerlifted planner, using cryptominisat as the internal SAT solver (Höller and Behnke 2022; Soos, Nohl, and Castelluccia 2009).

We evaluate all planners on the unit-cost subset of the HTG benchmarks. All experiments were run on a compute cluster (Intel Xeon Gold 6130) with a 30 minute time limit and an 8 GiB memory limit.

Table 1 reports the number of tasks solved by each configuration. Among the heuristic-search baselines, $h^{\text{CAN}}_{\text{L-sys1}}$ is competitive but never dominant. In Visitall it trails only $h^{\text{L-hmax}}$ and outperforms $h^{\text{LMC-least}}$. This is also the only domain in which $h^{\text{CAN}}_{\text{L-sys1}}$ improves visibly over its grounded counterpart $h^{\text{CAN}}_{\text{GR-sys1}}$. LiSAT dominates every domain except Labyrinth, where only $h^{\text{L-hmax}}$ scales and solves every task. $h^{\text{CAN}}_{\text{L-sys2}}$ trails all other configurations and solves only nine tasks in total.

Figure 1 compares $h^{\text{CAN}}_{\text{L-sys1}}$ against $h^{\text{CAN}}_{\text{GR-sys1}}$. $h^{\text{CAN}}_{\text{L-sys1}}$ consistently expands more states (left), confirming that the lifted heuristic is less informed, but it is often faster per task (right), with speedups of up to one order of magnitude. Against $h^{\text{L-hmax}}$ and $h^{\text{LMC-least}}$ (Figure 2), $h^{\text{CAN}}_{\text{L-sys1}}$ is again faster, with speedups of up to two orders of magnitude. Some of this gap likely reflects differences between Tyr and Powerlifted as host planners, not just between the heuristics.

[2]We disable invariant synthesis and pruning of irrelevant atoms to obtain an equivalent task representation as for the lifted planner.

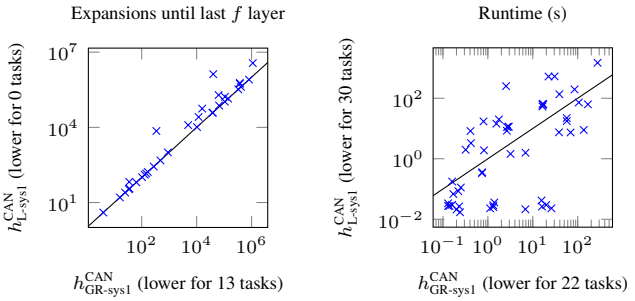

Figure 1: $h^{\text{CAN}}_{\text{L-sys1}}$ against its grounded counterpart: node expansions up to the last f-layer (left) and runtime (right).

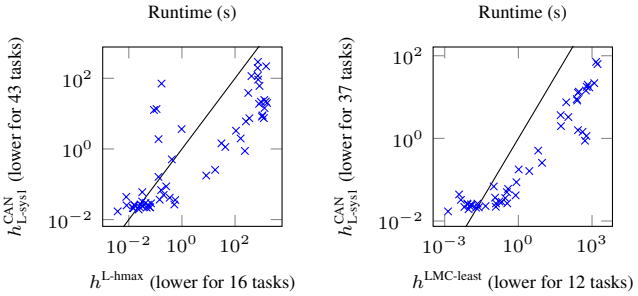

Figure 2: Runtime of $h^{\text{CAN}}_{\text{L-sys1}}$ against lifted hmax (left) and LM-cut (right).

6 Discussion

The total coverage of $h^{\text{CAN}}_{\text{L-sys1}}$ is modest, but the heuristic is not strictly dominated by any single baseline: for every baseline, at least one domain exists where $h^{\text{CAN}}_{\text{L-sys1}}$ matches or beats it. The runtime advantage we observe consistently offsets the weaker heuristic estimates that the lifted relaxation produces.

We see three avenues for closing the gap to the grounded canonical heuristic. First, the projected applicability check (Algorithm 1) can adopt the query-optimization techniques developed for full lifted successor generators (Corrêa et al. 2020). Second, the pattern-selection step could be aligned more closely with ground-level patterns of multi-valued variables by taking mutex information into account, e.g. from incorporating lifted mutex groups (Fišer 2023). Third, the admissible aggregation can move beyond the canonical heuristic to cost-partitioning methods such as post-hoc optimization (Pommerening, Röger, and Helmert 2013; Seipp, Keller, and Helmert 2021) and saturated cost partitioning (Seipp, Keller, and Helmert 2020), which yield stronger admissible estimates from the same set of PDBs.

7 Conclusions & Future Work

We presented an approach for computing pattern database heuristics in a lifted way. Empirically, $h^{\text{CAN}}_{\text{L-sys1}}$ is often faster per task than other admissible lifted heuristics, but its overall coverage and heuristic quality still lag behind the baselines. The clear next steps are to support non-unit action costs, incorporate lifted mutex information and reduce the overhead of computing projections. Together with the cost-partitioning extensions discussed above, these improvements should bring lifted PDBs closer to the state of the art for admissible lifted heuristics.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725. We thank Daniel Fišer for helpful conversations during the development of this paper, and the anonymous reviewers for their feedback, which improved the work.

References

Areces, C.; Bustos, F.; Dominguez, M. A.; and Hoffmann, J. 2014. Optimizing Planning Domains by Automatic Action Schema Splitting. In *Proc. ICAPS 2014*, 11–19.

Corrêa, A. B.; and De Giacomo, G. 2024. Lifted Planning: Recent Advances in Planning Using First-Order Representations. In *Proc. IJCAI 2024*, 8010–8019.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-Relaxation Heuristics for Lifted Classical Planning. In *Proc. ICAPS 2021*, 94–102.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation using Query Optimization Techniques. In *Proc. ICAPS 2020*, 80–89.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2022. The FF Heuristic for Lifted Classical Planning. In *Proc. AAAI 2022*, 9716–9723.

Edelkamp, S. 2001. Planning with Pattern Databases. In *Proc. ECP 2001*, 84–90.

Fišer, D. 2023. Operator Pruning Using Lifted Mutex Groups via Compilation on Lifted Level. In *Proc. ICAPS 2023*, 118–127.

Francès, G. 2017. *Effective Planning with Expressive Languages*. Ph.D. thesis, Universitat Pompeu Fabra.

Gnad, D.; Torralba, Á.; Domínguez, M. A.; Areces, C.; and Bustos, F. 2019. Learning How to Ground a Plan – Partial Grounding in Classical Planning. In *Proc. AAAI 2019*, 7602–7609.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proc. AAAI 2007*, 1007–1012.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26: 191–246.

Höller, D.; and Behnke, G. 2022. Encoding Lifted Classical Planning in Propositional Logic. In *Proc. ICAPS 2022*, 134–144.

Horčík, R.; and Fišer, D. 2021. Endomorphisms of Lifted Planning Problems. In *Proc. ICAPS 2021*, 174–183.

Horčík, R.; and Fišer, D. 2023. Gaifman Graphs in Lifted Planning. In *Proc. ECAI 2023*, 1052–1059.

Horčík, R.; Fišer, D.; and Torralba, Á. 2022. Homomorphisms of Lifted Planning Tasks: The Case for Delete-free Relaxation Heuristics. In *Proc. AAAI 2022*, 9767–9775.

Lauer, P.; Torralba, Á.; Fišer, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *Proc. IJCAI 2021*, 4119–4126.

Lauer, P.; Torralba, Á.; Höller, D.; and Hoffmann, J. 2025. Continuing the Quest for Polynomial Time Heuristics in PDDL Input Size: Tractable Cases for Lifted h^{add}. In *Proc. ICAPS 2025*, 74–83.

Masoumi, A.; Antoniazzi, M.; and Soutchanski, M. 2015. Modeling Organic Chemistry and Planning Organic Synthesis. In *Proc. GCAI 2015*, 176–195.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003, Yale University.

Pommerening, F.; Keller, T.; Halasi, V.; Seipp, J.; Sievers, S.; and Helmert, M. 2021. Dantzig-Wolfe Decomposition for Cost Partitioning. In *Proc. ICAPS 2021*, 271–280.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In *Proc. IJCAI 2013*, 2357–2364.

Rovner, A.; Sievers, S.; and Helmert, M. 2019. Counterexample-Guided Abstraction Refinement for Pattern Selection in Optimal Classical Planning. In *Proc. ICAPS 2019*, 362–367.

Seipp, J. 2019. Pattern Selection for Optimal Classical Planning with Saturated Cost Partitioning. In *Proc. IJCAI 2019*, 5621–5627.

Seipp, J.; Keller, T.; and Helmert, M. 2017. A Comparison of Cost Partitioning Algorithms for Optimal Classical Planning. In *Proc. ICAPS 2017*, 259–268.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *JAIR*, 67: 129–167.

Seipp, J.; Keller, T.; and Helmert, M. 2021. Saturated Posthoc Optimization for Classical Planning. In *Proc. AAAI 2021*, 11947–11953.

Soos, M.; Nohl, K.; and Castelluccia, C. 2009. Extending SAT Solvers to Cryptographic Problems. In *Proc. SAT 2009*, 244–257.

Ståhlberg, S. 2023. Lifted Successor Generation by Maximum Clique Enumeration. In *Proc. ECAI 2023*, 2194–2201.

Wichlacz, J.; Höller, D.; Fišer, D.; and Hoffmann, J. 2023. A Landmark-Cut Heuristic for Lifted Optimal Planning. In *Proc. ECAI 2023*, 2623–2630.

Wichlacz, J.; Höller, D.; and Hoffmann, J. 2022. Landmark Heuristics for Lifted Classical Planning. In *Proc. IJCAI 2022*, 4665–4671.

Wichlacz, J.; Torralba, Á.; and Hoffmann, J. 2019. Construction-Planning Models in Minecraft. In *Proc. HPlan 2019*.