Strengthening Canonical Pattern Databases with Structural Symmetries

Silvan Sievers and **Martin Wehrle** and **Malte Helmert**
University of Basel, Switzerland
{silvan.sievers,martin.wehrle,malte.helmert}@unibas.ch

Michael Katz
IBM Watson Health, Haifa, Israel
katzm@il.ibm.com

Abstract

Symmetry-based state space pruning techniques have proved to greatly improve heuristic search based classical planners. Similarly, abstraction heuristics in general and pattern databases in particular are key ingredients of such planners. However, only little work has dealt with how the abstraction heuristics behave under symmetries. In this work, we investigate the symmetry properties of the popular canonical pattern databases heuristic. Exploiting structural symmetries, we strengthen the canonical pattern databases by adding symmetric pattern databases, making the resulting heuristic invariant under structural symmetry, thus making it especially attractive for symmetry-based pruning search methods. Further, we prove that this heuristic is at least as informative as using symmetric lookups over the original heuristic. An experimental evaluation confirms these theoretical results.

Introduction

Heuristic search is a state-of-the-art approach to cost-optimal classical planning. There are two main components that speed up current planners based on heuristic search. The first one is informative admissible heuristics. Over the years, multiple admissible heuristic classes have been introduced for planning. One such class is abstraction heuristics, with a famous representative being the pattern database (PDB) heuristics (Culberson and Schaeffer 1998; Edelkamp 2001; Haslum et al. 2007). The second component is search space pruning techniques, with a prominent representative being symmetry-based pruning search algorithms (Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012; 2015). The approach is based on finding and exploiting state space automorphisms. These automorphisms essentially define symmetries between states, allowing to modify the search algorithms to take advantage of this information by pruning states that are symmetric to previously seen ones.

When using symmetry-based pruning in heuristic search, it is important to understand how the chosen heuristic interacts with symmetries. Shleyfman et al. (2015) perform such an investigation for all major planning heuristic classes, with the exception of abstraction heuristics. They show that many heuristics are either invariant under symmetry or can be easily adjusted to become such. This is an important result that affects the choice of the actual search algorithm. If the heuristic is not invariant under symmetry, the choice which of the (symmetric) search nodes to prune can greatly affect the outcome. In classical heuristic search, one popular technique to offset the consequences of this decision is symmetric lookups, also known as dual lookups, which compute the heuristic values also for the symmetric states (Felner et al. 2005; 2011). If the heuristic in use is invariant under symmetry, such symmetric lookups are of course not useful. Sievers et al. (2015a) investigated the effect of symmetric lookups for classical planning. In particular, they applied symmetric lookups to several abstraction heuristics, such as merge-and-shrink (Helmert et al. 2014; Sievers, Wehrle, and Helmert 2014), CEGAR (Seipp and Helmert 2014) and canonical pattern databases (CPDBs) (Haslum et al. 2007).

Another way to use information from symmetries is to exploit them to strengthen existing heuristics. Sievers et al. (2015b) use factored symmetries to enrich merge-and-shrink heuristics. We continue this line of work with the CPDB heuristic in this paper, exploiting structural symmetries (Shleyfman et al. 2015) to strengthen the CPDB heuristic by adding symmetric PDBs. As a result, we obtain a heuristic that is invariant under symmetry, making it appealing to be used in symmetry-based pruning search algorithms. Furthermore, the resulting heuristic is at least as informative as using symmetric lookups over the original CPDB heuristic. As the enhanced heuristic operates over a larger collection of PDBs, in order to reduce the memory consumption, we propose keeping symmetric PDBs implicitly, generalizing previously used domain-dependent techniques (Felner et al. 2005; Helmert and Röger 2010) to classical domain-independent planning. We perform an empirical investigation, showing the benefits of the suggested approach in terms of both increased informativeness and improved memory consumption for PDB storage.

Background

We consider planning tasks in the SAS$^+$ formalism (Bäckström and Nebel 1995), augmented with action costs. In this formalism, a planning task is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, where $\mathcal{V}$ is a finite set of finite *state variables* v, each associated with a domain $\mathcal{D}(v)$. A *partial state* s

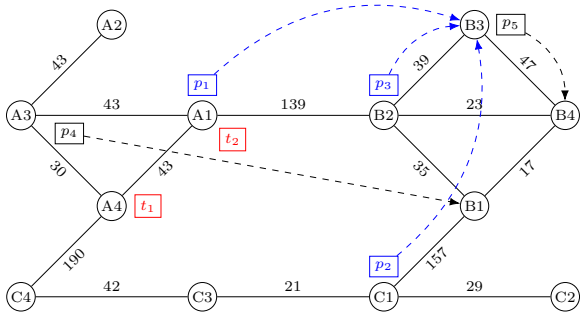

Figure 1: Instance #5 of the `transport-opt11` domain, with symmetric packages and trucks highlighted.

assigns each variable $v \in vars(s)$ a value from $\mathcal{D}(v)$, denoted $s[v]$, where $vars(s) \subseteq \mathcal{V}$. If $vars(s) = \mathcal{V}$, s is called a *state*. Two partial states s and s' *comply* if $s[v] = s'[v]$ for all $v \in vars(s) \cap vars(s')$. $\mathcal{O}$ is a finite set of operators o, each of the form $o = \langle pre(o), \mathit{eff}(o), cost(o)\rangle$, where both $pre(o)$ and $\mathit{eff}(o)$ are partial states and $cost(o) \in \mathbb{N}_0^+$ is the non-negative cost of the operator. s_0 is the *initial state* and $s_\star$ is the *goal* description, a partial state.

The semantics of a planning task is defined as follows. An operator $o \in \mathcal{O}$ is *applicable* in a state s if $pre(o)$ complies with s. Its application in s results in the state s', denoted $s(o)$, that complies with $\mathit{eff}(o)$ and for all $v \notin vars(\mathit{eff}(o))$, $s'[v] := s[v]$. An *s-plan* is a sequence of operators $\pi = \langle o_1, \ldots, o_{n-1}\rangle$ such that it is iteratively applicable starting in s and finally leads to some *goal state* s_n, i. e. a state that complies with $s_\star$. Formally, there must exist states $s_1, \ldots s_n$ with $s_1 = s$ such that o_i is applicable in s_i and $s_{i+1} = s_i(o_i)$. An s-plan is called a *plan* if $s = s_0$. The *cost* of such a plan π is the sum of its operators' cost, i. e. $cost(\pi) = \sum_{i=1}^{n} cost(o_i)$. An *optimal plan* is a plan with minimal cost among all plans. Optimal planning deals with finding optimal plans.

Figure 1 shows our running example (ignoring the colors for the moment), instance #5 of the TRANSPORT domain from the optimal sequential track of the International Planning Competition 2011. There are three cities A, B, and C, each with four locations $1, \ldots, 4$. Locations of cities are connected by roads of a certain length as shown in the figure. There are five packages $p_1, \ldots, p_5$, drawn at their initial locations, that must be delivered to the locations indicated by dotted arrows. To achieve this, there are two trucks t_1 and t_2, drawn at their initial locations, each with a capacity of carrying up to three packages. The planning task has three types of operators: PICK-UP and DROP for loading and unloading of packages in and from trucks, incurring cost of 1, and DRIVE for moving trucks between two locations if there is a road connecting them, incurring cost equal to the length of the road. A typical SAS$^+$ task uses five state variables v^{p_i} for the packages p_i, encoding at which location or in which truck a package is, and four state variables v^{t_i} and v^{c_i}, encoding the location and the available capacity of the trucks t_i, respectively.

For the remainder of this section, we assume that a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star\rangle$ with states S is given. A *heuristic* is a function $h : S \mapsto \mathbb{N}_0^+$ that assigns each state s an estimate of the cost-to-go to reach a goal state. The *perfect heuristic* h^* assigns each states s the true minimal cost of reaching a goal state. A heuristic h is *admissible* if it never overestimates the true cost, i. e. $h(s) \leq h^*(s)$ for all states s. Combining the A* search algorithm (Hart, Nilsson, and Raphael 1968) with an admissible heuristic results in finding optimal plans.

Pattern Databases

A *pattern database (PDB)* (Culberson and Schaeffer 1998) is a heuristic defined by a subset of the planning task's variables, $P \subseteq \mathcal{V}$, called the *pattern*. The pattern induces an *abstraction* of the original planning task by considering states equivalent iff they agree on all variables from the pattern, i. e. states s and s' are considered equivalent iff $s[v] = s'[v]$ for all $v \in P$. We note that for the case of SAS$^+$ tasks, the abstract planning task $\Pi^P = \langle P, \mathcal{O}^P, s_0^P, s_\star^P\rangle$ can be obtained from Π by simply syntactically removing all references to variables not contained in P. A PDB for pattern P, denoted h^P, stores perfect heuristic values for Π^P and a perfect hash function to map states s of Π to their abstract counterparts s^P of Π^P. PDBs are admissible heuristics due to the nature of the state space abstraction.

Heuristics in general and PDBs in particular are additive if their heuristic values can be summed without violating admissibility of the resulting heuristic for all states. Formally, a set of patterns (also called *pattern collection*) $\{P_1, \ldots, P_n\}$ for Π is *additive* (and hence the set of PDBs $\{h^{P_1}, \ldots, h^{P_n}\}$ is additive) if the heuristic $h(s) := \sum_{i=1}^{n} h^{P_i}(s)$ is admissible for all states $s \in S$.

A simple additivity criterion (a sufficient, but not necessary condition of additivity), has been presented by Haslum et al. (2007). Two patterns P and Q for Π are *disjoint-additive* if there is no operator $o \in \mathcal{O}$ such that variables $v \in P$ and $v' \in Q$ are both *affected* by o, i. e. $v, v' \in vars(\mathit{eff}(o))$. A set of patterns is disjoint-additive if all patterns of the set are pairwise disjoint-additive. Given a pattern collection C and the collection A of all maximal (w.r.t. set inclusion) disjoint-additive subsets of C, the *canonical PDB (CPDB) heuristic* (Haslum et al. 2007) for a state s is defined as

$$h^{\mathcal{C}_C}(s) = \max_{B \in A} h^B(s) = \max_{B \in A} \sum_{P \in B} h^P(s).$$

Informally, the CPDB heuristic computes the sum over PDBs whenever this is admissible, and the maximum otherwise. Note that this is the best way of admissibly combining the patterns in C for the disjoint additivity criterion.

Haslum et al. (2007) also presented a *hill climbing (HC)* procedure that performs a search in the space of pattern collections, aiming at obtaining pattern collections which yield the best results with the CPDB heuristic. In a nutshell, the HC procedure initializes the pattern collection with singleton patterns for all variables mentioned in the goal. It then iteratively considers adding new patterns to the collection that are one-variable extensions of patterns from the current collection (patterns are extended by adding one causally rel-

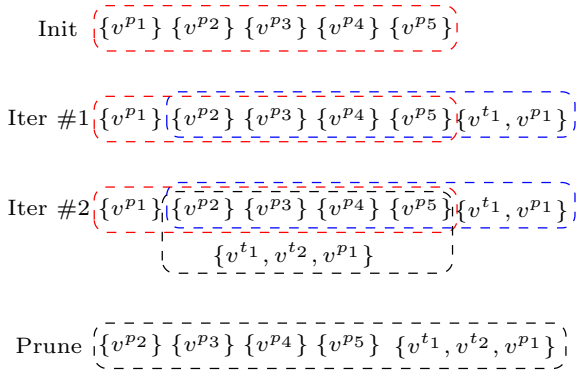

Figure 2: Several iterations of HC on instance #5 of the `transport-opt11` domain, showing the current pattern collections and their maximal disjoint-additive subsets.

evant variable to it). These candidate patterns are evaluated by computing the CPDB heuristic that would result from including the candidate pattern on sample states. The procedure stops if no significant improvement can be obtained or a time limit is reached. At the end, all patterns that are only part of maximal disjoint-additive subsets that are dominated by others are pruned from the collection. The combination of the CPDB heuristic with pattern collections obtained through HC is commonly denoted by *iPDB* in the literature.

Figure 2 shows two exemplary iterations of the HC procedure. At each step, it lists the current pattern collection and depicts the maximal disjoint-additive subsets by dashed boxes around the patterns they contain. The initial pattern collection contains all singleton patterns for goal variables. In the first iteration, an extension of $\{v^{p_1}\}$ with $\{v^{t_1}\}$ is added to the collection, and in the second iteration, $\{v^{t_1}, v^{t_2}, v^{p_1}\}$ is added, introducing new maximal disjoint-additive subsets. After pruning the dominated patterns $\{v^{p_1}\}$ and $\{v^{t_1}, v^{p_1}\}$ and the maximal disjoint-additive subsets they are part of in an optimization step, all remaining patterns are pairwise disjoint-additive, and hence a single maximal disjoint-additive subset remains, shown in black in the figure. A computation of the CPDB heuristic hence adds all heuristic values of the PDBs for the individual patterns, i. e. $h^{\mathcal{C}_C}(s) = h^{\{v^{p_2}\}}(s) + h^{\{v^{p_3}\}}(s) + h^{\{v^{p_4}\}}(s) + h^{\{v^{p_5}\}}(s) + h^{\{v^{t_1}, v^{t_2}, v^{p_1}\}}(s)$.

Structural Symmetries

Shleyfman et al. (2015) defined structural symmetries for STRIPS planning tasks. Later, the definition was adapted to SAS$^+$ for Fully Observable Non-deterministic Planning (Winterer, Wehrle, and Katz 2016). Here, we restrict the definition of Winterer, Wehrle, and Katz (2016) to the classical setting.

Definition 1 (Structural Symmetry)**.** *For a* SAS$^+$ *planning task* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$*, let* F *be the set of* Π*'s facts, i. e., pairs* $\langle v, d \rangle$ *with* $v \in \mathcal{V}$*,* $d \in \mathcal{D}(v)$*. A* structural symmetry *for* Π *is a permutation* $\sigma : \mathcal{V} \cup F \cup \mathcal{O} \to \mathcal{V} \cup F \cup \mathcal{O}$*, where*

1. $\sigma(\mathcal{V}) = \mathcal{V}$ *and* $\sigma(F) = F$ *such that* $\sigma(\langle v, d \rangle) = \langle v', d' \rangle$ *implies* $v' = \sigma(v)$*;*
2. $\sigma(\mathcal{O}) = \mathcal{O}$ *such that for* $o \in \mathcal{O}$*,* $\sigma(pre(o)) = pre(\sigma(o))$*,* $\sigma(\mathit{eff}(o)) = \mathit{eff}(\sigma(o))$*,* $cost(\sigma(o)) = cost(o)$*;*
3. $\sigma(s_\star) = s_\star$*;*

where $\sigma(\{x_1, \ldots, x_n\}) := \{\sigma(x_1), \ldots, \sigma(x_n)\}$*, and for a partial state* s*,* $s' := \sigma(s)$ *is the partial state obtained from* s *such that for all* $v \in vars(s)$*,* $\sigma(\langle v, s[v] \rangle) = \langle v', d' \rangle$ *implies* $s'[v'] = d'$*.*

Note that given a structural symmetry σ, its application to sets or tuples X is naturally defined as the set/tuple of element-wise applications of σ. The set of all structural symmetries Γ_Π of a planning task Π forms a group under the composition operation. In practice, a set of structural symmetries that generates (a subgroup of) the symmetry group Γ_Π can be efficiently computed using off-the-shelf tools for discovery of automorphisms in explicit graphs (Shleyfman et al. 2015). For simplicity, in what follows, by a symmetry group Γ we refer to a subgroup of the symmetry group Γ_Π of the planning task Π.

In the running example shown in Figure 1, we find a set of symmetry generators Σ consisting of three elements. One generator permutes the variables v^{p_1} and v^{p_2}, another one the variables v^{p_2} and v^{p_3}, which together cover the symmetries between the packages p_1, p_2, and p_3 that all need to be delivered to the same goal location (highlighted in blue in the figure).[1] The third generator permutes the variables of the trucks, i. e. it swaps v^{t_1} with v^{t_2}, and v^{c_1} with v^{c_2}, hence covering the symmetry between the trucks (highlighted in red in the figure). By composing these three generators, we obtain a symmetry group of the planning task.

Symmetry-based pruning search algorithms use symmetries to prune some of the symmetric states encountered during search (if previously seen symmetric states have been reached with the same or lower cost). *DKS* (Domshlak, Katz, and Shleyfman 2012) is such an algorithm. It runs A* and performs additional duplicate pruning with structural symmetries. This preserves optimality because due to the structure-preserving property of goal-stable automorphisms, a plan from state s exists iff the plan under symmetry (hence of the same cost) exists from the symmetric state s'. In our running example, any two states that only differ in that the positions of p_1 and p_2 are swapped are symmetric under the first symmetry generator and hence it is enough to consider only one of the two states in a forward search.

Symmetric lookups for classical planning (Sievers et al. 2015a) are a technique to exploit (structural) symmetries during search such as A*. For a given heuristic h, a state s and a symmetry group Γ, the *symmetric lookup heuristic over* h is defined as $h_{SL}(s) := \max_{s \in S} h(s)$, where $S := \{s, s^1, \ldots, s^m\}$ is a set of states symmetric to s under structural symmetries from Γ, including s itself. S can be chosen arbitrarily to trade off computation time against informativeness of the symmetric lookups, i. e. $m = 0$ is

[1] Domshlak, Katz, and Shleyfman (2012) showed that for use in a forward search, structural symmetries do not need to stabilize the initial state.

possible as well as computing the set of all states symmetric to s under Γ.

Symmetric Patterns and Implicit PDBs

Our method to enhance the CPDB heuristic is based on computing symmetric patterns of the pattern collection obtained via HC and adding the PDBs for these symmetric patterns to the CPDB heuristic.

Definition 2 (Symmetric Patterns)**.** *Given a pattern $P = \{v_1, \ldots, v_n\}$, the* symmetric pattern *under structural symmetry σ is defined as $\sigma(P) = \{\sigma(v_1), \ldots, \sigma(v_n)\}$.*

Our first result establishes that a symmetric PDB has the same heuristic values as the original PDB for all states under the mapping of the symmetry.

Theorem 1. *Let Π be a* SAS$^+$ *planning task, P be a pattern, and σ be a structural symmetry of Π. For each state s of Π we have $h^P(s) = h^{\sigma(P)}(\sigma(s))$.*

Proof. Let $Q = \sigma(P)$ and let Π^P and Π^Q be the abstract planning tasks. Note that σ maps Π^P to Π^Q, mapping variables, operators, and the goal. Let s^P be the partial state obtained from s by restricting s to the variables in P. Then s^P is a state in Π^P. Similarly, let t^Q be the partial state obtained from $t := \sigma(s)$ by restricting t to the variables in Q. Then t^Q is a state in Π^Q. Further, note that $\sigma(s^P) = t^Q$, i. e., σ maps s^P to t^Q. Since σ is a structural symmetry, there is a 1:1 correspondence between the paths from s and t in the original state space of Π. As Π^P and Π^Q are both abstractions of the same state space (that of Π), abstract paths from s^P and t^P correspond to paths from s and t, and hence there is also a 1:1 correspondence between these paths in the abstractions, giving us the desired result. □

Based on this result, which is in the spirit of symmetric lookups, we suggest the following *implicit representation* of symmetric PDBs that avoids to compute the actual PDB. For a pattern P and a symmetric pattern Q such that $\sigma(Q) = P$ for some structural symmetry σ, instead of storing the PDBs (i. e. computing the abstract state distances) for both P and Q, we can compute the PDB for P and only keep the tuple $\langle h^P, \sigma \rangle$ as an implicit representation of the PDB for Q. When computing a heuristic value for state s with the symmetric PDB for Q, we can exploit Theorem 1 and reduce this computation to a lookup in the PDB for P by the following computation: $h^Q(s) = h^P(\sigma(s))$.

To make the computation of the lookup in the symmetric PDB more efficient, we do not need to permute the entire state s, but only the partial state s^Q, i. e. the part of s relevant to Q. Hence it is enough to store the part of σ relevant to Q, which allows us to map s^Q to the partial state $\sigma(s)^P$, i. e. the symmetric partial state relevant to P. Then we can look up the heuristic value in h^P and return it. While this still incurs a slight runtime overhead when computing heuristic values compared to lookups in a fully computed PDB, the computation time required to compute the full symmetric PDB and the memory required to store it can be avoided.

Canonical PDBs and Structural Symmetries

We now turn our attention to dealing with pattern collections as used by the CPDB heuristic. Our first definition, however, is independent of the way the pattern collection is obtained.

Definition 3. *Given a symmetry group Γ, a pattern collection C is* closed under symmetry group Γ *if for all structural symmetries $\sigma \in \Gamma$ and for all patterns $P \in C$, $\sigma(P) \in C$.*

Informally, the pattern collection is closed under symmetry if all symmetric patterns of all patterns are already part of the collection. Naturally, not all collections are closed under symmetry. Given a collection C that is not closed under Γ, adding all symmetric patterns to the collection results in the *symmetric closure* $\overline{C}$ that is closed under symmetry group Γ.

From here on, we focus on pattern collections that are disjoint-additive, as required by the CPDB heuristic.

Theorem 2. *Given a structural symmetry σ and a disjoint-additive pattern collection C, the pattern collection $\sigma(C)$ is disjoint-additive.*

Proof. Let P and Q be two patterns in C. Since C is disjoint-additive, for all operators $o \in \mathcal{O}$ we have $vars(\mathit{eff}(o)) \cap P = \emptyset$ or $vars(\mathit{eff}(o)) \cap Q = \emptyset$, and thus $vars(\mathit{eff}(\sigma(o))) \cap \sigma(P) = \emptyset$ or $vars(\mathit{eff}(\sigma(o))) \cap \sigma(Q) = \emptyset$ because σ is a structural symmetry. Hence $\sigma(P)$ and $\sigma(Q)$ are disjoint-additive, and since this holds for any pair of patterns from C, also $\sigma(C)$ is disjoint-additive. □

With this result, we can now state that the CPDB heuristic is invariant under a given symmetry group if used with a pattern collection that is closed under the symmetry group.

Theorem 3. *Given a symmetry group Γ and a pattern collection C, if C is closed under Γ, then for each state s and for each structural symmetry $\sigma \in \Gamma$, we have $h^{\mathcal{C}_C}(s) = h^{\mathcal{C}_C}(\sigma(s))$.*

Proof. Let A be a maximal disjoint-additive subset of C. Then, from Theorem 2 we have that $\sigma(A)$ is also disjoint-additive. Further, by Definition 3, since C is closed under Γ, we have $\sigma(A) \subseteq C$. Assume to the contrary of maximality of $\sigma(A)$ that there exists a pattern P in $C \setminus \sigma(A)$, such that $\sigma(A) \cup \{P\}$ is disjoint-additive. Then, from Theorem 2, $A \cup \{Q\}$ for some Q such that $\sigma(Q) = P$ is also disjoint-additive. Further, $Q \notin A$, since $P \notin \sigma(A)$, contradicting the maximality of A. □

Heuristics that are invariant under symmetry are particularly attractive for search techniques that use structural symmetries for pruning such as DKS. DKS prunes a search node if the state s of the node is symmetric to the state s' of a previously seen node. If the heuristic in use is invariant under symmetry, then the search effort from these two states onward is the same, whilst if the heuristic is not invariant under symmetry, it might be beneficial to continue with the state s instead of pruning it and relying on s'.

Another direct consequence of Theorem 3 is that there is no theoretical added value in performing symmetric lookups over the CPDB heuristic with pattern collections that are closed under symmetry. In fact, in general the symmetric

lookups heuristic over the CPDB heuristic with C is dominated by the CPDB heuristic with the symmetric closure $\overline{C}$, as the next theorem shows.

Theorem 4. *Given a symmetry group Γ and a pattern collection C, for each state s, $h^{\mathcal{C}_C}_{SL}(s) \leq h^{\mathcal{C}_{\overline{C}}}(s)$.*

Proof. Since $C \subseteq \overline{C}$, we have $h^{\mathcal{C}_C}(s') \leq h^{\mathcal{C}_{\overline{C}}}(s')$ for all states s'. In particular, it holds for each $s' = \sigma(s)$ for some $\sigma \in \Gamma$. From Theorem 3 we have that $h^{\mathcal{C}_{\overline{C}}}(s) = h^{\mathcal{C}_{\overline{C}}}(s')$ for all $s' = \sigma(s)$, and thus $h^{\mathcal{C}_C}(\sigma(s)) \leq h^{\mathcal{C}_{\overline{C}}}(s)$ for all $\sigma \in \Gamma$, giving us the desired result. □

Implementation

Building on the theoretical results of the previous sections, we present the following approach to enhance the CPDB heuristic through using structural symmetries. The algorithm begins with computing symmetries of the given planning task. In practice, a symmetry group Γ is usually not given explicitly as a collection of its elements (the symmetry group Γ_Π of a task Π is not known to be polynomially computable), but rather via a set of *symmetry generators* Σ that span the group Γ. Such symmetry generators can be computed in low-order polynomial time in the size of the planning task with off-the-shelf tools for discovery of automorphisms in explicit graphs.[2]

The algorithm then continues with the computation of a pattern collection C with the HC procedure as usual and then turns this collection C into the symmetric closure $\overline{C}$. Afterwards, it prunes patterns from dominated maximal disjoint-additive subsets as usual (to avoid storing unnecessary PDBs and performing unnecessary heuristic computations also for the symmetric PDBs), and then computes the PDBs of the patterns of the final pattern collection for the CPDB heuristic, possibly using the implicit representation for PDBs. From Theorem 3, we know that the resulting heuristic is invariant under symmetry. Additionally, from Theorem 4 we also know that this approach is at least as good using symmetric lookups with the CPDB heuristic on the original collection C.

Besides using HC and computing PDBs, the main ingredient of our algorithm is the computation of the symmetric closure $\overline{C}$ given a pattern collection C. Performing a complete breadth-first search in the space of symmetric patterns, we can compute $\overline{C}$ from C for any given pattern collection C, independent of its origin. The open list of the search is initialized with all patterns from C. Expanding a pattern P consists in applying each structural symmetry σ from Σ to P once, adding the symmetric patterns $\sigma(P)$ to the open list. After expansion, P is added to the closed list to avoid generating duplicates. The search runs until the open list is empty, at which point it generated all symmetric patterns for all $P \in C$ (i. e. applying all symmetries of the group Γ given implicitly through the generators Σ to all patterns). While the the runtime of this algorithm is exponential in the variables $\mathcal{V}$ of the planning task in the worst case, the computation is very fast in practice.

The basic variant of our approach where we compute full PDBs for the entire pattern collection $\overline{C}$ is called HC-CPDB-symm. The alternative is to compute implicit PDBs for all symmetric patterns added to C, i. e. for patterns in $\overline{C} \setminus C$. This requires to compute full PDBs for all patterns in C and to adapt the above algorithm to not only generate the symmetric patterns, but also the symmetry mappings from the symmetric patterns back to their original patterns (in C). We call this approach HC-CPDBS-symm-impl.

We note that the original pattern collection might already include some patterns that are symmetric to each other and further savings might be obtained by keeping the PDBs of these patterns implicitly as well. However, detecting these symmetry relations amongst patterns and deciding which of the PDBs to keep implicit is computationally expensive, and hence in this work, we restrict the use of implicit PDBs to only the newly generated symmetric patterns.

Experiments

In this section, we evaluate our approach which we implemented in Fast Downward (Helmert 2006). Our benchmark set comprises the planning domains of the optimal sequential tracks of all International Planning Competitions (IPCs) up to 2014, which gives rise to 1667 tasks in 57 domains. The experiments were run on machines with Intel Xeon E5-2660 CPUs running at 2.2 GHz, with each run limited to 30 minutes and 2GB of memory. All configurations use a time limit of 900s for the hill climbing procedure HC as suggested by Scherrer, Pommerening, and Wehrle (2015), and the (Fast Downward) default maximum sizes of 2000000 states for each PDB and 20000000 states for all PDBs in total.[3] Symmetries are computed with the graph automorphism tool Bliss (Junttila and Kaski 2007) as described by e. g. Shleyfman et al. (2015), and the total time budget for all computations related to symmetries is 300s. In all experiments, when not using the DKS search algorithm for symmetry-based pruning, we use A*. In both cases, when reporting the number of expansions, we always report expansions until last f-layer to avoid tie-breaking issues, aggregated over commonly solved tasks. The runtimes displayed in the tables are in seconds, averaged over commonly solved tasks either by using the geometric mean (gm) or the arithmetic mean (am). Best results are highlighted in bold.

A* Search

We begin our evaluation with using A* to eliminate the influence of symmetry-based pruning of the DKS algorithm which we evaluate afterwards. We compare the original heuristic HC-CPDB (corresponding to iPDB (Haslum et al.

[2] We only need to consider generators σ that do not stabilize variables, i. e. for which $\sigma(v) \neq v$ for at least one variable v, otherwise we would have $\sigma(P) = P$ for any pattern P, and hence identical perfect heuristic values for both PDBs (that also means if values of variables are permuted, the heuristic value cannot change).

[3] We also experimented with (much) smaller and somewhat larger size limits to investigate their potential influence, but found no change in the relative performance of the different heuristics, and also the absolute performance only changed marginally (maximum change of 6 solved tasks). Hence we only report results for the default size limits as used in Fast Downward.

	HC-CPDB			
	orig	**symm**	**symm-impl**	**SL**
Coverage (# solved tasks)	**814**	813	813	809
Expansions 85th percentile	513000	**429290**	**429290**	**429290**
Expansions 90th percentile	926373	**880093**	**880093**	909015
Expansions 95th percentile	3378274	**2661710**	**2661710**	2698737
Search out of memory	774	736	730	**483**
Search out of time	**70**	109	115	366
Search time (gm)	0.43	**0.42**	0.43	0.82
Total time (gm)	4.10	4.14	**4.08**	5.79
Symmetric PDBs time (gm)	-	0.00	**0.00**	-
Symmetric PDBs time (am)	-	3.02	**0.00**	-

Table 1: A* with the HC-CPDB heuristic in different variants: the original heuristic, the heuristic enhanced with symmetric PDBs (full PDBs and implicit PDBs), and using symmetric lookups over all symmetric states.

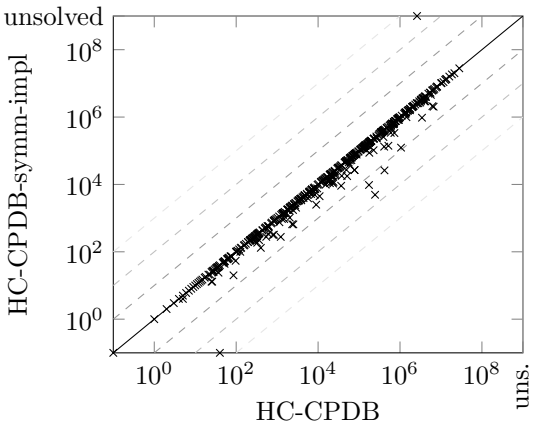

Figure 3: Expanded states for HC-CPDB vs HC-CPDB-symm-impl.

2007) as implemented in Fast Downward (Sievers, Ortlieb, and Helmert 2012)) to the two variants of our approach, i. e. HC-CPDB-symm and HC-CPDB-symm-impl. Furthermore, to test the practical implications of Theorem 4, we include the combination of the CPDB heuristic with symmetric lookups, denoted HC-CPDB-SL (Sievers et al. 2015a), where we compute the set of *all symmetric states* for a given state, using a similar complete breadth-first search as to compute all symmetric patterns.[4] Table 1 shows coverage, number of expansions (different percentiles over commonly solved tasks), the number of tasks for which the search hit the memory and time limits, the search time, the total time (which in contrast to search time includes the time to compute symmetries and the time to perform HC), and the time required to compute the symmetric closure of the pattern collection and the PDBs for the additional symmetric patterns (also included in total time but not in search time).

The first observation we make is that compared to the baseline, coverage increases with neither of the approaches of using symmetries. In particular, using symmetric lookups, coverage decreases slightly, as already noted by Sievers et al. (2015a). While using symmetric lookups only moderately decreases the number of expansions required, our approaches require the fewest expansions. In fact, as can be seen from the scatter plot shown in Figure 3, our approach HC-CPDB-symm-impl (the same as HC-CPDB-symm) dominates HC-CPDB, requiring strictly fewer expansions (in 194 tasks across 33 domains). While not shown, the same is true for HC-CPDB-symm(-impl) compared to HC-CPDB-SL, hence indeed confirming Theorem 4. Table 2 shows the summed expansions of all four variants for the 33 domains in which these variants require different amounts of expansions. Differences between HC-CPDB-symm and HC-CPDB-symm-impl are due to the HC procedure hitting the time limit in different iterations.

[4]Sievers et al. (2015a) instead report results for a set of 10 randomly generated symmetric states, but this configuration solves 5 tasks less than using the full set in our experiments.

	HC-CPDB			
	orig	**symm**	**symm-impl**	**SL**
BARMAN-OPT11	16836883	**15368212**	**15368212**	15368292
DEPOT	1445907	**1345976**	**1345976**	1355988
DRIVERLOG	2958328	**2683812**	**2683812**	2695195
ELEVATORS-OPT08	6314143	**6212113**	**6212113**	**6212113**
ELEVATORS-OPT11	5311976	**5209946**	**5209946**	**5209946**
GED-OPT14	27170082	**9065556**	**9065556**	10038602
GRIPPER	12533069	**12532583**	**12532583**	12532801
LOGISTICS00	218153	**217441**	**217441**	**217441**
LOGISTICS98	286292	**259265**	**259265**	267515
MICONIC	90265550	**88049903**	**88049903**	88134507
MPRIME	1229963	**1109620**	**1109620**	1229639
MYSTERY	1455463	**1443841**	**1443841**	1455250
NOMYSTERY-OPT11	11935954	**8383343**	**8383343**	8493587
OPENSTACKS	747591	**746430**	**746430**	746542
PEGSOL-08	1150308	466075	**465372**	538683
PEGSOL-OPT11	1363689	676404	**669728**	750551
PIPESWORLD-NOTANK.	33275583	**33235577**	**33235577**	33266046
PIPESWORLD-TANKAGE	8574679	**8514265**	**8514265**	**8514265**
PSR-SMALL	5155732	**5155718**	**5155718**	**5155718**
SCANALYZER-08	17195126	**17035042**	**17035042**	17105413
SCANALYZER-OPT11	17195120	**17035036**	**17035036**	17105407
SOKOBAN-OPT08	18173615	**14928222**	**14928222**	15142916
SOKOBAN-OPT11	5132629	**4542038**	**4542038**	4555111
STORAGE	5988362	**5665472**	**5665472**	5926753
TETRIS-OPT14	639332	**524044**	**524044**	635909
TIDYBOT-OPT11	2197242	**2020309**	**2020309**	2163974
TIDYBOT-OPT14	3824249	**3653359**	**3653359**	3791913
TPP	4138020	**4137980**	**4137980**	4137990
TRANSPORT-OPT08	1293570	**1052443**	**1052443**	**1052443**
TRANSPORT-OPT11	1291983	**1050856**	**1050856**	**1050856**
TRANSPORT-OPT14	13063876	**13062044**	**13062044**	**13062044**
TRUCKS	14194832	**13379703**	**13379703**	**13379703**
ZENOTRAVEL	1518693	**1517833**	**1517833**	1517895

Table 2: Expansions summed for each domain where the configurations of Table 1 require a different amount of expansions.

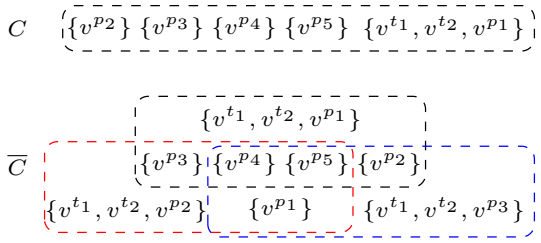

Figure 4: Computing the symmetric closure $\overline{C}$ from C as obtained by HC on instance #5 of the `transport-opt11` domain, including dominance pruning; showing the pattern collections and their maximal disjoint-additive subsets.

Coming back to our example from Figure 1, the baseline (HC-CPDB) computes the pattern collection as shown in the example computation in Figure 2. This pattern collection C yields a heuristic value of $h^{\mathcal{C}_C}(s_0) = 2+2+2+2+180 = 188$ for the initial state, while the optimal plan cost is 614. Note that all PDBs for packages not at a goal contribute exactly 2 to the heuristic value, for loading and unloading the package. The PDB with the two truck variables (in addition to a package variable) computes a value of $139+39+2$, for driving t_2 twice and loading and unloading p_1.

Our approach HC-CPDB-symm continues from this pattern collection C as illustrated in Figure 4. To obtain the symmetric closure of $\overline{C}$ from C, the symmetric patterns are added to C as shown. This results in two additional maximal disjoint-additive subsets, shown in red and blue in the figure. The maximal disjoint-additive subset that groups all singleton PDBs for each p_i is dominated by all others and pruned (not shown). With $\overline{C}$, the CPDB heuristic computes the maximum over the black, red, and blue maximal disjoint-additive subsets, e. g. for the initial state, $h^{\mathcal{C}_{\overline{C}}}(s_0) = \max\{180+2+2+2+2, 476+2+2+2+2, 180+2+2+2+2\} = 484$, a considerable increase, due to the addition of the pattern $\{v^{t_1}, v^{t_2}, v^{p_2}\}$. This exemplarily explains the fewer expansions required to solve this task with HC-CPDB-symm (c. f. Table 2).

In principle, the HC procedure could find this "better" pattern directly, but this is very unlikely to happen for the following reasons. Only variables of trucks (location and capacity) are causally relevant to the variables of packages. Hence the only candidate variables for extending patterns are the variables of trucks. In particular, because there is at most one variable of a package in each pattern for the same reason, adding the capacity variable of any truck can never improve the heuristic. To summarize, the only possibility of extending patterns is to add the location variable of a truck. However, adding a single location variable of a truck to any of the initial singleton patterns only increases the maximum heuristic value of the PDB from 2 to 3. This increase in heuristic in quality is high when the first pattern containing a variable of a truck is added, but after the collection contains the pattern $\{v^{t_1}, v^{t_2}, v^{p_1}\}$, extending any of the other patterns does not yield a notable heuristic improvement. Only adding the location variables of both trucks simultaneously would allow the HC procedure to find the "better" pattern

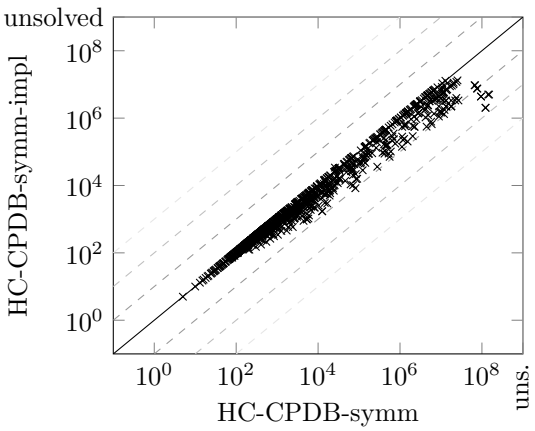

Figure 5: Estimated number of integers required to be stored for symmetric PDBs, comparing using full against implicit PDBs.

we get via adding symmetric patterns.

Going back to the overview shown in Table 1, comparing the reasons of failure of search for the baseline and our approaches, we observe a trade-off, i. e. our approaches hit the time limit more frequently and the memory limit less frequently than the baseline. In general, although the number of expansions decreases, for the majority of tasks, both search time and total time are very comparable for the baseline and our approaches. The runtime does not decrease although the expansions decrease because the number of PDB lookups required by the CPDB heuristic on the larger symmetric closures of pattern collections increases considerably. In particular, for tasks where adding symmetric patterns results in many new maximal disjoint-additive subsets of the pattern collection, the number of PDB lookups required to compute heuristic values with the CPDB heuristic can increase by up to two orders of magnitude. This is prohibitively large and results in the search not completing within the 30 minute time limit. An example task where this happens is the instance 11-1 of the LOGISTICS00 domain, which is solved by the baseline but not by our approaches.

Comparing computing full symmetric PDBs against computing implicit symmetric PDBs, we note that as expected, search time is slightly higher with implicit PDBs due to the additional computation required to map the abstract state from the implicit PDB to the original full PDB. Still, the total runtime is lower. One possible reason for this is the time required to compute symmetric patterns and PDBs. Looking at these runtimes, we first note that the overhead caused by this computation is in general negligible (geometric mean rounded to 0 seconds). Second, looking at the arithmetic mean, we see that computing full PDBs requires more time compared to computing implicit PDBs. We also assess the memory consumption of both our approaches. As an estimate of memory consumption, Figure 5 shows a scatter plot comparing the number of integers required to store full PDBs and implicit PDBs. As expected, using implicit PDBs strictly saves memory compared to storing full PDBs.

	HC-CPDB with DKS			
	orig	**symm**	**symm-impl**	**SL**
Coverage (# solved tasks)	887	**893**	891	886
Expansions 85th percentile	379185	**341430**	**341430**	351576
Expansions 90th percentile	819599	**788324**	**788324**	816721
Expansions 95th percentile	3510224	**2584593**	**2584593**	2745883
Search out of memory	490	475	472	**336**
Search out of time	**281**	290	294	436
Search time (gm)	0.56	**0.54**	0.54	0.86
Total time (gm)	5.33	5.37	**5.22**	6.60

Table 3: The DKS search algorithm with the HC-CPDB heuristic in different variants: the original heuristic, the heuristic enhanced with symmetric PDBs (full PDBs and implicit PDBs), and using symmetric lookups over all symmetric states.

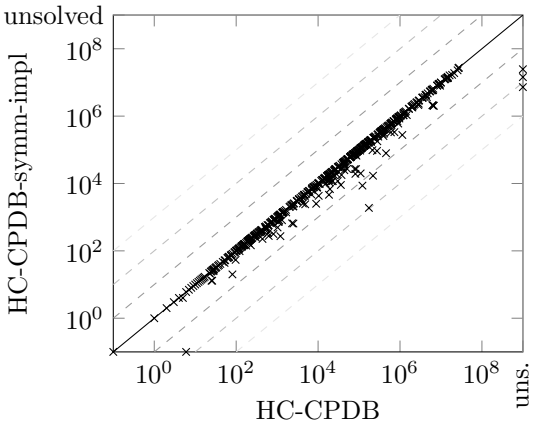

Figure 6: Expanded states for HC-CPDB vs HC-CPDB-symm-impl, both combined with DKS.

Symmetry-based Pruning with DKS

After having established that our approach increases heuristic informativeness, we now evaluate our improved CPDB heuristic, which is invariant under symmetries, in conjunction with the symmetry-based pruning search algorithm DKS. Table 3 shows the results for the same comparison as Table 1.

While using symmetric lookups again only helps in terms of expansions but not in terms of coverage, we observe that using our symmetry-improved CPDB heuristics indeed increase coverage compared to the original CPDB heuristic if combined with the DKS algorithm. The difference in coverage between using full PDBs and implicit PDBS is due to two tasks where the increased time required to compute heuristic values with implicit PDBs is too large. Figure 6 compares expansions, confirming the aggregated results shown in the table. Note that here as well, our improved heuristic strictly dominates the original one in terms of expansions. Concerning the number of times the search reaches the time or memory limit, and search and total time, the results are very similar to the ones with regular A*. We conclude that as suspected, using the now invariant-under-symmetry heuristic HC-CPDB-symm helps improving the performance of the symmetry-based pruning search algorithm DKS.

Conclusions

In this work, we applied structural symmetries to strengthen the canonical pattern databases heuristic. In particular, our approach computes symmetric patterns of the pattern collection used with the canonical pattern databases heuristic, thus computing the symmetric closure of the pattern collection. As a result, we obtain a heuristic that is invariant under symmetry, which makes it particularly appealing to be used with symmetry-based pruning search algorithms. We also prove that the resulting heuristic dominates the symmetric lookups heuristic over the original canonical pattern databases heuristic. Further, in order to allow for storing the larger symmetry-enhanced pattern collection without significantly increasing the memory consumption, we suggest storing the symmetric pattern databases implicitly, generalizing an approach previously suggested for domain-dependent heuristic search to domain-independent classical planning. Our empirical evaluation shows that it is beneficial to enrich the canonical pattern database heuristic with symmetric patterns, both improving heuristic informativeness and reducing memory consumption of storing the pattern databases if using the implicit representation.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Reasoning about Plans and Heuristics for Planning and Combinatorial Search" (RAPAHPACS).

References

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

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. Technical Report IS/IE-2015-03, Technion, Haifa.

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

Felner, A.; Zahavi, U.; Schaeffer, J.; and Holte, R. C. 2005. Dual lookups in pattern databases. In Kaelbling, L. P.,

and Saffiotti, A., eds., *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 103–108. Professional Book Center.

Felner, A.; Zahavi, U.; Holte, R.; Schaeffer, J.; Sturtevant, N.; and Zhang, Z. 2011. Inconsistent heuristics in theory and practice. *Artificial Intelligence* 175:1570–1603.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Helmert, M., and Röger, G. 2010. Relative-order abstractions for the pancake problem. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, 745–750. IOS Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM* 61(3):16:1–63.

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

Junttila, T., and Kaski, P. 2007. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX 2007)*, 135–149. SIAM.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In Burgard, W., and Roth, D., eds., *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, 1004–1009. AAAI Press.

Scherrer, S.; Pommerening, F.; and Wehrle, M. 2015. Improved pattern selection for PDB heuristics in classical planning (extended abstract). In Lelis, L., and Stern, R., eds., *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*, 216–217. AAAI Press.

Seipp, J., and Helmert, M. 2014. Diverse and additive Cartesian abstraction heuristics. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 289–297. AAAI Press.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and symmetries in classical planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3371–3377. AAAI Press.

Sievers, S.; Wehrle, M.; Helmert, M.; and Katz, M. 2015a. An empirical case study on symmetry handling in cost-optimal planning as heuristic search. In Hölldobler, S.; Krötzsch, M.; Peñaloza-Nyssen, R.; and Rudolph, S., eds., *Proceedings of the 38th Annual German Conference on Artificial Intelligence (KI 2015)*, volume 9324 of *Lecture Notes in Artificial Intelligence*, 151–165. Springer-Verlag.

Sievers, S.; Wehrle, M.; Helmert, M.; Shleyfman, A.; and Katz, M. 2015b. Factored symmetries for merge-and-shrink abstractions. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3378–3385. AAAI Press.

Sievers, S.; Ortlieb, M.; and Helmert, M. 2012. Efficient implementation of pattern database heuristics for classical planning. In Borrajo, D.; Felner, A.; Korf, R.; Likhachev, M.; Linares López, C.; Ruml, W.; and Sturtevant, N., eds., *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS 2012)*, 105–111. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2358–2366. AAAI Press.

Winterer, D.; Wehrle, M.; and Katz, M. 2016. Structural symmetries for fully observable nondeterministic planning. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3293–3299. AAAI Press.