An Empirical Case Study on Symmetry Handling in Cost-Optimal Planning as Heuristic Search

Silvan Sievers[1], Martin Wehrle[1], Malte Helmert[1], and Michael Katz[2]

[1] University of Basel, Switzerland
{silvan.sievers,martin.wehrle,malte.helmert}@unibas.ch
[2] IBM Haifa Research Lab, Israel
katzm@il.ibm.com

Abstract. Symmetries provide the basis for well-established approaches to tackle the state explosion problem in state space search and in AI planning. However, although by now there are various symmetry-based techniques available, these techniques have not yet been empirically evaluated and compared to each other in a common setting. In particular, it is unclear which of them should be preferably applied, and whether there are techniques with stronger performance than others. In this paper, we shed light on this issue by providing an empirical case study. We combine and evaluate several symmetry-based techniques for cost-optimal planning as heuristic search. For our evaluation, we use state-of-the-art abstraction heuristics on a large set of benchmarks from the international planning competitions.

1 Introduction

Common tasks in heuristic search and classical planning face the state explosion problem, meaning that the task's state space grows exponentially in the size of a compact description. As a consequence, the ability to effectively tackle the state explosion problem is crucial in order to scale to large problem sizes. A well-established approach for this purpose is based on the detection and exploitation of *problem symmetries*. Originating in the area of computer aided verification [13], symmetries have also been successfully applied in the heuristic search and planning communities [8, 9, 18, 7, 24, 17, 3, 4, 23, 20]. Search techniques based on symmetries traditionally take into account that "symmetrical" states can be treated in an analogous way as the "original" state, thereby attempting to reduce the size of the task's reachable search space. For example, for a robot that has to carry a blue and a red ball to a destination location, it does not matter in which hand it actually carries the blue and the red ball, rendering the corresponding states symmetrical.

Symmetries have been studied in several variations. *Symmetrical lookups*, introduced in the context of pattern database heuristics for the sliding tile puzzle [2], maximize heuristic values over symmetrical states. Similarly, *dual lookups* can be considered as an instantiation of symmetry exploitation for permutation problems. In a nutshell, dual lookups compute two heuristic values per state, one for the actual state and one for the "dual" state which is known to have the same goal distance. Hence, maximizing the estimations over these states preserves admissibility [7, 24]. For classical planning as

heuristic search, symmetries have been applied to prune symmetrical states explicitly [17, 3]. In addition, Sievers et al. [23] recently studied symmetries on a factored level for computing abstraction heuristics based on the merge-and-shrink framework [12]. Apparently, each of these techniques has shown to be useful in a particular context, but from a more global point of view, it is unclear which technique should be applied in which setting, if there are techniques that perform stronger than others, and if they can be combined to increase performance even further.

In this paper, we provide an empirical evaluation of these symmetry techniques. As the planning community offers a large and diverse benchmark set from the international planning competitions, we perform our study in the context of domain-independent planning. Our evaluation includes symmetries for cutting the search space as well as for computing merge-and-shrink heuristics. Furthermore, we adapt the concept of symmetrical and dual lookups to planning. While Shleyfman et al. [20] have shown that several planning heuristics are invariant under symmetries, this is presumably not the case for abstraction heuristics like merge-and-shrink, which are subject to our study.

2 Background

A SAS$^+$ planning task [1], augmented with operator costs, is defined as a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$ consisting of a finite set $\mathcal{V}$ of finite-domain state variables, a finite set of operators $\mathcal{O}$, an initial state s_0, a goal $s_\star$, and an operator cost function $cost$. States are defined by mappings from the variables in $\mathcal{V}$ to corresponding values in their domains. The goal description is specified as a conjunction of variable/value pairs (also called *facts*). An operator consists of a precondition and an effect which are both represented as conjunctions of facts. An operator o is applicable in a state s if o's precondition complies with s, and applying o in s yields the successor state $s(o)$ by setting o's effect variables in s accordingly. A plan is a sequence of operators that is sequentially applicable in s_0 and leads to a state that complies with the goal $s_\star$. The cost of a plan π is the sum of the costs of operators in the plan. A plan π is called optimal if its cost is minimal among all plans. A planning task Π induces a state transition graph $\mathcal{T}_\Pi$, where $\mathcal{T}_\Pi$'s vertices are Π's states, and there is an edge between states s and s' if there is an operator o that is applicable in s and $s' := s(o)$.

We will provide a short introduction to techniques that exploit symmetries for different purposes. We refer to the literature for details and more formal descriptions.

2.1 Structural Symmetries & Orbit Space Search

We base on the notion of *structural symmetries* (called symmetries for short in the following) which have recently been introduced by Shleyfman et al. [20]. Such symmetries map facts to facts and operators to operators in a way that forces the induced mapping on the state transition graph $\mathcal{T}_\Pi$ to be an automorphism of $\mathcal{T}_\Pi$ that maps goal nodes to goal nodes. Symmetries induce equivalence relations on $\mathcal{T}_\Pi$'s nodes, i.e. on the set of Π's states. Two states s and s' are in the same equivalence class if there is a symmetry σ such that $\sigma(s) = s'$.

In general, finding the coarsest equivalence relation is NP-hard [15]. Hence, in practice, an equivalence between two states s and s' is established via a *procedural* mapping $\mathcal{C} : \mathcal{S} \rightarrow \mathcal{S}$ from states to states in their equivalence class which induces an over-approximation of the coarsest equivalence relation. Two states s and s' are said to be equivalent if they are mapped to the same state by $\mathcal{C}$. The equivalence classes induced by $\mathcal{C}$ are called *orbits*. Pruning algorithms based on symmetry elimination only consider the orbits instead of all states. In the following, we consider symmetry elimination based on *orbit space search* [5]. Orbit space search performs the search directly on the space induced by the orbits of all states: For all encountered states s, a symmetrical representative of the orbit of s is computed and used for the further search.

2.2 Factored Symmetries & Merge-And-Shrink

Symmetries have also been studied on a factored level [23] for computing merge-and-shrink (M&S) heuristics [12]. Merge-and-shrink heuristics represent a popular class of abstraction heuristics. Starting from the "atomic" abstractions that represent the projection on single variables, the merge-and-shrink computation iterativeley selects two elements from the current set of abstractions, possibly unifies abstract states in one or both abstractions so that the product of their sizes respects a given size limit (the so-called *shrink* step), and then computes the synchronized product of the two abstractions (the so-called *merge* step). The resulting synchronized product replaces the two abstractions, and the process repeats until one abstraction is left.

In this framework, for a given set of abstractions Θ during the computation of merge-and-shrink, *factored symmetries* capture *locally* symmetrical aspects of (some of) the abstractions in Θ. Such symmetries can be used for lossless shrinking and to devise merging strategies for computing merge-and-shrink abstractions by preferably merging those abstractions that are affected by common factored symmetries.

2.3 Symmetrical Lookups

The concept of symmetrical lookups has been successfully proposed in the area of search, but has not been evaluated in the planning area so far. We have adapted symmetrical lookups for planning as follows: For a given heuristic h and state s, we define the heuristic value $\bar{h}(s)$ for s as the maximum of $\{h(s), h(s^1), \ldots, h(s^m)\}$, where s^i for $i \in \{1, \ldots, m\}$ are states located in the same orbit as s (i.e. states symmetrical to state s). From a theoretical point of view, there is no further restriction on the set $S = \{s^1, \ldots, s^m\}$ of symmetrical states. At the extreme ends of the spectrum, S could be empty (i.e. no symmetrical lookups are performed), or could contain the whole set of states from the orbit of s (which is presumably expensive to compute in practice). From a practical point of view, several strategies to compute S are possible, and it remains an experimental question to find the sweet-spot of the tradeoff to increase the heuristic values as much as possible, while still being efficiently computable.

While the use of symmetrical lookups preserves the admissibility of a given heuristic h, it generally renders h to be *inconsistent*, which means that the resulting heuristic $\bar{h}$ does no longer satisfy the equality $\bar{h}(s) \leq \bar{h}(s') + cost(o)$, where s' is the successor state of s when operator o is applied. To alleviate this problem, *bidirectional pathmax*

(BPMX) has been proposed and successfully applied in the heuristic search community [6]. Informally speaking, BPMX "repairs" inconsistent jumps in the heuristic values for s and successor state $s(o)$ (and vice versa) by adapting the values accordingly. Like symmetrical lookups, BPMX has not been applied in the planning context. Because symmetrical lookups as we adapted them to planning face the same problem of rendering heuristic values inconsistent, we also adapt BPMX to planning: whenever a node is expanded, the heuristic values of its successors are recursively updated up to a given recursion depth.

3 Experimental Study

We evaluate the previously discussed symmetry techniques for classical planning using A* search or orbit space search in combination with several abstraction heuristics from the planning literature. All techniques are implemented in the Fast Downward planner [11]. We use the optimal benchmarks from the International Planning Competition (IPC) up to IPC2011 with language features supported by the investigated heuristics (44 domains with a total of 1396 tasks). All experiments are performed on computers with Intel Xeon E5-2660 CPUs running at 2.2 GHz and with a time bound of 30 minutes and a memory bound of 2 GB, as common in IPCs. We compute both structural symmetries and factored symmetries with the graph automorphism tool *Bliss* [14].

3.1 Symmetries in Common Planning Benchmarks

In a first experiment, we investigate the occurrence of symmetries in the set of considered planning benchmarks. To the best of our knowledge, despite the success of symmetry handling in planning, no previous work has *quantitatively* analyzed the occurrence of symmetries in commonly used planning benchmarks so far. In the following, we report the number of tasks of every domain in which we discovered at least one non-trivial symmetry generator, the sum and the median of such generators aggregated over all tasks of every domain, as well as the number of discovered symmetry generators of different *orders*.[3] Table 1 lists the data.

The first and general observation we make (columns 2 and 3) is that there are lots of symmetries in this standard set of planning benchmarks, even more than one might have expected. In particular, only 3 domains (Blocksworld and the two Parcprinter domains) do not expose a symmetry in any task, and in 1103 tasks symmetries do occur. Furthermore, in 38 out of 44 domains, more than half of the tasks contain symmetries, and in most of these 38 domains, almost all the tasks are symmetrical. This huge number of symmetries is remarkable as the domains stem from quite different areas, covering both academic and real-world scenarios. It shows that symmetries are a quite general concept that often occurs in practice. Obviously, the large number of symmetries particularly explains the recent success of the symmetry techniques evaluated in planning.

[3] The order of a generator σ is defined as the smallest number of function compositions with itself that yields the identity function, i.e. $order(\sigma) = n$ if n is the smallest number with $\underbrace{\sigma \circ \ldots \circ \sigma}_{n} = id$.

Table 1. Properties of IPC domains: total number of tasks (total), number of tasks with at least one symmetry (symm), and number of generators for every domain (sum and median over all tasks), histogram of generators' order.

	# tasks		# generators		# generators of order			
	total	symm	sum	median	2	3	4	5
airport	50	42	205	4	205	-	-	-
barman-11	20	20	73	4	73	-	-	-
blocks	35	0	0	0	0	-	-	-
depot	22	22	118	4	118	-	-	-
driverlog	20	20	85	3	85	-	-	-
elevators-08	30	20	38	1	38	-	-	-
elevators-11	20	13	23	1	23	-	-	-
floortile-11	20	20	48	2	48	-	-	-
freecell	80	1	1	0	1	-	-	-
grid	5	5	27	5	27	-	-	-
gripper	20	20	460	23	460	-	-	-
logistics00	28	28	116	3.5	116	-	-	-
logistics98	35	35	2757	51	2756	1	-	-
miconic	150	141	1893	13	1892	1	-	-
mprime	35	35	649	15	649	-	-	-
mystery	30	28	471	10.5	471	-	-	-
nomystery-11	20	20	54	2.5	54	-	-	-
openstacks-08	30	30	221	7	221	-	-	-
openstacks-11	20	20	149	7	149	-	-	-
openstacks	30	11	20	0	19	-	-	1
parcprinter-08	30	0	0	0	0	-	-	-
parcprinter-11	20	0	0	0	0	-	-	-
parking-11	20	20	150	7.5	150	-	-	-
pathways-noneg	30	29	210	7	210	-	-	-
pegsol-08	30	30	58	2	58	-	-	-
pegsol-11	20	20	40	2	40	-	-	-
pipesworld-notankage	50	50	470	8	470	-	-	-
pipesworld-tankage	50	50	1547	22.5	1547	-	-	-
psr-small	50	32	73	1	73	-	-	-
rovers	40	32	381	4	381	-	-	-
satellite	36	36	12115	92	12115	-	-	-
scanalyzer-08	30	26	201	6	200	1	-	-
scanalyzer-11	20	18	142	6.5	142	-	-	-
sokoban-08	30	27	114	3	113	-	1	-
sokoban-11	20	19	66	3	65	-	1	-
tidybot-11	20	7	13	0	13	-	-	-
tpp	30	29	197	6	197	-	-	-
transport-08	30	30	76	2	76	-	-	-
transport-11	20	20	50	2	50	-	-	-
trucks	30	29	96	3	96	-	-	-
visitall-11	20	11	16	1	16	-	-	-
woodworking-08	30	22	244	5	244	-	-	-
woodworking-11	20	16	150	5	150	-	-	-
zenotravel	20	19	199	6	199	-	-	-
Total	1396	1103	24016	4	24010	3	2	1

In addition, it suggests that further improvements based on symmetry-exploiting techniques are achievable—we will come back to this point below.

Furthermore, we observe that there is a remarkable difference in the number of generators found in the different domains, both with respect to the sum and with respect to the median (columns 4 and 5), which shows that the domains are structured quite differently in this respect. Interestingly, considering the generators' order for every task, we observe that the vast majority of generators has the simplest possible order of two.

In rare cases, however, there also exist more complex generators of higher order. We should note that even having all generators being of order 2 does not ensure that all elements of the group will be of that order. An example can be seen in Gripper domain, where generators found by Bliss represent either (a) symmetries between two grippers, or (b) symmetries between ball i and ball $i + 1$, for all $1 \leq i < n$. All generators are of order 2, but there are elements of all orders up to n that can be composed out of those generators. Note also that other tools for finding automorphisms of coloured graphs might have found a different set of generators. Lastly, note that group or generator orders are only some of the features describing group structure and knowing a group structure could result in better exploitation of the found symmetries. Apparently, as we will see in the next sections, the existing methods already yield powerful symmetry elimination techniques for planning. It remains an open question *why* so many of the generators discovered by Bliss have this particular order. We suspect that the generators' order depends on the *representation* of the considered planning task. Currently, we have used the SAS$^+$ representation generated by the Fast Downward planner. It will be interesting to investigate if there are more suitable SAS$^+$ representations that are more amenable to finding generators of higher order, and if the available symmetry-techniques can profit from them.

3.2 Setup of the Study

Our evaluation focuses on abstraction heuristics because they represent a popular class of planning heuristics used for cost-optimal planning that are presumably not invariant under symmetries. In more detail, we evaluate heuristics based on merge-and-shrink [12], iPDB [10] in the implementation by Sievers, Ortlieb and Helmert [21], and CEGAR [19].

To use symmetrical lookups and orbit space search in combination with merge-and-shrink heuristics, we need to make sure that the heuristics yield admissible values for all symmetrical states. While this might seem obvious at first glance, admissibility is no longer guaranteed for regular merge-and-shrink heuristics within a straight-forward combination: in Fast Downward, abstract states in intermediate abstractions are pruned if they are unreachable from the initial state of the task. However, as the applied symmetries do not stabilize the initial state, admissibility can be violated because a non dead-end state could have a symmetrical state which corresponds to such a pruned abstract state. To address this issue, we simply disable this pruning within the computation of merge-and-shrink in all configurations that combine merge-and-shrink with orbit search. (The alternative of only using symmetries that additionally stabilize the initial state yields fewer symmetries and performs worse.) For the combinations of merge-and-shrink with symmetrical lookups and without orbit space search, we address this problem by ignoring symmetrical states with values of infinity if the original state is not evaluated to infinity. (The alternatives of only using symmetries that also stabilize the initial state again performs worse, and disabling the pruning of unreachable states within merge-and-shank is slightly worse in this setting.)

In the following, we focus on merge-and-shrink heuristics because all symmetry techniques (including factored symmetries) are applicable. Results for iPDB and CEGAR are discussed at the end of the section.

3.3 Symmetrical Lookups and BPMX

We investigate the following questions: First, how much can symmetrical lookups reduce the number of expansions and increase the coverage (i.e. the number of solved problems)? Second, what is the influence of the number of considered symmetrical states? Third, can these techniques improve the total runtime? Fourth, how important is BPMX in this setting? We report results for the best available merge-and-shrink configuration in Fast Downward, which uses the merging strategy DFP [22] and the shrinking strategy based on bisimulation [16] with size limit 50000.

We address the first three questions (i.e. no use of BPMX yet) in a first experiment, reported in Table 2. We compare the baseline, i.e. A^* with merge-and-shrink (base), to merge-and-shrink with the inclusion of symmetrical lookups for 1 symmetrical state found by a short random walk in the orbit (slone), and for 5, 10 or all symmetrical states found by a breadth first search in the orbit (slsub5, slsub10, slall). We observe the following trends. Symmetrical lookups generally help in increasing the coverage and reducing the number of expansions,[4] both with respect to the sum over all commonly solved tasks as well as with respect to the median over all tasks solved by at least one configuration (where unsolved tasks are counted as infinity). While the extreme ends of the spectrum (considering one vs all symmetrical states) does not hit the sweet spot of the tradeoff to be as informative and efficient as possible, considering *some* symmetrical states can considerably decrease the number of expansions while still being efficiently computable. In the following, when referring to symmetrical lookups, we always mean the best configuration where the h value is maximized over 10 additional symmetrical states (and call it "sl" from here on).

Table 2. M&S (base) vs. M&S with symmetrical lookups: one symmetrical state computed via a short random walk (slone), a subset of symmetrical states of size 5/10 (slsub5/10), all symmetrical states (slall).

	base	**slone**	**slsub5**	**slsub10**	**slall**
Coverage	652	656	**658**	**658**	**658**
Expansions sum	607602428	501671723	493848579	**471769190**	493848579
Expansions median	1263	1059	**811**	**811**	**811**

Next, we compare the baseline against the best configuration with symmetrical lookups in more details on a per-task base. Figure 1 shows results as scatterplots of expansions and total time for this comparison. The plot for expansions (left) shows that using symmetrical lookups improves the heuristic quality in quite a lot of problems (reduced number of expansions). Considering the runtime (right plot), we see that computing symmetrical states as expected incurs a computational overhead that results in a general increase of runtime. Still, the coverage increases as previously shown in Ta-

[4] Note that we generally report the number of expansions excluding the last f layer to avoid the (arbitrary) tie-breaking effects in the last f layer.

ble 2, and hence the heuristic quality improvement in this comparison outweighs the increase in runtime.

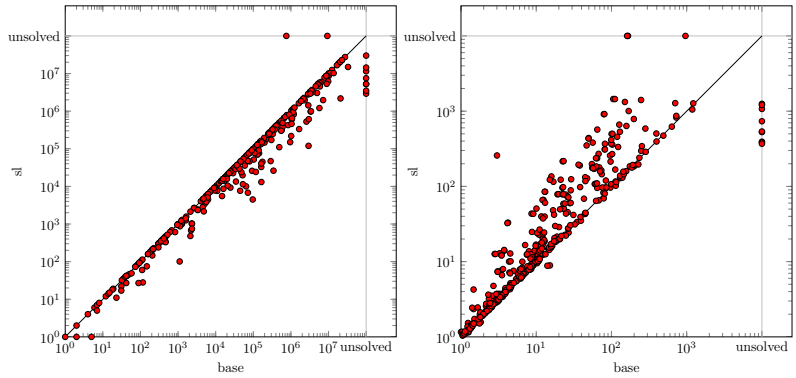

Fig. 1. M&S (base) vs. M&S with symmetrical lookups (sl): expansions (left) and total time (right)

Finally, we investigate to which extent BPMX can help in combination with symmetrical lookups, which generally render heuristics to be inconsistent. We evaluate three variants: the most simple variant that only updates h values of successor states (bp1), a variant that updates the h values recursively up to a depth of 2 (i.e. for successor states of successor states, including the parents if there exist invertible operators, which corresponds to BPMX in the sense of Felner et al. [6]) (bp2), and a variant that performs up to 10 recursive updates of h values of previously visited states (bp10). Table 3 shows the results.

Table 3. M&S (base) vs. M&S with symmetrical lookups (sl) with and without BPMX of varying depth X (-bpX).

	base	**sl**	**sl-bp1**	**sl-bp2**	**sl-bp10**
Coverage	652	**658**	**658**	**658**	**658**
Expansions sum	607602428	**471769190**	471769292	471769236	471769236
Expansions median	1260	**751**	**751**	**751**	**751**

We observe that, in contrast to results reported in the heuristic search community, BPMX does not help when applied for planning problems. The reduction in expansions compared to using symmetrical lookups without BPMX is very small, in a range not visible when comparing the median over tasks, and there is no coverage gain. As one would expect based on these numbers, a comparison of symmetrical lookups with BPMX (with recursive updates up to depth 10) against the baseline yields scatterplots

that look the same as the plots in Fig. 1. Also a direct comparison of using symmetrical lookups with and without BPMX shows that the number of expansions remains the same for nearly all tasks, and the runtime slightly increases for some tasks. A more detailed analysis reveals that heuristic value corrections due to BPMX only occur in 13 domains, in approximately 2% of all tasks for which the merge-and-shrink abstraction was successfully computed, which shows that in most cases, merge-and-shrink with symmetrical lookups still remains a consistent heuristic.

We conclude that symmetrical lookups with not too much overhead, i.e. for a limited number of additional symmetrical states evaluations in every state, can yield performance improvements for planning. Using BPMX may improve the heuristic quality in very few cases, but the computational overhead never pays off (but it also does not hurt in terms of coverage in the tested configurations). In the following, we hence stick to using symmetrical lookups without BPMX.

3.4 Results for Merge-and-Shrink Heuristic

In our final experiments for merge-and-shrink, we compare all techniques. We again use A^* with merge-and-shrink as baseline (base), and combine merge-and-shrink with symmetrical lookups (sl) as before, with orbit space search (oss), and with factored symmetries in the configuration "symm" reported by Sievers et al. [23] (fs). Table 4 shows domain-wise coverage and the number of expansions as the sum over commonly solved tasks and as the median over all tasks solved by at least one configuration.

Comparing the individual techniques to the baseline (columns 2–5), we observe that all symmetry techniques help, both in terms of coverage and expansions. For the domains with no symmetries (Blocksworld and both Parcprinter domains), there is a slight reduction in coverage for some configurations due to the overhead of searching for symmetries without finding any. For all other domains, orbit space search and symmetrical lookups only reduce coverage in a very few cases, whereas factored symmetries perform worse in a few more cases. Generally, orbit space search yields by far the strongest performance improvement (both in terms of coverage and expansions), compared to symmetrical lookups and factored symmetries that increase the coverage rather modestly, but also reduce expansions.

When combining the individual techniques (columns 6–9), we again observe that orbit space search increases the coverage for all configurations, i.e. it is always beneficial to include it with either of the other two techniques, compared to only using one of the other techniques (with one exception in the domain nomystery). The combination of factored symmetries with orbit space search is particularly beneficial, achieving the overall highest coverage, improving over both using only factored symmetries or only using orbit space search. The opposite holds for the combination of symmetrical lookups with orbit space search: coverage decreases compared to only using orbit space search. We generally observe that adding symmetrical lookups to a configuration decreases the number of expansions as expected, but does not increase performance in terms of coverage, due to the computational overhead.

Table 4. Domain-wise coverage and aggregated expansions (sum and median) for M&S with all symmetry combinations. Abbreviations: base: A*, oss: orbit space search, sl: symmetrical lookups, fs: factored symmetries; X-Y: combination of X and Y; all: combination of oss, sl and fs.

	base	**oss**	**sl**	**fs**	**oss-sl**	**oss-fs**	**sl-fs**	**all**
airport (50)	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
barman-11 (20)	4	**8**	4	4	7	**8**	4	7
blocks (35)	**27**	**27**	**27**	26	**27**	**27**	26	26
depot (22)	6	8	7	7	8	**9**	7	**9**
driverlog (20)	12	**13**	12	12	**13**	**13**	12	**13**
elevators-08 (30)	16	**19**	17	16	**19**	18	17	18
elevators-11 (20)	13	**16**	14	13	**16**	15	14	15
floortile-11 (20)	**5**	**5**	**5**	2	**5**	3	2	3
freecell (80)	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
grid (5)	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
gripper (20)	19	**20**	19	18	**20**	**20**	18	**20**
logistics00 (28)	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
logistics98 (35)	**5**	**5**	**5**	4	**5**	**5**	4	**5**
miconic (150)	72	76	73	77	75	**78**	76	**78**
mprime (35)	**23**	22	**23**	**23**	22	**23**	**23**	**23**
mystery (30)	16	16	16	16	15	**17**	16	16
nomystery-11 (20)	18	18	**20**	16	**20**	18	16	18
openstacks-08 (30)	20	**24**	19	20	23	**24**	19	23
openstacks-11 (20)	15	**19**	14	15	18	**19**	14	18
openstacks (30)	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
parcprinter-08 (30)	**14**	13	**14**	**14**	13	13	**14**	13
parcprinter-11 (20)	**10**	9	**10**	**10**	9	9	**10**	9
parking-11 (20)	2	2	2	**7**	2	**7**	**7**	**7**
pathways-noneg (30)	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
pegsol-08 (30)	**29**	**29**	**29**	27	**29**	28	27	28
pegsol-11 (20)	**19**	**19**	**19**	17	**19**	18	17	18
pipesworld-nt (50)	16	**18**	15	16	16	**18**	16	16
pipesworld-t (50)	14	**17**	14	15	**17**	**17**	15	**17**
psr-small (50)	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**
rovers (40)	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
satellite (36)	6	**7**	6	6	6	**7**	6	**7**
scanalyzer-08 (30)	13	**18**	13	12	**18**	17	12	17
scanalyzer-11 (20)	10	**14**	10	9	**14**	13	9	13
sokoban-08 (30)	26	29	27	**30**	29	**30**	**30**	**30**
sokoban-11 (20)	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
tidybot-11 (20)	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
tpp (30)	6	7	7	6	**8**	7	6	7
transport-08 (30)	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
transport-11 (20)	6	**7**	**7**	6	**7**	**7**	**7**	**7**
trucks (30)	7	8	7	8	8	**9**	8	**9**
visitall-11 (20)	9	9	9	**10**	9	**10**	**10**	**10**
woodworking-08 (30)	**13**	**13**	**13**	**13**	**13**	12	**13**	12
woodworking-11 (20)	**8**	**8**	**8**	**8**	**8**	7	**8**	7
zenotravel (20)	**12**	**12**	**12**	10	**12**	11	11	**12**
Coverage sum (1396)	652	696	658	654	691	**698**	655	692
Expansions sum	5.16e+8	2.68e+8	4.01e+8	3.65e+8	2.54e+8	2.39e+8	3.44e+8	**2.32e+8**
Expansions median	5077	4292	4481	7432	2814	5499	6216	4593

3.5 Results for iPDB and CEGAR Heuristics

We investigate orbit space search and symmetrical lookups with iPDB and with the CEGAR heuristic in its best configuration using the landmarks and goals decomposition [19]. Again, we report results for computing 10 symmetrical states when using symmet-

rical lookups, and leave out BPMX as its benefit is negligible also in this context. Table 5 shows a domain-wise overview of coverage and summarized expansions (summed over commonly solved tasks and the median over all tasks solved by at least one configuration of the heuristic) for A* with the corresponding heuristic (base), and for the corresponding heuristic combined with orbit space search (oss), symmetrical lookups (sl), and the combination thereof (oss-sl).

Table 5. CEGAR and iPDB with A* (base), with orbit space search (oss), with A* and symmetrical lookups (sl), and with a combination of oss and sl (oss-sl).

	CEGAR				iPDB			
	base	**oss**	**sl**	**oss-sl**	**base**	**oss**	**sl**	**oss-sl**
airport (50)	**32**	24	30	28	**23**	**23**	**23**	**23**
barman-11 (20)	4	**8**	4	6	4	**8**	4	7
blocks (35)	**18**	**18**	**18**	**18**	**28**	**28**	**28**	**28**
depot (22)	6	**7**	6	**7**	8	10	7	**11**
driverlog (20)	10	11	10	**12**	**13**	**13**	**13**	**13**
elevators-08 (30)	18	**19**	**19**	**19**	20	**21**	20	**21**
elevators-11 (20)	15	**16**	**16**	**16**	16	**17**	16	**17**
floortile-11 (20)	**2**	**2**	**2**	**2**	2	**3**	2	**3**
freecell (80)	52	**53**	**53**	52	**20**	**20**	**20**	
grid (5)	**2**	**2**	**2**	**2**	**3**	**3**	**3**	**3**
gripper (20)	7	**20**	7	**20**	7	**20**	7	**20**
logistics00 (28)	**20**	15	**20**	16	**21**	20	**21**	20
logistics98 (35)	**9**	5	8	6	**5**	**5**	**5**	**5**
miconic (150)	71	**75**	66	73	55	**60**	55	58
mprime (35)	26	24	**27**	23	**23**	**23**	**23**	**23**
mystery (30)	**17**	16	**17**	15	16	**17**	16	**17**
nomystery-11 (20)	**14**	13	**14**	**14**	18	**20**	19	**20**
openstacks-08 (30)	20	**24**	19	23	20	**24**	19	23
openstacks-11 (20)	15	**19**	14	18	15	**19**	14	18
openstacks (30)	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
parcprinter-08 (30)	**22**	**22**	**22**	**22**	**13**	**13**	**13**	**13**
parcprinter-11 (20)	**17**	**17**	**17**	**17**	**9**	**9**	**9**	**9**
parking-11 (20)	**0**	**0**	**0**	**0**	**7**	**7**	**7**	**7**
pathways-noneg (30)	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
pegsol-08 (30)	**28**	**28**	**28**	**28**	28	**29**	27	28
pegsol-11 (20)	**18**	**18**	**18**	**18**	19	**20**	18	19
pipesworld-notankage (50)	17	**20**	17	18	21	**24**	20	22
pipesworld-tankage (50)	13	**18**	13	17	16	**20**	16	19
psr-small (50)	49	**50**	49	**50**	49	**50**	49	**50**
rovers (40)	**7**	**7**	**7**	**7**	**8**	**8**	**8**	**8**
satellite (36)	6	**7**	6	6	**6**	**6**	**6**	**6**
scanalyzer-08 (30)	12	**16**	12	15	13	**18**	13	**18**
scanalyzer-11 (20)	9	**12**	9	11	10	**14**	10	**14**
sokoban-08 (30)	22	**27**	20	26	29	**30**	29	**30**
sokoban-11 (20)	19	**20**	17	**20**	**20**	**20**	**20**	**20**
tidybot-11 (20)	**14**	10	**14**	**14**	**14**	**14**	**14**	**14**
tpp (30)	7	**8**	7	**8**	6	**7**	6	**7**
transport-08 (30)	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
transport-11 (20)	**6**	**6**	**6**	**6**	6	**7**	6	**7**
trucks (30)	**12**	**12**	**12**	**12**	8	**10**	9	**10**
visitall-11 (20)	**9**	**9**	**9**	**9**	**16**	**16**	**16**	**16**
woodworking-08 (30)	**12**	**12**	**12**	**12**	**9**	**9**	**9**	**9**
woodworking-11 (20)	**7**	**7**	**7**	**7**	**4**	**4**	**4**	**4**
zenotravel (20)	12	12	**13**	**13**	11	**12**	11	11
Sum (1396)	698	**731**	689	728	661	**723**	657	713
Expansions sum	50.8e+8	29.2e+8	44.5e+8	**19.1e+8**	33.2e+8	15.0e+8	31.4e+8	**13.3e+8**
Expansions median	5118	7285	5799	**2906**	6931	2440	6339	**1952**

For CEGAR, we observe that orbit space search is again, as for merge-and-shrink, the most improving symmetry-based technique. However, the are also several domains in which coverage decreases and the median of expansions is even higher than with the baseline. Presumably, due to the way they are constructed, CEGAR abstractions are especially well-informed along a path from the initial state to the goal, but not necessarily on a symmetrical path (because our symmetries do not stabilize the initial stated). Furthermore, due to the CEGAR computation starting the refinement near goal states, CEGAR abstractions yield well-informed heuristics close to the goal, which often results in fewer expansions on the last f layer than with other heuristics. Indeed, computing the median of expansions including the last f layer, the number for CEGAR with orbit space search is smaller than for the baseline. The second observation for CEGAR is that adding symmetrical lookups reduces the summed number of expansions as for merge-and-shrink, but also decreases coverage due to the computational overhead both compared to the baseline and orbit space search.

Considering iPDB, we observe a behavior more similar to merge-and-shrink than CEGAR: orbit space search is the best performer in terms of coverage in 42 out of 44 domains, and expansions are greatly decreased. However, including symmetrical lookups is again only beneficial in terms of expansions, but not for coverage.

4 Discussion

Our case study shows that symmetries frequently occur in various planning tasks, and confirms that currently available symmetry-techniques can significantly help for planning with state-of-the-art abstraction heuristics. Most notably, this is the case for orbit space search, where the number of solved problems usually increases considerably for all of the considered heuristics. For the other techniques, i.e. symmetrical lookups and factored symmetries, the improvement in coverage is often rather modest (and in some cases, coverage can even decrease). We furthermore observe that BPMX does not perform equally strong for planning as for search.

Despite the improvement obtained with existing symmetry-techniques, one can argue that there is even room for further improvement because there are several domains (e.g., Airport or Rovers) where symmetries do occur, but the number of solved tasks could not be increased nevertheless. Even in the domain with the highest number of discovered symmetries (Satellite), the coverage increase is rather modest (one more solved task with merge-and-shrink and the CEGAR heuristic, no improvement for iPDB). Exploiting these symmetries more accurately can potentially yield even stronger symmetry-techniques. Additionally, it will be interesting to investigate the impact of the SAS^+ representation on the occurrence of symmetries.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project “Automated Reformulation and Pruning in Factored State Spaces (ARAP)”.

References

1. Bäckström, C., Nebel, B.: Complexity results for SAS^+ planning. Computational Intelligence 11(4), 625–655 (1995)
2. Culberson, J.C., Schaeffer, J.: Pattern databases. Computational Intelligence 14(3), 318–334 (1998)
3. Domshlak, C., Katz, M., Shleyfman, A.: Enhanced symmetry breaking in cost-optimal planning as forward search. In: McCluskey, L., Williams, B., Silva, J.R., Bonet, B. (eds.) Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012). AAAI Press (2012)
4. Domshlak, C., Katz, M., Shleyfman, A.: Symmetry breaking: Satisficing planning and landmark heuristics. In: Borrajo, D., Kambhampati, S., Oddi, A., Fratini, S. (eds.) Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013). pp. 298–302. AAAI Press (2013)
5. Domshlak, C., Katz, M., Shleyfman, A.: Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. Tech. Rep. IS/IE-2015-03, Technion, Haifa (2015)
6. Felner, A., Zahavi, U., Holte, R., Schaeffer, J., Sturtevant, N., Zhang, Z.: Inconsistent heuristics in theory and practice. Artificial Intelligence 175, 1570–1603 (2011)
7. Felner, A., Zahavi, U., Schaeffer, J., Holte, R.C.: Dual lookups in pattern databases. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005). pp. 103–108. Professional Book Center (2005)
8. Fox, M., Long, D.: The detection and exploitation of symmetry in planning problems. In: Dean, T. (ed.) Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999). pp. 956–961. Morgan Kaufmann (1999)
9. Fox, M., Long, D.: Extending the exploitation of symmetries in planning. In: Ghallab, M., Hertzberg, J., Traverso, P. (eds.) Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002). pp. 83–91. AAAI Press (2002)
10. Haslum, P., Botea, A., Helmert, M., Bonet, B., Koenig, S.: Domain-independent construction of pattern database heuristics for cost-optimal planning. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007). pp. 1007–1012. AAAI Press (2007)
11. Helmert, M.: The Fast Downward planning system. Journal of Artificial Intelligence Research 26, 191–246 (2006)
12. Helmert, M., Haslum, P., Hoffmann, J., Nissim, R.: Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. Journal of the ACM 61(3), 16:1–63 (2014)
13. Ip, C.N., Dill, D.L.: Better verification through symmetry. Formal Methods in System Design 9(1–2), 41–75 (1996)
14. Junttila, T., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In: Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX 2007). pp. 135–149. SIAM (2007)
15. Luks, E.M.: Permutation groups and polynomial-time computation. In: Groups and Computation, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 11, pp. 139–175 (1993)
16. Nissim, R., Hoffmann, J., Helmert, M.: Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In: Walsh, T. (ed.) Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011). pp. 1983–1990 (2011)

17. Pochter, N., Zohar, A., Rosenschein, J.S.: Exploiting problem symmetries in state-based planners. In: Burgard, W., Roth, D. (eds.) Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011). pp. 1004–1009. AAAI Press (2011)
18. Rintanen, J.: Symmetry reduction for SAT representations of transition systems. In: Giunchiglia, E., Muscettola, N., Nau, D. (eds.) Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003). pp. 32–40. AAAI Press (2003)
19. Seipp, J., Helmert, M.: Diverse and additive Cartesian abstraction heuristics. In: Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014). pp. 289–297. AAAI Press (2014)
20. Shleyfman, A., Katz, M., Helmert, M., Sievers, S., Wehrle, M.: Heuristics and symmetries in classical planning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015). pp. 3371–3377. AAAI Press (2015)
21. Sievers, S., Ortlieb, M., Helmert, M.: Efficient implementation of pattern database heuristics for classical planning. In: Borrajo, D., Felner, A., Korf, R., Likhachev, M., Linares López, C., Ruml, W., Sturtevant, N. (eds.) Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS 2012). pp. 105–111. AAAI Press (2012)
22. Sievers, S., Wehrle, M., Helmert, M.: Generalized label reduction for merge-and-shrink heuristics. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014). pp. 2358–2366. AAAI Press (2014)
23. Sievers, S., Wehrle, M., Helmert, M., Shleyfman, A., Katz, M.: Factored symmetries for merge-and-shrink abstractions. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015). pp. 3378–3385. AAAI Press (2015)
24. Zahavi, U., Felner, A., Holte, R.C., Schaeffer, J.: Duality in permutation state spaces and the dual search algorithm. Artificial Intelligence 172(4–5), 514–540 (2008)