# Abstraction-based Action Ordering in Planning

**Maria Fox** and **Derek Long** and **Julie Porteous**
Department of Computer and Information Sciences
University of Strathclyde, Glasgow, UK

## Abstract

Many planning problems contain collections of symmetric objects, actions and structures which render them difficult to solve efficiently. It has been shown that the detection and exploitation of symmetric structure in planning problems can dramatically reduce the size of the search space and the time taken to find a solution. We present the idea of using an abstraction of the problem domain to reveal symmetric structure and guide the navigation of the search space. We show that this is effective even in domains in which there is little accessible symmetric structure available for pruning. Proactive exploitation represents a flexible and powerful alternative to the symmetry-breaking strategies exploited in earlier work in planning and CSPs. The notion of *almost symmetry* is defined and results are presented showing that proactive exploitation of almost symmetry can improve the performance of a heuristic forward search planner.

## 1 Introduction

Symmetries frequently occur in search problems such as planning problems [Joslin and Roy, 1997; Fox and Long, 1999; Rintanen, 2003], CSPs [Roy and Pachet, 1998; Gent and Smith, 2000; Roney-Dougal *et al.*, 2004] and model checking [Ip and Dill, 1996; Audemard and Benhamou, 2002]. Many techniques have been developed for symmetry-breaking to improve search performance, with emphasis on using symmetries to prune search spaces rather than to suggest branches to pursue. In this paper we propose the use of symmetries to direct the forward search of an FF-style planner in a positive way. We show that the proactive use of symmetries can lead to significant performance improvements across a variety of planning domains.

The automatic identification of all of the symmetries of a problem is NP-hard. Because of the complexity of the identification problem most researchers working in symmetry have broken symmetries by hand by expressing specific symmetry-breaking constraints in the modelling of the problem [Gent and Smith, 2000; Ip and Dill, 1996; Roney-Dougal *et al.*, 2004]. Planning domain models are structured in a way that often makes some of the underlying symmetric structure accessible and a subset of the available symmetries can be efficiently identified using automatic techniques [Fox and Long, 1999; 2002]. However, it has been observed that the most directly accessible symmetries are often not the ones that would be most useful to exploit.

It happens that structures within a planning problem are often *almost*, but not quite, symmetric, and that treating them as symmetric would increase the efficiency of the search. *Almost symmetries* are revealed by the application of an appropriate abstraction to the domain. As we will see, the term *almost* refers to the fact that the associated symmetry belongs to the abstracted domain and might be unsound with respect to the original domain. Such symmetries can only be exploited in a positive way (to suggest how best to develop the plan) because using them for pruning would compromise completeness. In this paper we consider how the application of a certain simple abstraction, which we call the *property-based abstraction*, can reveal a form of almost symmetry which we then show can be effectively exploited to solve the original problem. We begin with a simple motivating example, then we present the definitions and examples that support the following discussion. We then describe the extension of FF that enables the proactive exploitation of symmetric problem structure and discuss a collection of results obtained from STRIPS domains used in the 2002 planning competition.

## 2 Identifying Almost Symmetry

Consider the problem of transporting a number of crates from one location to another (as in the Depots domain [Fox and Long, 2003]). The crates are stacked in several different piles in the initial state and must be moved into new configurations. Any solution plan will involve unstacking the crates, loading them onto transport and then delivering and unloading them. At an abstract level at which the precise locations of the crates are ignored, the crates can be treated as symmetric because the same sub-plans are required to get all of the crates into their goal configurations.

The fact that the cargoes start stacked in different piles means that they cannot be automatically identified as functionally equivalent [Fox and Long, 1999]). Furthermore, since their initial configurations are different they cannot be identified as symmetric in the sense exploited in [Joslin and Roy, 1997]). Nevertheless, there is a high degree of symmetry

in the structure of the problem even though it is not immediately accessible to automatic analysis. The crates are *almost symmetric* because they can be made symmetric by abstracting out the specific domain objects to which they are related (eg: crate1 is on *something*, but it doesn't matter what), without losing sight of the properties (being *on* something as opposed to having something else *on top*) that distinguish them. When a collection of objects (in this case crates) are almost symmetrical we reason that we may well need to perform the same operations on these objects during the course of any eventual solution plan.

We say that two actions are almost symmetric if they are two different instances of the same operator in which objects in the corresponding argument positions are symmetric in the abstracted domain. The stategy we describe below favours the selection of actions that are almost symmetric to actions that have already been applied in the plan.

# 3 Symmetries in Planning Domains

In this section we develop a formal framework for defining symmetries in planning problems. We begin with familiar definitions of a planning domain, a planning problem and a plan. Symmetries have been studied for decades by mathematicians as *groups*. We therefore adopt a group-theoretic characterisation, defining each symmetry as a permutation that preserves the equivalence of the domain configurations to which it is applied. The elements of our groups act on the objects in a domain.

**Definition 3.1** *A* planning problem, *P, is a tuple (O,I,G,C), where O is a set of operator schemas, I is a set of initial state facts, G is a set of goal propositions and C is a set containing all of the constants appearing in I∪G.*

**Definition 3.2** *A* plan, *p, for a planning problem, (O,I,G,C), is a sequence of operator schemas from O each instantiated with constants from C such that each instantiated operator schema can be applied from the preceding state, starting at I, and the final state satisfies G.*

The first symmetry we consider is the form explored by Joslin and Roy [Joslin and Roy, 1997] and subsequently considered by Rintanen [Rintanen, 2003].

**Definition 3.3** *A* configuration symmetry *of a planning problem P = (O,I,G,C) is a group, S, acting on the constants in C, such that under the action of the elements of S, the initial and goal states are invariant and every plan for P remains a valid plan for P.*

This definition requires that a plan should remain valid under the action of elements of the symmetry $S$, but it does not require that the plan should be invariant. For example, consider the simple blocks problem illustrated in figure 1. The group $\{id, (CE)(DF)\}$[1] is a configuration symmetry. However, although valid plans will remain valid, plans will not be invariant under the transpositions of C with E and D with F, since this will rearrange the steps of a plan.

[1]The identity element is denoted by *id*. The elements are expressed in the standard cycle notation for groups.

A special subclass of configuration symmetries contains those symmetries that have proved to be amongst the most fruitful in existing exploitations of symmetries [Fox and Long, 1999; 2002]. This is the class that is generated by all pairwise transpositions of a set of symmetric objects:

**Definition 3.4** *A configuration symmetry, $S$, of P = (O,I,G,C), is a* functional symmetry *if it is a permutation group, $S_n$, acting on a subset of $n$ of the constants in C.*

## 3.1 The Property-based Abstraction

All symmetries can be seen as a consequence of some form of abstraction. The abstraction renders certain groups of actions, objects or structures equivalent by not specifying the details of the relationships that differentiate them. In some problems, applying an abstraction to remove the distinctions between objects can lead to the discovery of new potential symmetries that are not present in the base domain. It is this observation that leads us to make the following definition:

**Definition 3.5** *An* abstraction *of a planning problem, P, is a mapping, f, from P onto a new planning problem, Q.*

This definition is deliberately permissive, since we do not wish to exclude any possible abstractions. In general we will be considering compositional mappings in which the components of Q are constructed by applying an abstraction independently to the different components of P. The following definition formalises the property-based abstraction described earlier. Properties were first defined in [Fox and Long, 1998]. A property of an argument to a predicate is the pair containing the predicate name and the argument position index (this is denoted below by subscripting the predicate name with the argument position).

**Definition 3.6** *The following mapping is the* property-based *abstraction for planning problem $P$.*

*For each proposition in a planning problem, P, formed from predicate $p$ applied to $k$ arguments, $x_1, \ldots, x_k$, $f(p(x_1, \ldots, x_k)) = \{p_1(x_1), \ldots, p_k(x_k)\}$. The $k$ unary target predicates, $p_1, \ldots, p_k$ are called* properties.

The property-based abstraction simplifies the structure of a problem by removing the linkage between pairs of objects and considering the problem as the union of separate projections of the original problem for each constant. This has the effect of abstracting out which objects play which roles with respect to other objects.

If we apply an abstraction to a planning problem and then identify a symmetry in the abstracted problem, this symmetry can have a useful relationship to the original problem:

**Definition 3.7** *Given a planning problem, P, and an abstraction, $f$, mapping P into the planning problem Q, a symmetry of Q is called an* almost symmetry *for P with respect to the abstraction $f$.*

Abstractions can throw away so much of the structure of a problem that the abstracted domain has no useful connection with the original domain. In practice we are interested in abstractions that preserve significant structure from the original problem. One way in which structure might be preserved is in the behaviour of plans under an abstraction:

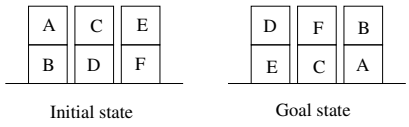


Figure 1: A simple example blocks problem

**Definition 3.8** *An abstraction, $f$, of planning problem P is* plan preserving *if, when p is a plan for P, $f(p)$ is a plan for $f(P)$.*

The property-based abstraction of a problem is not necessarily plan preserving because each property is available only once in the abstracted domain and once it is deleted it cannot be replaced. Thus, in cases where a set of propositions collapses to a single property there may be no solution to the abstracted problem.

**Definition 3.9** *An almost symmetry of planning problem P with respect to the abstraction $f$ is a* strong (weak) almost symmetry *if $f$ is (is not) plan preserving.*

In the case where $f$ is the identity, any symmetry on $P$ with respect to $f$ is, trivially, a strong almost symmetry.

The property-based abstraction is just one abstraction function that reveals exploitable almost symmetries. In general, the abstraction process can remove important constraints from the original problem, so can only act as a guide to the solution of the original problem: indeed, even for use as a guide, an abstraction must be chosen with care. For this reason, almost symmetries must be handled cautiously: pruning a search space using an almost symmetry is very likely to compromise completeness.

Our starting point for identifying almost symmetries between objects is the method used by Joslin and Roy [Joslin and Roy, 1997]. For a given planning problem we build a graph representing the object relationships in the initial and goal states of the problem and then use NAUTY [McKay, 1990], the graph automorphism discovery tool, to identify automorphisms in the graph. The key difference in our approach is the way in which the *graph* is constructed. Given a planning problem, our approach identifies a subset of the almost symmetries of the domain. We construct a coloured graph that represents an abstraction of the problem under the property-based abstraction. This construction is performed by the *makeGraph* algorithm shown in figure 2. NAUTY is then used to find the symmetries in the abstracted problem. The colouring of the nodes distinguishes between the properties that arise in the property-based abstraction described in definition 3.6. Thus, NAUTY finds the almost symmetries of the problem with respect to the property-based abstraction.

The following theorem guarantees that *makeGraph* generates an almost functional symmetry with respect to the property-based abstraction. The proof is omitted for brevity.

**Theorem 3.10** *The automorphisms of the graph makeGraph(P), for planning problem P, restricted to the constants in P, form an* almost functional symmetry *with respect to the property-based abstraction of P.*

```
1: makeGraph(P)
2: input: planning problem P = (O, I, G, C)
3: output: coloured graph N
4: initialise an empty graph N.
5: for all c ∈ C do
6:    create vertex v_c with colour equal to its type
7:    add v_c to N
8:    initialise an empty set of propositions, Q_I
9:    for all propositions p ∈ I do
10:      if p contains c then
11:         add p to Q_I
12:   let props = the bag of properties of c in Q_I
13:   create a vertex v_{Q_I} with colour equal to (|Q_I|,props)
14:   {the names of the properties in props identify the arities of
      the predicates and the types of their arguments}
15:   add v_{Q_I} to N
16:   create an edge e between v_c and v_{Q_I}
17:   add e to N
18:   initialise an empty set of propositions, Q_G
19:   for all propositions p ∈ G do
20:      if p contains c then
21:         add p to Q_G
22:   let props = the bag of properties of c in Q_G
23:   create a vertex v_{Q_G} with colour equal to (|Q_G|,props)
24:   add v_{Q_G} to N
25:   create an edge e between v_c and v_{Q_G}
26:   add e to N
27: return N
```

Figure 2: The construction of the coloured graph.

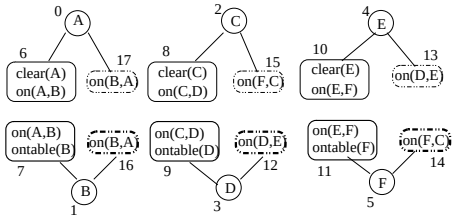


Figure 3: The graph constructed by makeGraph for the blocks example. Nodes with the same shape and outline have the same colour and are numbered 0-17.

Having identified the almost symmetries that exist at the object level in the domain we can construct the almost symmetric relation on actions, as described above. The application of a property-based abstraction, as described in definition 3.6, allows the identification of the symmetry in figure 1 between the blocks on the tops of the stacks and between the blocks on the bottoms of the stacks. In such an interpretation it can be seen that the blocks A, C and E, which are *on* other blocks and therefore at the tops of the three stacks, will move to being at the bases of the three new stacks in the goal. Similarly, B, D and F will move from being beneath other blocks to being at the tops of the new stacks. By abstraction, the fact that these groups of blocks start in different positions has been made irrelevant and the symmetry in their behaviour becomes apparent.

The coloured graph that we construct for this problem is shown in figure 3. The round nodes are the 6 blocks in the problem. These nodes have the same colour because the

blocks are all of the same type (so can be considered to share the same unary type predicate). Each of these nodes is connected to the set of propositions that mention the corresponding block in the initial state, and the set of propositions that mention the block in the goal state. It can be seen that when the properties obtained from the propositions in these sets are different the corresponding nodes have different colours. For example, nodes 16 and 17 have different colours because, although they contain the same proposition, the colour of node 16 contains the property $on$ with respect to block A (the property $on_2$) while the colour of node 17 contains the property $on$ with respect to block B (the property $on_1$).

The output from NAUTY is the generators for the graph automorphism group. When restricted to domain objects, the output for this problem is: vertices 0, 2, 4 or $\{A, C, E\}$ and vertices 1, 3, 5 or $\{B, D, F\}$

The almost symmetry that is obtained by first applying the property-based abstraction to this blocks world problem, and then applying NAUTY to the resulting graph, belongs to the abstracted problem in which the detailed relationships between particular blocks have been removed. It can be seen that the blocks A, C and E are interchangeable in the property-based abstraction, as are the blocks B, D and F. Interestingly, in this particular simple example these collections of blocks are interchangeable in the plan as well, so that the property-based abstraction is plan preserving in this case and the almost symmetry is strong (as defined in definition 3.9).

## 4 Proactive Symmetry Exploitation

In most existing work exploiting symmetry in search problems, the exploitation has been *negative*, in the sense that it is used to prune the search space in order to avoid searching symmetrically equivalent parts of the search space. This is useful when one part of the space has been searched fruitlessly, since the symmetry implies that the symmetrically equivalent parts of the space will also prove fruitless. Various approaches can be used to prune the search space, including the introduction of symmetry breaking constraints into the problem encoding itself (this technique is used in the planning context in [Joslin and Roy, 1997; Rintanen, 2003]), by forcing an ordering between otherwise symmetric choices for example, and monitoring choices in the search machinery itself, pruning equivalent choices as they arise (the approach followed in [Fox and Long, 1999; 2002; Long and Fox, 2003]). However, in the context of certain heuristic search strategies the utility of these approaches is significantly reduced. In particular, the forward search approach of FF effectively breaks symmetries by not backtracking over action choices. Although some benefit might be obtained by avoiding the consideration of symmetric action choices during a breadth-first search of a plateau, pruning symmetric branches appears likely to be less useful in heuristic search planners than in systematic search planners.

In order to exploit symmetry in heuristic search planners, we propose an alternative strategy to the pruning approach: in cases where action choices are available that are symmetric with choices that have already been made and adopted, we propose to encourage those choices.

```
1:  actionSelect(h,G,As,H)
2:  input: The heuristic value of the current state h, a symmetry G,
    a set of proposed actions As, a plan head H
3:  output: An extended plan head H'
4:  initialise a vector vs of counts, one for each action in As.
5:  for all actions a in As do
6:     for all actions b in H do
7:        if a uses object c and b uses object d and (b d) ∈ G then
8:           increment vs[a]
9:  sort As according to vs
10: for all a in As do
11:    if heuristic value of a > h then
12:       H' = H+a
13:       return H'
```

Figure 4: A modified version of FF favouring symmetric action choices.

## 5 Exploitation of symmetry information

In the work described in this paper we have explored a simple strategy where the symmetry information was used as a *heuristic guide* for selection between proposed action choices. During plan generation FF performs a forward state space search and at each stage proposes actions to apply at the current state. We amend the action selection strategy of FF to favour actions that are almost symmetric to actions selected earlier in the plan. This is a positive exploitation of symmetry because we use the symmetric structure of the problem to propose rather than to prune action choices.

The heuristic prefers actions that use objects that are symmetrical to objects that have appeared in *the same* actions earlier in the plan. This exploits the observation that where symmetric objects each require treatment in a plan they are likely to require symmetric treatment.

This heuristic is straightforward to implement in FF. At each stage during plan generation we have a current plan which consists of the actions applied so far to get from the initial state to the current state. At this stage a set of possible next actions are proposed by FF. These are the helpful actions, all of which occur in the first layer of the relaxed plan from the current state and are immediately applicable in the current state. FF normally chooses between these by applying its relaxed distance measure to the states that they produce and choosing the first helpful action that produces the state closest to the goal. We modify the strategy that FF uses for selecting between these by ordering the helpful actions so that those that use objects that are symmetrical to any that have appeared in actions in the plan so far are visited first in the selection strategy. The consequence of this is that only actions that improve the heuristic distance will be selected by the modified strategy. We do not modify the heuristic estimate itself. To order the actions we record, for each action, a count of the number of symmetric objects it uses and sort the actions into descending order of this count. Any ties are broken using FF's standard action selection strategy. The necessary modifications to the action selection step are shown in figure 4.

To illustrate the use of the symmetry information in FF version 2.3 (ff-v2.3), consider a simple Depots example (pfile3 in the IPC3 archive). The following sets of objects are found

| Action | Symmetric args | Score |
|---|---|---|
| (a) | (distributor1,depot0) | 1 |
| (b) | (distributor1,depot0), (depot0,distributor1) | 2 |
| (c) | (hoist2,hoist0) | 1 |
| (d) | | 0 |
| (e) | (hoist1,hoist0), (crate4,crate5) | 2 |

Table 1: Scoring helpful actions for symmetric arguments.

to be symmetric according to their properties in the initial and goal states of the problem: {*pallet0, pallet1, pallet2*}, {*crate4,crate5*}, {*crate0, crate2*}, {*truck1, truck2*}, {*depot0, distributor1*} and {*hoist0, hoist1, hoist2*} At some intermediate point during plan generation the plan contains the following actions:

*drive(truck1,distributor0,depot0)*
*drive(truck1,depot0,distributor1)*
*drive(truck0,depot0,distributor1)*
*lift(hoist0,crate5,crate2,distributor1)*
*load(hoist1,crate5,truck1,distributor1)*

The set of helpful actions proposed by FF at this point in the plan contains:

(a) *drive(truck1,distributor1, distributor0)*
(b) *drive(truck1,distributor1,depot0)*
(c) *lift(hoist2,crate2,pallet2,distributor1)*
(d) *lift(hoist0,crate1,pallet0,depot0)*
(e) *lift(hoist1,crate4,crate3,distributor0)*

The scores associated with these actions are obtained by counting the number of corresponding pairs of symmetric arguments between each action and an occurence of the *same action* in the plan so far. In scoring, we do not increment the score in the case where the argument in the helpful action is identical to the corresponding argument in the earlier action as we want to encourage acting on *other* objects in the same symmetric set.

We now consider the helpful actions in turn, scoring them, as shown in figure 1. The heuristic must choose between the two actions weighted 2. In this case the lift action is selected (it is likely that the competing drive action was not highly valued by the distance heuristic because it undoes the effect of an immediately preceding step).

It can be seen that our strategy is forcing the planner to commit earlier to actions that have already proved successful for symmetrical objects (we needed to lift up $crate5$ and, since $crate4$ and $crate5$ are symmetric it may be a good idea to do the same thing with $crate4$).

## 6 Experimental Results

In this section we report results of experiments that compare the performance of the planner ff-v2.3 [Hoffmann and Nebel, 2001; Hoffmann, 2002] with a version of the same planner that uses symmetry information as a positive heuristic in deciding which action to apply next. We refer to this planner as *ff-v2.3+symm*. We consider six domains: two of these are artificially constructed to contain a high degree of directly exploitable symmetry, whilst the remaining four are more natural domains, taken from the IPC3 competition bench marks. In the latter group (Depots, Driverlog, Rovers and Freecell), there is very little directly exploitable symmetry — existing

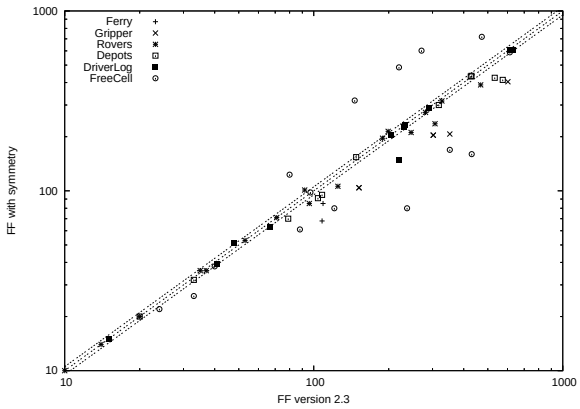


Figure 5: Comparison of states visited.

techniques discover at most 4 symmetric objects in any of these problems. Our results show that we obtain significant advantages in both the artificial and the natural domains. The Gripper results we present are for randomly generated problems in which the number of balls and the number of grippers both vary. The Ferry problems are randomly generated with varying numbers of ferries and cars. Increasing the numbers of grippers and ferries in these problems reduces the number of times that a *move* or a *sail* is the only helpful action and therefore increases the relative density of choicepoints at which symmetric actions are available. We begin by presenting graphs of the results comparing numbers of states explored and time taken (which includes the time required to carry out the symmetry analysis itself).

Figure 5 shows the comparison of the performances of *ff-v2.3* and *ff-v2.3+symm* in terms of the number of states visited in the search. The plot is log-scaled and the middle of the three lines represents equal performance. The other lines represent a $10\%$ performance difference either side of equal. Again, points below the line represent a performance advantage for *ff-v2.3+symm*. It can be seen that five of the Freecell problems and one Rovers problem (six problems in total) were solved at least $10\%$ more efficiently by *ff-v2.3*. By contrast, twenty eight problems were solved at least $10\%$ more efficiently by *ff-v2.3+symm*.

Figure 6 shows the comparative time performance on problems taken from the six domains. In this plot, and in the statistical reported below, we exclude instances solved in under one second. There are two reasons for this: firstly, noise effects in the measurements are more severe in values this small and, secondly, the implementation of our symmetry analysis is not optimised for speed, making its overhead a relatively distorted penalty for these small problems. The line represents equal performance and points below the line represent better performance by *ff-v2.3+symm*. It can be seen that *ff-v2.3* outperforms *ff-v2.3+symm* in only four cases.

To confirm statistical significance of our results, we have used the Wilcoxon-Mann-Whitney matched pairs signed ranks test. This test is more appropriate than a matched pairs t-test, since the distribution of the differences in performance

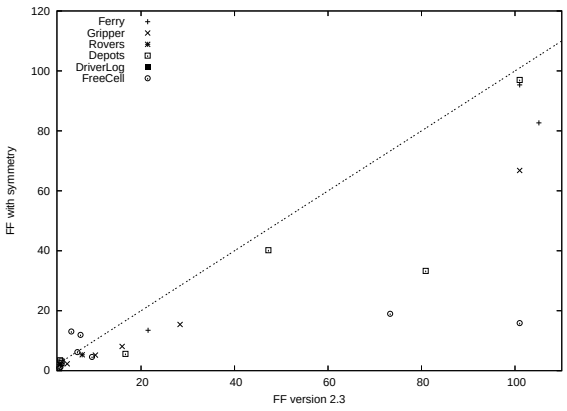


Figure 6: Comparison of time performance.

| | | Time | | | States visited | | |
|---|---|---|---|---|---|---|---|
| Test | orig | spl | Z | p | spl | Z | p |
| 6 doms | 82 | 26 | 3.43 | 0.0006 | 65 | 4.14 | <0.0004 |
| 4 doms | 69 | 17 | 2.18 | 0.0292 | 55 | 3.12 | 0.0018 |
| Symm | 11 | 9 | 2.72 | <0.05 | 10 | 2.80 | <0.05 |

Table 2: Results from matched pairs ranked signs tests showing sample size, Z-value, and significance (all less than 5%). Tests are for all domains (6 doms), 4 IPC3 domains (4 doms) and Gripper+Ferry (Symm).

is certainly not normal: where problems are solved with little search the potential advantage is restricted, while it can be larger in harder problems. The matched pairs signed ranks test is more sensitive and more robust in cases where the underlying distribution is not known. We show, in figure 2, that in both sets of domains the performance of *ff-v2.3+symm* is statistically significantly better than that of *ff-v2.3*, both in terms of time taken and in terms of number of states visited. For each of the tests we performed we provide the sample size, the Z-value obtained (using the Wilcoxon-Mann-Whitney test) and the corresponding p-value. Since equal performance cases are discarded in performing the test, we also report the original sample size (including equal performance cases), for comparison. Since the Gripper and Ferry domains contain artificially high degrees of symmetry, we perform the tests both including and excluding these domains (although note that only 10 problems were considered for these domains). We do not report results for individual domains, but it is interesting to note that these are not significant, indicating that the effects of almost symmetry across the whole problem set are not the consequence of strong performance in a subset of the domains, but are distributed across all of them.

## 7 Conclusions and Further Work

In this paper we have introduced a method for extracting almost symmetries from a planning problem and exploiting them proactively in a forward search planner. We have presented results showing that proactive symmetry exploitation can improve the search performance of a planner and sometimes result in the discovery of higher quality plans. Specifically, using symmetry information to inform the heuristic selection of the next action can reduce the number of states expanded during search, the number of steps added to the plan and also the overall time taken to generate a solution plan.

A useful aspect of the proactive approach is that it can be combined with negative symmetry-breaking techniques. It is feasible to combine the action choice heuristic with a completeness preserving pruning strategy such as has been exploited by Rintanen [Rintanen, 2003] and Fox and Long [Fox and Long, 1999], since these approaches are not in any way mutually exclusive.

## References

[Audemard and Benhamou, 2002] G. Audemard and B. Benhamou. Reasoning by symmetry and function ordering in finite model generation. In *Proc. 18th Int. Conf. on Automated Deduction (CADE-18)*, volume 2392 of *LNCS*, 2002.

[Fox and Long, 1998] M. Fox and D. Long. The automatic inference of state invariants in TIM. *JAIR*, 9, 1998.

[Fox and Long, 1999] M. Fox and D. Long. The Detection and Exploitation of Symmetry in Planning. In *Proc. of the 16th Int. Joint Conf. on AI (IJCAI)*, 1999.

[Fox and Long, 2002] M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *Proc. of AIPS'02*, 2002.

[Fox and Long, 2003] M. Fox and D. Long. An extension to PDDL for expressing temporal planning domains. *Journal of AI Research*, 20, 2003.

[Gent and Smith, 2000] I. P. Gent and B. Smith. Symmetry breaking during search in constraint programming. In *Proc. of ECAI*, 2000.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14, 2001.

[Hoffmann, 2002] J. Hoffmann, 2002. The FF-v2.3 planner is available to download from http://www.informatik.uni-freiburg.de/∼hoffmann/ff.html.

[Ip and Dill, 1996] C. Norris Ip and David L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9, 1996.

[Joslin and Roy, 1997] D. Joslin and A. Roy. Exploiting symmetry in lifted CSPs. In *Proc. of 14th National Conf. on AI (AAAI-97)*, 1997.

[Long and Fox, 2003] D. Long and M. Fox. Plan permutation symmetries as a source of planner inefficiency. In *Proc. of UK Workshop on Planning and Scheduling*, 2003.

[McKay, 1990] B. McKay. *Nauty* Users Guide 1.5. Technical Report TR-CS-90-02, Australian National University, 1990.

[Rintanen, 2003] J. Rintanen. Symmetry reduction for SAT representations of transition systems. In *Proc. of the 13th Int. Conf. on Planning and Scheduling*, 2003.

[Roney-Dougal *et al.*, 2004] C.M. Roney-Dougal, I.P. Gent, T. Kelsey, and S. Linton. Tractable symmetry breaking using restricted search trees. In *Proceedings of ECAI*, 2004.

[Roy and Pachet, 1998] P. Roy and F. Pachet. Using symmetry of global constraints to speed up the resolution of CSPs. In *Workshop on Non-binary Constraints, ECAI*, 1998.