Plan Permutation Symmetries as a Source of Planner Inefficiency

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

Abstract

This paper briefly reviews sources of symmetry in planning and highlights one source that has not previously been tackled: *plan permutation symmetry*. Symmetries can be a significant problem for efficiency of planning systems, as has been previously observed in the treatment of other forms of symmetry in planning problems. We examine how plan permutation symmetries can be eliminated and present evidence to support the claim that these symmetries are an important problem for planning systems.

Introduction

Symmetry is a phenomenon that arises in a wide range of combinatorial problems, such as CSPs (Gent & Smith 2000; Backofen & Will 1999; Joslin & Roy 1997; Crawford *et al.* 1996), model-checking (Ip & Dill 1996) and other areas (Audemard & Benhamou 2002). It is also common in planning problems. Where combinatorial problems that are tackled by systematic search techniques (including heuristic search techniques) symmetry is a source of inefficiency because the search can examine different choices that are symmetrically equivalent. This inefficiency can be exponentially expensive, since it is common for symmetries to give rise to factorial-function factors in the size of search spaces, resulting from the permutations of symmetric choices that can be searched independently of one another. Several authors have explored the possibilities for eliminating symmetries from planning problems (Joslin & Roy 1997; Fox & Long 1999; 2002; Rintanen 2003; Long & Fox 2003). In this paper we examine the sources of symmetry in planning problems and highlight one that has not been previously addressed: *plan permutation symmetry*. We explore the elimination of this form of symmetry and demonstrate that it can lead to improvements in performance.

Sources of Symmetry in Planning Problems

A symmetry is a function that maps the structure of a problem into itself. That is, a symmetry is an automorphism of the problem definition.

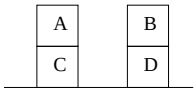

Figure 1: A simple initial state configuration for four blocks.

One of the primary sources of symmetry in planning problems is the naming of distinct objects that are all functionally equivalent. For example, if a problem involves transporting cargoes from one location to various destinations, the distinct cargoes will be named and distinguished, although any cargoes that start at the same place and must end at the same place, with no distinguishing features other than their names, are symmetrically equivalent. This symmetry has also been called *functional equivalence* (Fox & Long 1999), because it is a relationship between objects that can substitute one for another in their functional roles within a plan. This form of symmetry leads to simple symmetry groups, consisting of complete permutation groups on the objects that exhibit the symmetric qualities.

A second source of symmetry is a generalisation of the first: configurations of objects can be symmetric with one another. For example, in Figure 1 the blocks A and B are not symmetric with one another because a simple exchange of these two blocks leaves the initial state containing the propositions $\{on(A, D), on(B, C)\}$ in place of $\{on(A, C), on(B, D)\}$. However, a mapping that exchanges A with B and C with D *is* a symmetry, since it acts as an identity mapping when applied to the initial state. Symmetries of this kind can lead to more complex symmetry groups. Research on these symmetries in planning is reported in (Joslin & Roy 1997).

A third source of symmetry takes a different form altogether. In many problems there are collections of plans that are equivalent up to ordering of some subsequences of their actions. For example, in Figure 2, the plan $A; B; C$ is equivalent to the plan $B; A; C$. The automorphism that makes this symmetry is the mapping which swaps the names and bodies of actions A and B. This form of sym-

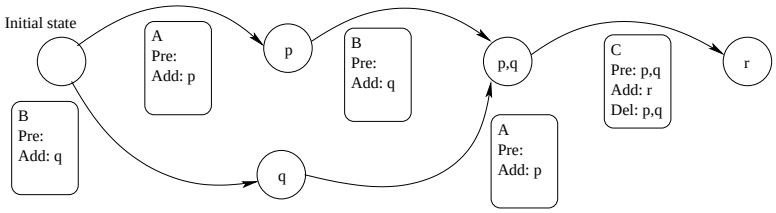

Figure 2: Symmetry between plans.

metry is related to the symmetries in transition systems explored by Emerson and Sistla (Emerson & Sistla 1996). It is also closely associated with the possibility for concurrency in plans, such as is considered in the Graphplan system (Blum & Furst 1995) or in partial-order planners (Barret *et al.* 1996). Any set of action instances that are applied concurrently within a plan can obviously be permuted without changing their collective behaviour, since the actions in a single time point are not considered to be ordered in any particular way (although some researchers have chosen to adopt the converse view that actions *can* be sequenced at a single time point (Bacchus & Kabanza 1998; McDermott 2003). Non-interference between actions is most apparent when the actions involved refer to propositions that form entirely disjoint sets. In principle, a planning problem that can be decomposed into disjoint sub-problems will induce a symmetry in which the disjoint components are freely permuted. However, not all such decompositions form the basis for symmetries since the disjoint components may not be structured in a way that allows a mapping between them that induces mappings between plans. Emerson and Sistla considered automorphisms on transition structures in which the symmetries are described by permutations on states and transitions (relations on pairs of states). The permutation of disjoint sub-problems is not of this form, since it is not always the case that the mapping can be extended to permute individual transitions.

Symmetry between different orderings of actions is tantalising, since there are many problems in which inefficiency is introduced by considering actions in different orders when the order does not actually matter. Commuting actions are an obvious cause of this form of symmetry. Unfortunately, it is often the case that actions can be permuted without impact on the plan, but the actions involved do not commute. This happens when the effects of the actions that change according to the order of execution are irrelevant to execution of the plan or to achievement of the goals. For example, if the domain illustrated in Figure 2 were to also contain an action, D, with precondition p then the transposition of A and B is no longer a symmetry, since the plan $A; D$ would be mapped to the invalid structure $B; D$.

We call symmetries of this third form *plan permutation symmetries*. We now proceed to consider the possibilities for detecting and eliminating these symmetries.

Identifying and Eliminating Plan Permutation Symmetries

Work on symmetry elimination in the context of CSPs has mainly concentrated on managing symmetries once they have been supplied to the system, rather than on identifying them automatically. Symmetries are generally described by adding appropriate additional constraints to the description of a problem. In planning, particularly domain-independent planning, the emphasis has been on using an unadorned problem description. This places a far greater importance on the automatic identification of symmetries and techniques that have been used to perform this task have been based on the identification of type structure (Fox & Long 1999) or finding graph automorphisms on graphs describing initial and goal states of problems (Joslin & Roy 1997).

The identification of plan permutation symmetries in general is a difficult problem. However, a subset of the symmetries are relatively easy to find: any pair of actions that will commute in all situations leads to a symmetry between plans in which the pair of actions are both present and adjacent to one another, either applied simultaneously or sequentially. Actions commute in all situations if they are non-mutex. The following definitions are taken from the account of the semantics of PDDL2.1 (Fox & Long 2003).

Definition 1 Ground Action *Given a planning instance, I, containing an action schema $A \in As_I$, the set of* ground actions for A, GA_A, *is defined to be the set of all the structures, a, formed by substituting objects for each of the schema variables in each schema, X, in $flatten(A)$ where the components of a are:*

- Name *is the name from the action schema, X, together with the values substituted for the parameters of X in forming a.*
- Pre_a, *the* precondition *of a, is the propositional precondition of a. The set of ground atoms that appear in* Pre_a *is referred to as* GPre_a.
- Add_a, *the* positive postcondition *of a, is the set of ground atoms that are asserted as positive literals in the effect of*

a.

- Del$_a$, *the* negative postcondition *of* a*,is the set of ground atoms that are asserted as negative literals in the effect of* a.
- NP$_a$, *the* numeric postcondition *of* a*, is the set of all assignment propositions corresponding to the numeric effects of* a.

The following sets of primitive numeric expressions (PNEs) are defined for each ground action, $a \in GA_A$*:*

- $L_a = \{f | f$ *appears as an lvalue in* $a\}$
- $R_a = \{f | f$ *is a PNE in an rvalue in* a *or appears in* $Pre_a\}$
- $L_a^* = \{f |$ f *appears as an lvalue in an additive assignment effect in* $a\}$

Definition 2 Mutex Actions *Two grounded actions,* a *and* b *are* non-interfering *if*

$$GPre_a \cap (Add_b \cup Del_b) = GPre_b \cap (Add_a \cup Del_a) = \emptyset$$
$$Add_a \cap Del_b = Add_b \cap Del_a = \emptyset$$
$$L_a \cap R_b = R_a \cap L_b = \emptyset$$
$$L_a \cap L_b \subseteq L_a^* \cup L_b^*$$

If two actions are not non-interfering they are mutex.

Proposition 1 *Actions* P *and* Q *commute if they are non-mutex.*

Proof: Suppose that the actions are non-mutex but do not commute. This means that there is some state $S = \langle V, F\rangle$ (where V is a valuation on propositions and F is a metric valuation on primitive numeric expressions) such that $P;Q$ applied to S yields a different result to $Q;P$ applied to S. Let $P;Q$ yield $\langle V_1, F_1\rangle$ and $Q;P$ yield $\langle V_2, F_2\rangle$. Then either $V_1 \neq V_2$ or else $F_1 \neq F_2$ (or both).

Suppose that $V_1 \neq V_2$. Then there is some proposition, p, such that, without loss of generality, $V_1(p) = true$ and $V_2(p) = false$. If $V(p) = true$ then one of P or Q must have p on their delete effects in order that $V_2(p) = false$. But since $V_1(p) = true$ it must be that P deletes p and Q reachieves it. Then Add$_Q \cap$ Del$_P \neq \emptyset$, which is a contradiction. If $V(p) = false$ then a similar argument leads to the same conclusion.

Therefore, it must be that $F_1 \neq F_2$. So there is some expression, e, such that $F_1(e) \neq F_2(e)$. If neither P nor Q affects the value of e then $F_1 = F = F_2$, so at least one must affect the value. If both actions affect e then, since the actions are non-mutex, it must be that the effects are both additive effects and they must, therefore, commute, which is a contradiction. If only one of P or Q affects the value of e then, without loss of generality, let it be P. Thus, P updates e according to some expression, e' (appearing on the right of the updating effect). If Q does not affect e' in any way then it is clear that P and Q commute, so it must be that Q affects e'. But then $L_Q \cap R_P \neq \emptyset$ and this is, once again, a contradiction.

This completes all cases and allows us to conclude the proof. $\diamond$

Identification of this special case leading to plan permutation symmetries is computationally trivial and can be performed in a preprocessing phase prior to planning — indeed, this analysis is conducted by many planners already in order to properly identify situations in which concurrency may be validly exploited. We will refer to this special case as commuting-step permutation symmetries, or CSPS.

Exploitation

Exploitation of symmetries in systematic search algorithms involves pruning branches of the search that would visit parts of the space that are symmetrically equivalent to parts that have already been explored. There are two main approaches (Roy & Pachet 1998; Gent & Smith 2000): addition of constraints to a problem prior to search in order to prevent exploration symmetric branches and, alternatively, the enforcement of pruning by modification of the search algorithm itself. This latter approach admits variants, but essentially the idea is to record choices and to recognise attempts to use symmetric choices as they occur.

In the context of a planning system, addition of constraints is not straightforward unless the architecture uses a general constraint solving approach, such as a SAT-solver (for example, Rintannen uses this approach (Rintanen 2003) and Joslin and Roy also use a similar technique (Joslin & Roy 1997)). Symmetry breaking during search is generally more straightforward in a dedicated planning architecture (Fox & Long 1999; 2002). CSPS can be handled by a combination of two mechanisms. Firstly, in the case where commuting actions are considered for addition at the same time point in a plan we want to ensure that only one ordering is actually searched. This can be achieved by the common symmetry-breaking strategy of introducing a lexicograhical ordering constraint: the actions can only be tried in lexicographically ordered sequences, where the actions are assigned arbitrary positions within a predetermined ordering. Thus, if we have actions A, B and C to consider for application at the same point in a plan, we will only allow the sequences: $\langle A\rangle$, $\langle B\rangle$, $\langle C\rangle$, $\langle A;B\rangle$, $\langle A;C\rangle$, $\langle B;C\rangle$ and $\langle A;B;C\rangle$, but not the remaining three alternative orderings of pairs from the set or any of the other five orderings of the three actions. Some planning strategies will impose this restriction on the selection of actions as a natural consequence of its search organisation. However, this is not true of all planners. Consider Graphplan when it is attempting to find a collection of actions to achieve the goals p, q and r, where action A achieves p and r, B

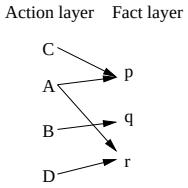

Figure 3: A Graphplan search problem that generates plan permutation symmetry.

achieves q, C achieves p and D achieves r (see Figure 3). Then the following possible sequences will be tried, in the order listed:

Noop$_p$; Noop$_q$; Noop$_r$
Noop$_p$; Noop$_q$; A
Noop$_p$; Noop$_q$; D
Noop$_p$; B; Noop$_r$
Noop$_p$; B; A
Noop$_p$; B; D
A; Noop$_q$
A; B
C; Noop$_q$; Noop$_r$
C; Noop$_q$; A
C; Noop$_q$; D
C; B; Noop$_r$
C; B; A
C; B; D

Notice that when A is used to achieve p then the usual implementations of Graphplan will recognise that r is already satisfied and not, therefore, introduce an action to achieve it separately. It can be seen that there is redundancy in this list: the second and fifth sets are subsumed by the seventh and eighth sets. Some implementations of Graphplan will check that the action set is minimal and will therefore prune the second and fifth sets (because Noop$_p$ is unnecessary when A has been added). However, it is also possible to see these sets as related by symmetry, ignoring the noop actions. A simple mechanism that can be employed to avoid this form of symmetry: to prevent two actions from being applied at the same time if they achieve a common effect. This is, in fact, correct behaviour under the semantics of PDDL2.1, in any case, in which mutexes are extended to include common effect propositions of any kind (Fox & Long 2003).

The second mechanism is concerned with preventing commuting actions from being tried in different orders *sequentially*. This is a far more subtle problem because the pair might not *both* commute with other actions that are being considered for inclusion in the plan at the same time point as one or other of the commuting pair. This means that it is not correct to simply impose a lexicographical constraint on the pair irrespective of the context in which they are applied. A slightly weaker constraint, but one that is correct, is to insist that no action may be applied in immediate sequence with a collection of (simultaneous) actions, unless it is mutex with at least one of them. That is, if an action can be applied simultaneously with a collection of actions then it cannot be applied at an adjacent time point. This constraint is only applicable to a planner that is capable of scheduling actions at the same time point, such as Graphplan variants. Enforcing this constraint ensures that an action that commutes with all actions at a given time point must be pushed to a time point that is not adjacent to them — enforcement at successive time points will prevent the action from being inserted at all until something has been added to the plan with which it does not commute. The effect of this constraint is then to make steps in a plan move forward (or backward, depending on the direction of search) through commuting sets of actions until it is adjacent to a time point at which non-commuting actions are applied (or they are at one end of the plan). In a Graphplan framework this is easy to implement as a check that an action being considered for insertion into a plan, to achieve a goal that is present as the precondition of a *noop*, is mutex with an action applied in the previous search layer (which, due to the backward search in Graphplan, will be the succeeding layer of actions). In this check we are not interested in the mutex with the noop that introduced the goal in the first place. Consider the problem depicted in Figure 3. If we suppose that action B is not mutex with any of the other actions then this constraint will prevent a plan being constructed in which B is executed in one layer and then A in the immediately succeeding layer (assuming no other steps are involved). Similarly, none of A, C or D may appear in a layer adjacent to a layer containing B (again, assuming no other actions are involved).

In the context of Graphplan, this technique must be implemented carefully to avoid a damaging interaction with no-good learning, including the EBL/DDB technique of Kambhampati (Kambhampati 2000). The reason for there being interaction is that goal sets created by gathering preconditions of actions from a layer might be determined to be unsatisfiable because of the removal of a choice of an action by the symmetry elimination. This could lead to a goal set being inapproproiately marked as unsolvable. In practice, the goal sets that can be marked in this way are very unlikely to arise in a different context during an attempt to satisfy the same original set of top-level goals.

The same kind of symmetry can be used in a sequential planner, such as FF (Hoffmann & Nebel 2001). In this case the symmetry can be eliminated by the exploitation of lexicographical constraints. The method for achieving this is to impose an artificial order on the set of commuting ac-

tions and to impose the constraint that when such actions appear adjacent to one another in a plan then they must appear in the order implied by this artifical ordering. This simply ensures that only one instance of each symmetric set of actions will be generated. FF manages to avoid the exponential growth in symmetric plan structures in this situation by recording the visited states as they are generated, but checking for revisiting states is a more expensive test than to simply ensure that commuting actions appear in the correct order.

Results

In order to confirm that elimination of CSPS in this way can achieve a useful pruning effect in a Graphplan search, we implemented the mechanism in a version of STAN (Long & Fox 1999) and explored its performance in a collection of problem instances. A common difficulty arising in the exploitation of symmetry is that the overhead of monitoring symmetries and managing the process of elimination turns out to be greater than the benefits achieved by avoiding the search in redundant parts of the search space. This can be caused, in part, if the search in the pruned branches would actually have terminated quickly in any case, due to other search control strategies working to reduce the search problem. Thus, we wanted to compare the behaviour of STAN with and without the new symmetry pruning mechanisms. It should be emphasised that our tests in STAN were influenced by the ease with which we can manipulate the behaviour of this system, but that the mechanisms we have described are not restricted to use in a Graphplan framework.

Our results confirmed that in STAN the first symmetry (action set symmetry) is already eliminated effectively by the behaviour of Graphplan. Thus, we present here, in Figure 4, only results for the second type of symmetry elimination.

The results reflect behaviour on a subset of problems from the 3rd IPC. The majority of the results show a significant reduction in the number of actions tried in the search for a plan. A few results are somewhat anomalous. Driverlog-4 and Zeno-12 lead to the generation of plans that are one layer longer than the plans generated without symmetry. This is not easily explained, and we are still attempting to uncover the precise reasons, but it appears possible that it is caused by the interaction with nogood learning. A second interesting observation is that it is not always the case that larger numbers of actions tried will automatically imply a longer time spent in search. This is somewhat surprising, but it is linked to the depth of the branches explored during the search and to the numbers of goals and mutexes generated by the actions tried. In several cases, the number of actions is dramatially reduced by the use of symmetry elimination, but the search time is not improved, or even slightly worsened. One part of the explanation for this is that the current scheme for elimination pushes actions to the end of the plan, but the search strategy favours pushing actions earlier. This means that the search can explore many branches that are doomed to fail before finally attempting one that could succeed. A better organisation of this search sequence could exploit the symmetry reduction more effectively. The results for Free-5 and Free-6 are also somewhat anomalous, the last one in particular showing a massive burden in increased action testing. We cannot explain this situation satisfactorily, at present, but are continuing to explore the behaviour in more detail.

Over all, although there are anomalies, the results show that there is a very significant potential reduction in the number of action applications searched available through the use of the symmetry elimination. We suspect that a closer analysis of the interactions with other mechanisms in STAN will reveal the reasons for the odd cases that generate significantly increased search spaces, running counter to the behaviour shown in the majority of cases.

Further Work

We wish to explore several other avenues. One is the implementation of these ideas within other planning frameworks, in order to demonstrate its versatility and broader utility. A second is to expand the forms of symmetry with which we are working. We have now considered forms of functional symmetry (Fox & Long 1999) between objects, dynamic symmetry (Fox & Long 2002) arising between objects during the development of plans and, in this paper, certain forms of plan permutation symmetry. There is clearly scope for extending the plan permutation symmetries we identify and exploit, but we are also interested in developing techniques to handle a different problem: *almost symmetric objects*. This idea is explored in more depth in (Long & Fox 2003), along with the possibilities for extension of the idea to a broader range of combinatorial search problems.

An interesting point about plan permutation symmetries is that the permutation groups that describe them are generated dynamically as the plan develops. In fact, symmetries can be added or removed as a plan extends in entirely unpredictable patterns. This is significant, because it means that none of the approaches to symmetry breaking that rely on having an *a priori* group structure can be applied (some others & Linton 2003). On the other hand, even the current efforts at recognising symmetries dynamically, such as (Fox & Long 2002), are not capable of handling this kind of highly unpredictable situation. This suggests that there is scope for more work on dynamic symmetry control and exploration of its applicability in a wider range of problems.

Although symmetry can arise very naturally in a planning domain as a consequence of there being a large number

Problem	STAN	Action trials	STAN+CSPS	Pruned	Action trials
Depots-1	0.01	27	0.01	0	27
Depots-2	0.03	607	0.03	17	440
Depots-3	0.21	17655	0.22	358	13223
Depots-4	0.4	14583	0.4	503	9640
Depots-7	0.2	8944	0.18	186	6125
Depots-10	1.51	2161	0.81	1566	35000
Depots-13	0.51	2274	0.51	59	2140
Depots-16	1.12	1432	1.12	130	1121
Depots-17	5.9	18867	5.9	1399	18913

Problem	STAN	Action trials	STAN+CSPS	Pruned	Action trials
DriverLog-1	0.01	32	0.01	0	32
DriverLog-3	0.01	946	0.01	55	767
DriverLog-4	0.26	66222	5.31	13096	1038860
DriverLog-5	0.21	60377	0.2	924	57232
DriverLog-7	0.05	4069	0.04	437	2050
DriverLog-9	0.44	50835	0.38	1967	35882
DriverLog-10	0.64	76666	0.79	3816	57874
DriverLog-11	1.45	121965	1.6	5702	117853

Problem	STAN	Action trials	STAN+CSPS	Pruned	Action trials
Zeno-1	0.01	3	0.01	0	3
Zeno-2	0.02	57	0.02	0	57
Zeno-3	0.06	181	0.06	54	217
Zeno-4	0.03	39	0.03	0	39
Zeno-5	0.1	430	0.1	8	343
Zeno-6	0.14	1155	0.14	18	207
Zeno-7	0.15	2482	0.15	217	1643
Zeno-8	0.49	313	0.49	29	365
Zeno-9	1.39	17022	0.36	65	1404
Zeno-10	8.61	393633	0.96	971	3509
Zeno-11	3.57	128164	2.15	1927	34750
Zeno-12	7.44	253302	13.48	2821	335011

Problem	STAN	Action trials	STAN+CSPS	Pruned	Action trials
Sat-1	0.01	43	0.01	0	43
Sat-2	0.06	6542	0.06	0	6542
Sat-3	0.04	1080	0.04	38	534
Free-1	0.04	386	0.04	105	773
Free-2	0.15	2363	0.15	236	4372
Free-3	0.33	6108	0.31	51	2252
Free-4	2.25	444166	2.11	1499	311353
Free-5	1.8	735	2.28	315	26947
Free-6	2.78	401	17.03	1100	1576928

Figure 4: Tables showing results for performance of STAN with and without CSPS elimination.

of functionally equivalent objects or object configurations, symmetries between objects are usually a consequence of abstractions in the description of planning domains. For example, the symmetry between the blocks in Figure 1 is a consequence of the abstraction that ignores precise positions of blocks on the table. If the positions were included then a symmetry that included the configuration of the pairs of blocks *together* with their corresponding table positions would still depend on the abstraction of the relative properties of the table positions, such as the distances to the edges of the table. In fact, symmetries in planning domains almost always follow from abstractions made by the domain engineer that describe objects in terms of just those features that the domain engineer knows to be relevant to the problem. By leaving out the irrelevant details, the domain engineer offers the planner the opportunity to exploit symmetries.

As planning domains become more complex the domain engineer's job will become harder. Appropriate abstractions will not always be obvious: to be certain of which properties are relevant or irrelevant to the solution of a given planning problem can require, in the worst case, that the problem be solved in advance. In order to support reusable domain encodings, it is also important that encodings should not be engineered only for the solution of one specific problem. This might mean that a domain should include details about the properties of objects that could have a bearing on one problem but that are irrelevant in the solution of another. For example, if the blocks in Figure 1 are only intended to be stacked in different orders then their relative positions are all that matter, but if they are also to be used in solving a problem of supporting heavy weights then their relative rigidity will be relevant, and if they are to be used in block-paving then their colour and material will become relevant.

Nebel, Dimopolous and Koehler (Nebel, Dimmopoulos, & Koehler 1997) have shown that it is possible to apply filtering techniques (RIFO) to isolate the relevant features of a planning problem using a static analysis on the domain prior to planning. Using this technique prior to the identification of symmetries will reduce the possibility that symmetries are lost because of properties included in the domain that are not required in the solution of the specific problem. The approach is straightforward: we apply the filtering technique to strip out irrelevant initial state information and then apply the symmetry identification to the reduced initial state and goal state, using TIM to derive types for the reduced domain. Note that it is important to apply TIM after the domain is filtered because the type structure can relax as a result of the filtering (TIM will differentiate types of objects based on their properties, so if properties are removed then type distinctions can be lost).

Some forms of irrelevance are more subtle than those detected by the RIFO techniques. These include features that are irrelevant because they cannot make a difference to the *efficient* solution of a problem. This is closely linked to a property we call *almost symmetry*: in many planning problems, objects begin in slightly different configurations, or are required to reach slightly different configurations, but the majority of their behaviour is essentially equivalent to the behaviour of other objects of the same type. For example, consider the problem of transporting a collection of cargoes from one location to another. Suppose that the cargoes must all end up at the same place and they all start at the same place, but that they begin stacked in several separate piles. Then, the plan will involve unstacking the cargoes, loading them into the transport, delivering and unloading them. Notice that the majority of the task is the same for every cargo: the loading, delivering and unloading will have to executed for every cargo. Unfortunately, the fact that the cargoes all begin stacked in different piles will mean that there is no symmetry at all of the single-object kind, and probably very little of the object configuration kind, despite the fact that the human observer can see that there is a high degree of symmetry in the body of the problem.

A similar situation might occur in the example of the coffee cups considered earlier if the cups begin with some in a dishwasher, others in a cupboard, some in a sink requiring washing and others on a draining board requiring drying. We refer to configurations like this as almost symmetric: the objects can be made symmetric by applying an appropriate abstraction to the domain. The abstraction is not neutral, as in the case of stripping irrelevant facts, because the plan is affected by the properties we want to abstract. However, if we solve the abstracted problem then we can add a plan fragment to the beginning of the solution in order to account for the difference between the real initial state and the abstracted initial state. In doing this, it is important to observe that the greater the abstraction the more difficult it will be to find the appropriate plan prefix, and also that the plan constructed by prepending a fragment to account for the abstraction can lead to an overall solution that is less efficient that one that is constructed directly from the original problem. For example, an abstraction of the cargoes would be to suppose that their initial positions are irrelevant and treat them as though they could all be immediately loaded. The plan prefix will then have to arrange for the cargoes to be available at the right point in the sequence of loading by clearing the higher cargoes. A simple plan prefix is to simply unstack all the cargoes at the outset, so that they can be freely loaded when necessary, but this could involve steps that are unnecessary, since loading in an appropriate order will ensure that each cargo is clear in the original configuration, without having to add extra steps to unstack them all. A more complex situation arises if the cargoes can only be unstacked onto a limited set of pallets, since there might not be a plan prefix that can achieve the abstracted situation

in which all the cargoes are simultaneously unstacked.

In general, almost symmetries can be seen as symmetries constructed by adding to or removing from the properties in the initial state and goal formula. It is possible to identify the objects for which symmetry is almost present by examining the type structure, but it is harder to determine what editing of the initial state and goal formula is a safe abstraction to introduce symmetry. Notice that modification of the goal formula by addition of properties will make the problem harder (possible unsolvable) while removal of properties will require that a plan post-fix be added to manage the achievement of the abstracted goals. Similarly, removal of initial state facts will make the problem harder, while addition of new facts can create an unreachable state (such as the state in which all the cargoes are clear in the preceding example).

Exploration of an appropriate handling of this abstraction mechanism is ongoing research. We have identified the relevant concept of *n-symmetry*, which hold between two objects if they can be made symmetric by the application of n actions. We believe that for small values of n this concept leads to constrained edits of the initial state and goal formula which compromise between the opportunities to exploit symmetry and the cost of accounting for the abstraction in the final plan.

Conclusions

Symmetry in search problems is an exciting area of current research, particularly in the UK, where recently two EPSRC-funded grants have been awarded to support work in symmetry[1]. Symmetry in planning has already stimulated several pieces of work, but there is scope for much more. In this paper we have introduced the notion of plan permutation symmetry. It is a symmetry that arises in many planning problems. There is clearly work to be done on understanding the right way to exploit the symmetry elimination technique in Graphplan and, presumably, in other architectures, but the results show a preliminary indication that the symmetry is there to be eliminated and to offer gains if it can be harnessed successfully.

References

Audemard, G., and Benhamou, B. 2002. Reasoning by symmetry and function ordering in finite model generation. In Voronkov, A., ed., *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *LNCS*, 226–240. Springer Verlag.

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22:5–27.

Backofen, R., and Will, S. 1999. Excluding symmetries in constraint-based search. In *Proceedings of CP-99*, volume 1713 of *LNCS*. Springer-Verlag.

Barret, A.; Christianson, D.; Friedman, M.; Golden, K.; Penberthy, J.; Sun, Y.; and Weld, D. 1996. UCPOP v4.0 user's manual. Technical Report TR 93-09-06d, Dept. of Computer Science and Engineering, University of Washington, Seattle, WA.

Blum, A., and Furst, M. 1995. Fast Planning through Plan-graph Analysis. In *Proceedings of 14th IJCAI*.

Crawford, J.; Ginsberg, M.; Luks, E.; and Roy, A. 1996. Symmetry breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Knowledge Representation and Reasoning (KR '96)*, 148–159.

Emerson, E., and Sistla, A. 1996. Symmetry and model-checking. *Formal methods in system design* 9 (1/2).

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *Proceedings of 16th IJCAI*.

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

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

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

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

Ip, C. N., and Dill, D. L. 1996. Better verification through symmetry. *Formal Methods in System Design* 9.

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

Kambhampati, S. 2000. Planning graph as (dynamic) CSP: Exploiting EBL, DDB and other CSP techniques in graphplan. *Journal of Artificial Intelligence Research* 12:1–34.

Long, D., and Fox, M. 1999. The efficient implementation of the plan-graph in STAN. *JAIR* 10.

Long, D., and Fox, M. 2003. Symmetries in planning problems. In *Proceedings of SymCon'03*.

[1] Ian Gent and Steve Linton have been awarded a grant to explore the use of GAP to provide a group-algebra foundation on which to build symmetry pruning in CSPs. The authors have also been awarded a grant to work on symmetry in planning.

McDermott, D. 2003. Reasoning about autonomous processes in an estimated-regression planner. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'03)*.

Nebel, B.; Dimmopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proc. of 4th European Conference on Planning, Toulouse*.

Rintanen, J. 2003. Symmetry reduction for SAT representations of transition systems. In *Proceedings of the 13th International Conference on Planning and Scheduling*.

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

some others, I. G., and Linton, S. 2003. Not sure what this was called. In *Proceedings of CP-03*, LNCS. Springer-Verlag.