Symmetries in Planning Problems

Derek Long and Maria Fox

University of Durham, Durham, UK

Abstract. Symmetries arise in planning in a variety of ways. This paper describes the ways that symmetry arises most naturally in planning problems and reviews the approaches that have been applied to exploitation of symmetry in order to reduce search for plans. It then introduces some extensions to the use of symmetry in planning before moving on to consider how the exploitation of symmetry in planning might be generalised to offer new approaches to exploitation of symmetry in other combinatorial search problems.

1 Introduction

Symmetry is a phenomenon that arises in a wide range of combinatorial problems, such as CSPs [1–4], model-checking [5] and other areas [6]. 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 [3, 7–9]. In this paper we reexamine the sources of symmetry in planning problems and consider the extent to which existing techniques have addressed them. We also consider an extension of the notion of symmetry that seems particularly relevant to planning problems. We conclude by examining how symmetries in planning problems and the existing research on addressing them might be abstracted to the wider context of general search solutions to combinatorial problems.

2 Planning Problems

In this section we briefly review a characterisation of planning problems to set the context for the discussion of symmetry that follows. A planning problem is considered to include a collection of action descriptions, each parameterised by some finite number of variables. Each action has a precondition formula and characterises the effect it has on the world in which it is applied. For simplicity we here consider preconditions to be simple conjunctions of atomic propositions (formed from predicates applied to terms which may be either variables of the action or constants) and the effects to be a set of atomic propositions representing the add effects of the action and a set of atomic propositions representing the delete effects of the action. More complex actions, including actions with numeric pre- and postconditions, temporal structure and conditional effects have been explored [10], but these additional complications will not be considered in this paper.

In addition to the action descriptions a planning problem contains a characterisation of an initial state (as a collection of atomic propositions that hold at the outset) and a formula describing a goal. We will assume that this is also a conjunction of atomic propositions for the purposes of this paper. The problem description will include a collection of constants which instantiate the arguments of the predicates that are used in the construction of the propositions of the initial and goal states. These constants will be called objects. Notice that in describing a problem with a large number of objects in which each object either starts with distinct properties, or might acquire distinct properties during execution of a plan, the objects must all be given distinct identifiers.

Although temporal plans can have complex concurrent structure, plans for planning problems of the form we have described take two alternative shapes: plans may be sequences of instantiated actions or else they may be generalised to allow sequences of sets of instantiated actions, where the sets contain actions that are considered to be executed concurrently. Actions that might interfere with one another may not be executed concurrently. We return to examine this idea further later in this paper.

3 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.

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 [7], 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 [3].

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

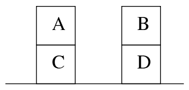

Fig. 1. A simple initial state configuration for four blocks.

names and bodies of actions A and B. This form of symmetry is closely related to the

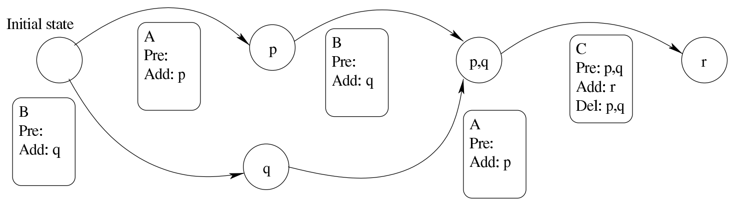

Fig. 2. Symmetry between plans.

symmetries in transition systems explored by Emerson and Sistla [11]. It is also closely associated with the notion of concurrency in plans mentioned in Section 2. Any set of action instances that are applied concurrently within a plan can obviously be permuted without changing their collective behaviour. 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. Unfortunately, it is often the case that actions can be permuted without impact on the plan, but the actions 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 $B; D$.

Rintanen [9] explores symmetries in this class. However, since planning problems are extremely compact encodings of transition systems he is unable to consider all of the possible symmetries of this form: the symmetries he actually exploits fall into the same class as those explored by Fox and Long [7].

4 Exploitation of Symmetry in Planning

Several researchers have exploited symmetries to reduce the work involved in finding plans [7, 8, 3, 9, 12]. All of this work respects one of the important themes in planning research: the objective of building a domain-independent solution to the problem of constructing plans. Planning, perhaps uniquely in the development of AI research, has maintained a focus on the objective of finding planning algorithms that do not rely on a user encoding guidance, either explicit or implicit, in the description of a problem. This may be contrasted with the CSP community, where much research has explored how best to encode domains to harness general CSP technologies effectively.

As a consequence of this theme in planning research it is not reasonable to expect that symmetries be supplied explicitly by a domain engineer in a planning problem. Therefore, one avenue of research that is particularly relevant to the exploitation of symmetry in planning has been the automatic identification of symmetries. However, finding symmetries is a hard problem and automatic techniques are unlikely to compete with human domain engineers in finding all relevant symmetries. Further, symmetries that have been found automatically will not necessarily be very important in reducing search. Human problem-solvers can be extremely effective at identifying symmetries that will actually lead to efficiency gains and ignoring smaller, less powerful symmetries, but fully-automated methods are less likely to be able to distinguish between them.

Two main techniques have been explored for finding symmetries, one focussing on finding object symmetries and the other on finding symmetries between object configurations (including single objects as a special case). The latter approach has been explored by Joslin and Roy [3], using NAUTY [13], the graph automorphism discovery tool, to identify automorphisms in the graph representing the object relationships present in the initial state and goal formula. The former approach is exemplified by the work of Fox and Long [7] where symmetries between functionally equivalent objects are discovered by a simpler analysis of the initial and goal states, based on a starting point of collections of objects of the same type (using the automatically inferred types generated by TIM [14]). This approach is very efficient and imposes an extremely low overhead on the planner, whether there is or is not symmetry in a problem. The use of NAUTY is more expensive, since the construction and analysis of the graph is much more costly than direct examination of the initial state and goal condition, but, of course, it can yield more symmetries than the TIM analysis.

Having found symmetries, researchers have each approached exploitation in slightly different, but closely related, ways. Joslin and Roy use a CSP-based approach, managing symmetry-breaking in the way that has become standard [1], by posing new constraints during search. Fox and Long modify the search algorithm of a Graphplan-based [15] planner in order to prune symmetric choices from the search space. Rintanen uses a SAT-solver, adding symmetry-breaking constraints to the SAT-encoding statically, in order to prune search. A static constraint approach suffers from the important drawback of requiring a potentially very large set of additional constraints, possibly including many that never become relevant to solving the problem, to be added to the problem description. The approach taken by Fox and Long has a low overhead during execution and is invoked only when symmetries are actually explored during the search. On the other hand, it is currently tightly integrated with the specific search engine of Graphplan.

One further development, explored by Fox and Long in [8], is the idea of dynamic symmetries. Dynamic symmetries are symmetries that arise in planning problems during the construction of a solution. To understand this better, it is helpful to consider the way that search and symmetry interact in planning. Consider a problem that involves supplying a group of people with cups of coffee with an initial state that includes a cupboard full of empty cups. The cups begin symmetric, so the choice of order in which to take cups from the cupboard and fill them with coffee is irrelevant. However, as soon as the first cup is taken from the cupboard and filled with coffee then it becomes asymmetric with the other cups. The filled cup is different from all the others because it contains coffee. The outcome of a search along a branch that considers an action involving the filled cup does not imply anything about a branch in which an empty cup is considered as an alternative. If the planner backtracks over the decision to fill the first cup then its symmetry with the other cups is restored. Thus, symmetry can be exploited to eliminate consideration of other empty cups as alternatives to the first cup in executing the action to fill the cup. If a second cup is filled, that too becomes asymmetric with the other empty cups. However, the two filled cups can now be seen to be interchangeable (assuming that we are ignoring temporal effects that might leave the first cup cooler than the second). As we fill more and more cups, we lose more and more of the original symmetries of the problem, but we gain symmetries in the growing collection of filled cups. Failure to exploit this dynamic symmetry in planning leads to a very serious deterioration in performance, since it is very common for a plan to involve making several steps with each of the symmetric objects within a plan. Without dynamic symmetries, the first choices for roles of the symmetric objects will eliminate all opportunities for further exploitation of symmetry and the same explosive combinatorial search will face the planner as would have faced it at the outset without the exploitation of the initial symmetries. To see the impact that is possible consider Figure 3, which shows two versions of the symmetry-breaking STAN planner working on Gripper problems. These require a robot with two symmetric grippers to carry a collection of symmetric objects from one room into an adjacent room. STAN v3 is only capable of exploiting initial symmetries, while SymmetricSTAN can exploit the dynamic symmetries in the problem.

5 Extending Symmetries in Planning Domains

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

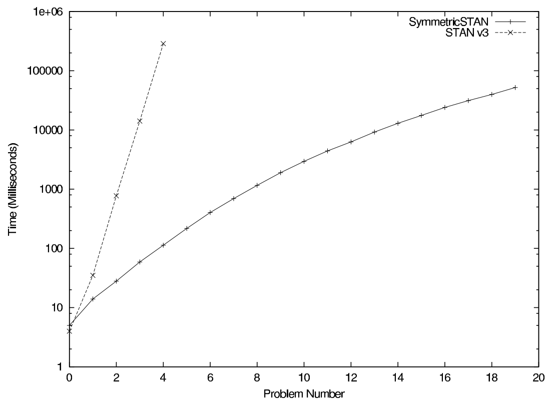

Fig. 3. Log-scaled plot showing the performance of two versions of STAN in solving Gripper problems.

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 [16] 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.

6 From Symmetries in Planning to CSPs

Planning and CSP are closely related. Several researchers have explored the use of CSP technology for planning [17, 18], including the exploitation of symmetry within that framework [3]. The usual approach is to see goal propositions as variables to which must be assigned a value corresponding to the achieving action. This model requires that the same goal proposition occurring at different times in the structure of the plan be differentiated, so it is actually a time-stamped goal proposition that is treated as a variable and, similarly, a time-stamped action that represents a possible value to assign to the variable. Within this framework it is possible to consider how the symmetries that have been described in planning domains transfer into the CSP models.

The object symmetries, both individual and configuration symmetries, induce symmetries that permute propositions and action instances (those that refer to the symmetric objects). Thus, these symmetries correspond to permutations of variables and of values in the same way as has been considered in work on symmetries in CSPs, such as [1, 2, 19]. The treatment of these symmetries within planning can then be seen as a specialisation of the approaches used in the parallel work in CSPs. The specialisations are possible because of the way that the symmetry groups are tied to the roles of objects, so that efficient symmetry-tracking can be managed by concentrating on the roles of the objects themselves, rather than on the larger sets of propositions and actions in which they appear.

Two novelties are apparent in the work on symmetries in planning. Firstly, the automatic recognition of symmetries is still at an early stage: work by Aloul *et al* [20] examines the idea in the context of SAT, and there is also some preliminary work exploring this idea in a slightly broader collection of CSP problems. The restriction of automatic detection to relatively constrained languages seems important: the more constrained forms of planning problems compared with general CSPs is probably more susceptible to automatic analysis to find symmetries. Nevertheless, finding graph automorphisms within the graphical representation of CSPs could be exploited to automate the discovery of symmetries within CSPs that would alleviate the need to explicitly code them. Secondly, the identification of dynamic symmetries is not commonly part of the exploitation of symmetries in CSPs. To use dynamic symmetries of the kind described above in CSPs using the well-established framework described in [1, 2] it would be necessary to have either an automatic symmetry discovery system that could be run online, which is unlikely to be practical, or else it would be necessary to construct the groups of anticipated symmetries in order to exploit them when they became active during the search. The difficulties in implementing such an approach are that it would no longer be possible to simply monitor symmetries incrementally to determine the point at which they broke, since symmetries might reappear or appear for the first time as the search progresses. This would mean that symmetries would have to be constantly retested against the current partial assignment in order to check whether they were valid symmetries. This cost can be managed within the restricted context in which it has been applied in planning because the ways in which symmetries (or, more properly, a subset of symmetries) can dynamically arise are highly constrained and can be managed by monitoring the states of relatively small sets of objects rather than the much larger sets of actions and propositions in which they appear. Even then, only single-object symmetries have been handled — configurations are far more problematic because of the complexity of monitoring the changing status of candidate configurations. The general problem of finding *all* symmetries in a problem, planning or not, is a hard one and it is important to maintain a constant watch on the trade-off between possible gains in search efficiency, through pruned symmetric branches, and the costs of finding and exploiting the symmetries.

Beyond the transfer of existing planning symmetries technology, it is also possible to consider the relevance of the extensions discussed in Section 5. The idea of filtering for relevance is of potential interest in CSPs. An example might be a description of the n-queens problem, complete with the colours of the squares. If the problem is to find a solution to the problem in which there are no queens on white squares in the middle $n/2$ columns of the board then the colours are relevant and they also destroy a symmetry (90^o rotation is no longer a symmetry), while they are irrelevant to the original problem. The idea of exploiting almost symmetry is also interesting: if a constraint is added to the n-queens problem stipulating that the queen in the first column must not be in the first row, then the symmetries of the problem are all broken (the constraint does not have any symmetries). However, it is quite obvious that relaxing this constraint, solving the symmetric problem exploiting symmetries and then reinstating the constraint to filter the solutions would be much more efficient than trying to solve the problem in its asymmetric form.

7 Conclusions

In this paper we have reviewed the ways in which symmetry arises in planning problems and the ways that it has been exploited. Some new proposals for exploitation of symmetry have also been outlined. We have also commented on the generalisation to CSPs of techniques used for symmetry recognition and exploitation in planning problems.

The most important theme of this paper, as far as the continued development of exploitation of symmetry in planning is concerned, is the notion of "almost symmetry". The observation that symmetry is generated by abstractions in problem descriptions leads directly to the proposal that symmetries can be discovered by applying appropriate abstractions to problem descriptions. The discovery of symmetries can offer benefits by allowing an abstracted problem to be solved efficiently and, then, an appropriate transformation applied to its solution to arrive at a solution to the original problem. We have seen that this technique could be applied in other search problems as well as in planning. In order to harness the idea it will be necessary to constrain the abstractions that are considered, both to avoid generating a new hard search problem that precedes any attempt to solve the original problem and also to avoid allowing the distance between the solution to the abstraction and a solution to the original problem to become unbridgeable. We believe that, in the context of planning problems, restricting the range of abstractions to simple graph edits on the representation of the initial state and goals, further constrained by the applicability of small chains of actions to individual objects or object configurations, will give rise to a manageable but useful set of abstractions. Our immediate research goal is to pursue this idea in detail and we intend to report our results within the symmetry community shortly.

References

1. Gent, I.P., Smith, B.: Symmetry breaking during search in constraint programming. In: Proceedings of ECAI. (2000)
2. Backofen, R., Will, S.: Excluding symmetries in constraint-based search. In: Proceedings of CP-99. Volume 1713 of LNCS., Springer-Verlag (1999)
3. Joslin, D., Roy, A.: Exploiting symmetry in lifted CSPs. In: Proceedings of 14th National Conference on AI (AAAI-97). (1997)
4. Crawford, J., Ginsberg, M., Luks, E., Roy, A.: Symmetry breaking predicates for search problems. In: Proceedings of the Fifth International Conference on Knowledge Representation and Reasoning (KR '96). (1996) 148–159
5. Ip, C.N., Dill, D.L.: Better verification through symmetry. Formal Methods in System Design **9** (1996)
6. Audemard, G., Benhamou, B.: 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., Springer Verlag (2002) 226–240
7. Fox, M., Long, D.: The detection and exploitation of symmetry in planning problems. In: Proceedings of 16th IJCAI. (1999)
8. Fox, M., Long, D.: Extending the exploitation of symmetries in planning. In: Proceedings of AIPS'02. (2002)
9. Rintanen, J.: Symmetry reduction for SAT representations of transition systems. In: Proceedings of the 13th International Conference on Planning and Scheduling. (2003)
10. Fox, M., Long, D.: An extension to PDDL for expressing temporal planning domains. Journal of AI Research **20** (2003)
11. Emerson, E., Sistla, A.: Symmetry and model-checking. Formal methods in system design **9 (1/2)** (1996)
12. Miguel, I.: Symmetry-breaking in planning: Schematic constraints. In: Proceedings of the CP'01 Workshop on Symmetry in Constraints. (2001) 17–24
13. McKay, B.: *Nauty* user's guide 1.5. Technical Report TR-CS-90-02, Austrialian National University, Canberra (1990)
14. Fox, M., Long, D.: The automatic inference of state invariants in TIM. JAIR **9** (1998)
15. Blum, A., Furst, M.: Fast Planning through Plan-graph Analysis. In: Proceedings of 14th IJCAI. (1995)

16. Nebel, B., Dimmopoulos, Y., Koehler, J.: Ignoring irrelevant facts and operators in plan generation. In: Proc. of 4th European Conference on Planning, Toulouse. (1997)
17. van Beek, P., Chen, X.: CPlan: A constraint programming approach to planning. In: Proc. of 16th National Conference on Artificial Intelligence, AAAI/MIT Press (1999) 585–590
18. Do, M.B., Kambhampati, S.: Solving planning graph by compiling it into a CSP. In: Proc. of 5th Conference on AI Planning Systems, AAAI Press (2000) 82–91
19. Roy, P., Pachet, F.: Using symmetry of global constraints to speed up the resolution of CSPs. In: Workshop on Non-binary Constraints, ECAI. (1998)
20. Aloul, F., Ramani, A., Markov, I., Sakallah, K.: Solving difficult SAT instances in the presence of symmetry. Transactions on Computer-aided Design **September issue** (2003)