The Detection and Exploitation of Symmetry in Planning Problems

Maria Fox and Derek Long
Department of Computer Science,
University of Durham, UK
maria.fox@dur.ac.uk, d.p.long@dur.ac.uk

Abstract

Many planning problems exhibit a high degree of symmetry that cannot yet be exploited successfully by modern planning technology. For example, problems in the Gripper domain, in which a robot with two grippers must transfer balls from one room to another, are trivial to the human problem-solver because the high degree of symmetry in the domain means that the order in which pairs of balls are transported is irrelevant to the length of the shortest transportation plan. However, planners typically search all possible orderings giving rise to an exponential explosion of the search space. This paper describes a way of detecting and exploiting symmetry in the solution of problems that demonstrate these characteristics. We have implemented our techniques in STAN, a Graphplan-based planner that uses state analysis techniques in a number of ways to exploit the underlying structures of domains. We have achieved a dramatic improvement in performance in solving problems exhibiting symmetry. We present a range of results and indicate the further developments we are now pursuing.

1 Introduction

STAN [Long and Fox, 1999] is a planner, based on the architecture of Graphplan [Blum and Furst, 1997], that uses a range of static state-analysis techniques to enhance its planning performance [Fox and Long, 1998]. These techniques work by giving the planner insights into the underlying structural features of the domain, and of problem instances in that domain, and making them accessible to exploitation by the planner. One of the characteristic features of problems in many domains is their underlying symmetry. A problem exhibits a high degree of symmetry if there are many functionally identical objects that cannot be usefully distinguished. For example in a construction world, in which there are hundreds of nails in a box, planners would quickly become lost in the search between alternative nail permutations in the solution of a construction instance. This search is wasted since all nail permutations are effectively equivalent. Where symmetry occurs in a problem it can be exploited in a very powerful way by treating all symmetric objects as indistinguishable. This allows the planner to avoid considering plans that differ, from those already considered, only in the specific symmetric objects referred to or in the order in which symmetric objects are manipulated.

In this paper we describe an algorithm for exploiting object and action symmetries that have been automatically extracted from a problem description. Symmetry is a feature of *problems,* rather than of domains, although some domains naturally give rise to highly symmetric problems. Our symmetry mechanism has been implemented in STAN and has yielded dramatic performance improvements across a variety of problems from standard benchmark domains. Despite the improvements that can be observed from our results we are not yet exploiting all of the symmetry that is available in a problem that displays symmetric structure. We explain why this is the case and discuss an extension to our current approach that will overcome this limitation.

2 Static Symmetry Detection

Symmetry analysis is a static analysis process that is independent of the planning architecture that will exploit the detected symmetries. Thus, although STAN is a Graphplan-based planner and the algorithm for exploiting symmetry is Graphplan-dependent, the processes by which symmetry is detected in a problem description are entirely Graphplan-independent. This is in keeping with our philosophy of providing planner-independent static analysis tools for extracting implicit structure from domain and problem descriptions which is then available, for exploitation by planners, to reduce search during plan construction.

The static analysis is in two phases. In the first phase groups of symmetric objects are automatically extracted from a problem description given as the initial and goal states expressed in the language of STRIPS [Fikes and Nilsson, 1971]. For our purposes we define symmetric objects to be those which are indistinguishable from one another in terms of their initial and final configurations. For example, in the Gripper domain, *ball* and *ball2* can be considered symmetric with one another if, in the initial state, *balk* and *$ball_2$* are both *at $room_a$* and, in the goal state, *balk* and *$ball_2$* are both *at $room_b$*. When constants appear in operator schemas they are abstracted, generating additional initial conditions which can only be satisfied by those constants. For example, the appearance of the constant *ball1* in an operator schema gives rise to the additional precondition *ball1(x),* which can only be satisfied by *ball1* using a newly created initial state condition *balli (balli).* This ensures that constants with special roles in operators cannot be mistakenly seen as symmetric to other objects in the domain.

Our strategy is to begin by identifying pairs of symmetric objects to form the bases of symmetry groups, and then to extend these groups by adding other objects from the same *types.* Types are inferred automatically by the Type Inference Module (TIM) of STAN [Fox and Long, 1998]. Type information reduces the search involved in building the object symmetry groups because objects of different types cannot be symmetric with one another and can be immediately excluded from comparison.

This analysis is sensitive to the specification of initial and goal states. Symmetry can be unnecessarily lost if redundant information is included in the problem specification. For example if, in the initial state, the *left* and *right* grippers are *free* and in the goal state nothing is specified regarding either gripper, then the grippers will be determined to be symmetric objects. However if, in the goal state, it is specified that the left gripper is free, but nothing is specified about the right gripper, then the left gripper will be assumed to have special properties that make it asymmetric to the right gripper even though, in fact, both grippers will be free by the time all of the balls have been transferred. Although this might make the symmetry analysis seem fragile, its sensitivity is actually one of its strengths. Any divergence in the states of objects is interpreted as evidence that they should not be treated as entirely indistinguishable. Except in the case where apparently distinguishing information is actually redundant this interpretation is correct.

The second stage in the process is the identification and grouping of symmetric actions: any actions whose parameters are drawn from the same collections of symmetric groups are themselves symmetric. For example, *pickup(balli,$room_a$,right)* is symmetric with *pickup(hall2,$room_a$,left)* but not with *pickupfbalk,roomt,,left) because $room_a$* and *roomb* are not symmetric. We produce action symmetry groups by pairwise comparisons of these action instances.

3 The Use of Symmetry by STAN

The exploitation of symmetry in STAN is dependent upon the graph construction and search techniques common to planners based on the Graphplan architecture. Graphplan uses constraint satisfaction techniques to search a layered graph which represents a compressed reachability analysis of a domain. The layers correspond to snapshots of possible states at instants on a time line from the initial to the goal state. Each layer in the graph comprises a set of facts that represents the union of states reachable from the preceding layer. This compression guarantees that the plan graph can be constructed in time polynomial in the number of action instances in the domain. The expansion of the graph, from which solutions can be extracted, is partially encoded in binary mutex relations computed during the construction of each layer. STAN implements an efficient representation of the graph in which a *wave front* [Long and Fox, 1999] further supports its compression. In Graphplan-style planners the search for a plan, from layer k, involves the selection and exploration of a collection of action choices to see whether a plan can be constructed, using those actions at the kth time step. If no plan is found the planner backtracks over the action choices.

The objective of the analysis and exploitation of symmetry is to reduce the number of action choices that are searched. Thus, when STAN backtracks over an action choice it avoids considering symmetric alternatives since a symmetric alternative to a given action instance can never make more progress than that action instance. For example, in the Gripper domain, if no plan to transport *n* balls to *$room_a$* can be found, ending at layer *k,* by dropping *balln* from the *right* gripper, then no plan will be found, ending at layer i, by dropping *balln* from the *left* gripper. Handling symmetry correctly involves a number of subtleties. For example, when a specific object in a symmetry group has been selected for a particular role in the plan it is no longer symmetric with the other objects in the group because it can now be distinguished from those other objects on the basis of the particular roles it plays. In our mechanism the symmetry of an object is *broken* as soon as it can be distinguished from the others in its group. On backtracking over an action choice we reinstate the symmetry of the objects whose symmetry was broken on selection of that action. We need both layer-dependent and layer-independent information to support the correct maintenance of action symmetry in the plan graph. A pseudo-code description of the Graphplan-style search algorithm of STAN, highlighting the modifications necessary to exploit object and action symmetries, is given in Figure 1. The basic algorithm should be familiar to readers conversant with the Graphplan algorithm. The bold typeface is used to indicate the extensions to the basic algorithm necessary to support the use of symmetry. The mechanism is non-heuristic and does not have to be manually selected or deselected when the planner is presented with specific domains, as is the case with some of the domain analysis techniques discussed in the literature [Nebel *et al.,* 1997]. The effects of the symmetry machinery on search completeness, which are benign, are discussed below. The overhead associated with the symmetry mechanism is discussed along with the empirical results below.

There are two key data structures involved in the correct exploitation of symmetry in STAN. One of these is

```
lb search from layer In with goal set gs:

If gs is empty
then search from layer In — 1 with the goal set constructed from
              all preconditions of actions selected at layer In
     If no plan is found
     then  reset tried groups at layer In — 1;
           backtrack to last choice point;
     else  return completed plan;
else  Consider first goal, g, goal set, gs:
           If g is still unachieved at layer In
           then For each action, a, which can achieve g at
                           layer In:
                      If symmetry group for a already
                           tried
                      then continue to next action;
                      Choose action a to achieve g;
                      Increment brokenSym entries for
                           symmetric parameters;
                      if searching from In with gs \ {g}
                           yields a plan
                      then  return the completed plan
                      else  retract action a;
                            decrement brokenSym entries for
                                 symmetric parameters;
                            mark action symmetry group
                                 as tried;
                            continue to next action;
                 When all actions have been tried, fail at this
                      point in search and backtrack;
           else  search at In with gs \ {g};
                 If a plan is found
                 then  return the plan
                 else  fail at this point and backtrack.
```

Figure 1: Core Graphplan-style Search Algorithm Indicating Modifications for Symmetry-Exploitation

layer-independent and one layer-dependent. The layer-independent structure, *brokenSym,* is a vector of integers, one entry for each domain object in a symmetry group of more than one object. Every time an action is applied the entries corresponding to the objects in the action instance are incremented. A non-zero entry indicates that the corresponding object is no longer symmetric. When an action is retracted the appropriate entries are decremented. The entries are not binary because a sequence of actions might need to be retracted in order to restore the symmetry of an object. The layer-dependent data structure is an array of matrices, *triedGroups,* one matrix for each group of symmetric actions, recording which action symmetry groups have been tried at that layer. The number of dimensions of a matrix is equal to the number of symmetric parameters referred to by actions in that symmetry group. The matrices can be allocated during graph construction. Their sizes are known at action instantiation, but they cannot be allocated at that time because it cannot be known how many layers there will be or which symmetric groups will be represented at each layer. The size of each dimension is one more than the number of objects in the corresponding collection of symmetric objects. The extra entry plays an important role in recognising symmetric actions as the following example shows.

Suppose that *drop(balli,roomb, left)* is applied at some layer n + 1. If *pickup (ball2,room$_a$,left)* is considered at layer n we would like to avoid considering *pickup(balls$_f$room$_a$,left)*' since balk 2 and 3 are symmetric so the two *pickups* should be considered as symmetric even though the symmetry of the left gripper has been broken. On the other hand, we want to try *pickup(ball2,room$_a$,right),* because the use of the left gripper at layer n + 1 might have been the cause of the failure of the first *pickup.* If we simply mark *pickups* at *room$_a$* as having been tried we will not try *pickup(ball2$_f$room$_a$,right),* which would lose completeness. If we don't mark *pickups* then we will try *pickup(ball3,room$_a$,left)* as well, which we do not want to do since this would lose the advantages of symmetry. The matrix allows us to identify the remaining symmetry in an action instance. In the above situation we mark *pickup(*,room$_a$,left)* as tried, where • stands for an arbitrary argument to the *pickup* instance, so we will not try *pickup(balls,room$_a$,left),* but will try *pickup(ball2,room$_a$,right).* Because the symmetry of *ball* was broken at layer *n*+1, so that *pickup(balli,room$_{af}$left)* is not symmetric with the other *pickups,* this action instance will also be tried. The • argument represents the collection of objects with unbroken symmetry in the appropriate object symmetry group. The extra entry in the matrix is used to record the status of the action symmetry group with the corresponding argument set to *.

The way these two data structures are used is as follows. On the point of choice of an action we check to see whether the symmetry group, to which the action being considered belongs, has been tried. If it has, we reject the action with no further search. Otherwise we mark all of its symmetric arguments as broken by incrementing the appropriate entries in *brokenSym.* During action instantiation the indices of these arguments in the *brokenSym* vector are recorded within the action instances, making access to the vector very efficient.

On backtracking through an action choice following failure, we decrement the appropriate entries in *brokenSym* and mark the action symmetry group as tried at the current layer. This is done by indexing into the appropriate matrix at the entry corresponding to the particular configuration of broken and unbroken symmetric arguments in the instance. If an argument is unbroken the index 0 is used on that dimension, representing the marking of the whole object symmetry group corresponding to the • argument discussed above. Otherwise the number of that object within its symmetry group (starting from 1) is used as the index. Since the arguments of an action are fixed when the action is instantiated, and the positions of the symmetric arguments, within their respective symmetry groups, are fixed during initial symmetry analysis, much of the work involved in identifying the correct indices is done once and for all during construction. This makes access to the matrices very efficient so that the marking and unmarking of symmetry groups represents a negligible overhead.

When backtracking through a layer the entire collection of matrices at that layer is reset to enable subsequent exploitation of symmetry to be unaffected by earlier search. *No-ops* do not break the symmetry of their arguments (because they do not cause the states of their arguments to change) and we treat each *no-op* as symmetric only with itself.

The symmetry machinery described in this paper does not yet deal with all of the symmetry that is there to be exploited in a problem. When search begins, all objects that can be identified as symmetric will be available for exploitation as symmetric objects. However, as search progresses, from the goal state towards the initial state, object symmetry is broken for increasing numbers of objects so that there is little, or no, symmetry left to exploit in the final search layers. This happens because we cannot yet recognise intermediate domain states at which the symmetry of objects could, in principle, be refreshed. In the Gripper domain, as more balls are moved into their final state, more alternative action instances become asymmetric and hence available for (useless) consideration during search. In this domain balls can be in any one of four states: at *room$_a$*> at *roomb* or held in the left or right gripper. Each of these alternatives potentially represents a *symmetry state* for balls. Only the first two of these symmetry states are interesting since only one ball can be held by any gripper at any time. We would like to be able to fully exploit these two symmetry states for balls. At present the mechanism only identifies the balls as symmetric whilst they are in their goal room because Graphplan planners search backwards from the goal state. If we could recognise being in the start room as a symmetry state for the balls we could refresh their symmetry as we progressed towards the initial state. In other domains there may be multi-state symmetries to exploit, involving many different symmetry states.

Despite the fact that we only exploit part of the symmetry in a problem we still obtain huge improvements in performance in the solution of inherently symmetric problems. In domains, such as Gripper, in which there is exponential growth in the search amongst symmetric choices, and in which objects have more than one symmetry state, we only obtain benefit from one of these symmetric states. In Gripper we obtain roughly a 50 per cent speed up. We have designed a modified Gripper domain to demonstrate the kinds of performance enhancement we obtain in a single symmetry state, and to indicate what improvements can be expected from full exploitation of symmetry. Our experiments in this, and other domains, are described later in this paper.

4 Symmetry and Search Completeness

Given a set of goals at layer *k* in the plan graph, STAN will try alternative non-symmetric action combinations in the search for a plan. If a combination fails to lead to a plan then the symmetry group of the action choice that caused the failure will be marked as tried and no subsequent combinations considered will contain any action in the tried symmetry group, even though some other combination containing that action choice might actually lead to a plan. The desired action will be successfully selected at laycfr $k+1$, in combination with no-ops to achieve the goals that were successfully achieved at layer *k*. For example, suppose that actions $o_1..o_i..o_n$ have been tried at layer fc, and that o,- belongs to symmetry group 5. Then 5 will be marked as tried at layer *k*. If $o_1..o_i..o_n$ fails the desired combination, suppose it is $o'_1..o_i..o_n$, will not be found at layer *k* because it contains an action in a tried symmetry group. This combination will be found across layers *k* and k + 1, since o'_1 will be tried at layer 4+1 and no-ops used to achieve the remaining goals achieved at layer 4. We therefore lose parallel optimal plan completeness but retain sequential optimal plan completeness. It is possible to construct problem instances in which the number of additional layers that have to be constructed outweighs the advantages of exploiting symmetry. We are therefore working on an extension to the symmetry mechanism to enable actions to be retried at a level if the search context in which they were last tried has changed.

5 Experimental Results

We have two objectives in presenting the following data: to show the advantage that the symmetry mechanism gives in solving symmetric problems and to demonstrate that no significant overhead is paid when there is little, or no, symmetry to exploit in a problem.

STAN is implemented in C++ and the following results were computed on a Sparc-10 under Unix. The results demonstrate the advantages obtained by the use of our symmetry machinery in solving a selection of problems from the Gripper domain, the simple TSP domain (a Travelling Salesman Problem on complete graphs), the Ferry domain and a modified Gripper domain. All but the last of these are standard benchmark domains. The modified Gripper domain is used to demonstrate evidence supporting our hypothesis that failure to exploit more than one symmetry state reduces our advantage proportionally with the number of symmetry states in the domain. We produced the modified Gripper domain by forcing *pickups* to be done in *room$_a$* (the start room) and drops to be done in *roomb* (the destination room). As can be observed, the exploitation of ball and gripper symmetry in this domain yields an exponential advantage over that obtained in the unmodified Gripper domain.

The data for the Ferry domain, shown in Figure 2, illustrates the benefits to be obtained from the exploitation of the symmetry of cars that must be transported from a single source to a single destination. The Ferry domain is similar to the Gripper domain in having two symmetry states for the cars, only one of which is currently being exploited by our machinery. Because of the unexploited symmetry state the performance of STAN with symmetry is still deteriorating exponentially, although the exponent is so much smaller than in STAN without symmetry that it does not become a problem until the instances are very large. Our analysis leads

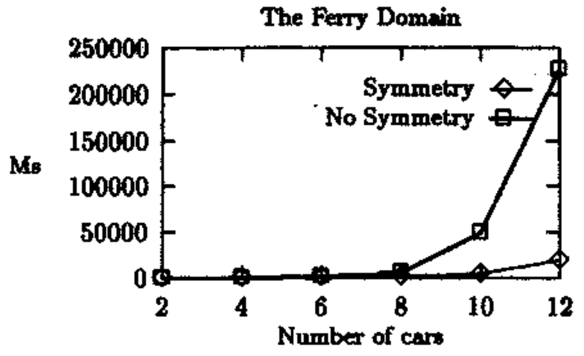

Figure 2: Comparison between STAN with Symmetry and STAN without Symmetry in the University of Washington Ferry Domain

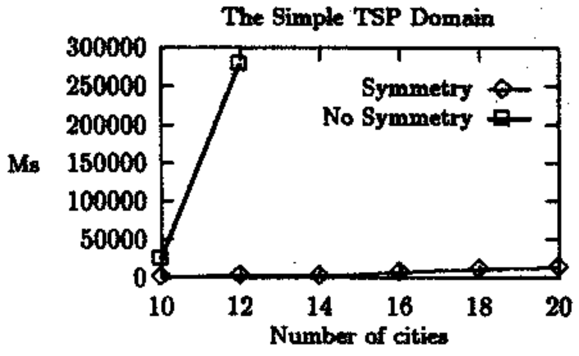

Figure 3: Comparison between STAN with Symmetry and STAN without Symmetry in the Carnegie Mellon University TSP Domain

us to conclude that exploitation of the remaining symmetry state will give us an exponential improvement in performance. This conclusion is supported by the Gripper experiments detailed below.

The simple TSP domain, involving traversal of a fully connected graph, gives rise to problems that are, in principle, trivial but that are beyond the capabilities of most Graphplan-style planners because of the n factorial permutations of the n cities to be visited. These permutations are all symmetric and Figure 3 shows that STAN with symmetry is able to exploit this feature and solve instances in linear time. In this domain STAN exploits an additional form of symmetry not yet discussed in this paper. This is *goal symmetry*, which arises when two or more goals are expressed using the same predicate and arguments from the same object symmetry groups. When two goals are symmetric and can only be achieved at the rate of one per layer, because of observed interactions between their potential achievers, STAN imposes an arbitrary ordering between them and does not search alternative orderings. This feature of certain domains is automatically detected by STAN using the invariant inference machinery discussed in [Fox and Long, 1998]. Putting the two ways of exploiting symmetry together yields substantial benefits in domains that feature this particular invariant.

The unmodified Gripper domain is the standard version designed by the IPP team and used in the AIPS-98 planning competition[1]. There are two symmetry states to exploit but we currently exploit only one of them. By modifying the domain, so that balls can only be picked up in the source room and dropped in the destination room, we dramatically reduce the significance of the symmetry state in which the balls are in the source room. This allows us to focus on the benefit obtained by exploiting the one symmetry state without being swamped by the cost of not exploiting the remaining one. Looking at Figures 4 and 5, we can observe an exponential speed up obtained by suppressing the significance of the

[1] http://ftp.cs.yale.edu/pub/mcdermott/aipacomp-results.fatal

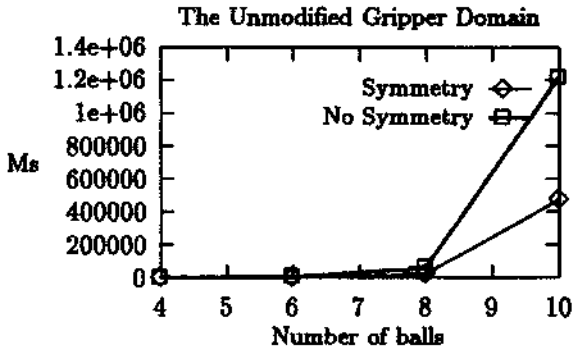

Figure 4: Comparison between STAN with Symmetry and STAN without Symmetry in the IPP Team's Gripper Domain

second symmetry state. This strongly supports our hypothesis that a full exploitation of all of the symmetry available within a family of problems can yield an exponential improvement in the performance of the planner.

Finally, we present evidence using the Logistics domain to demonstrate that the overhead of the symmetry machinery is negligible when there is no symmetry in a problem or when the symmetry cannot be effectively exploited. In the first case the data structures needed to support symmetry are not gven constructed so the overhead during search amounts to a single comparison confirming that there is no symmetry available for use at the points of action selection, retraction and backtracking over layers. In the second case the data structures are built, initialised and maintained to no positive effect. To explore problems with this character we constructed a family of Logistics problems involving the transportation of a number of packages from a single source city to a single destination city using a single airplane. We call this version of Logistics the one-dimensional Logistics domain. The packages are symmetric and the individual operator schemas yield separate action symmetry groups. However, in this family of problems there is no

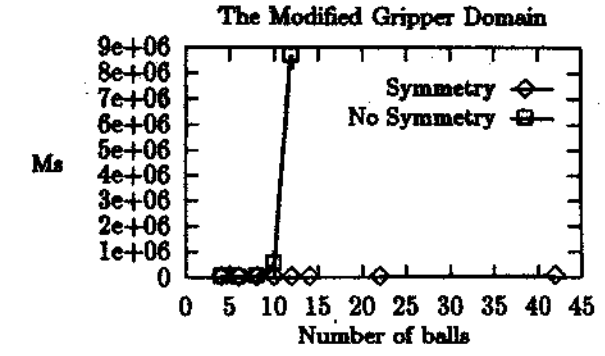

Figure 5: Companion between STAN with Symmetry and STAN without Symmetry in the Modified Gripper Domain. STAN without symmetry was terminated after 2.5 hours on the 12 ball problem.

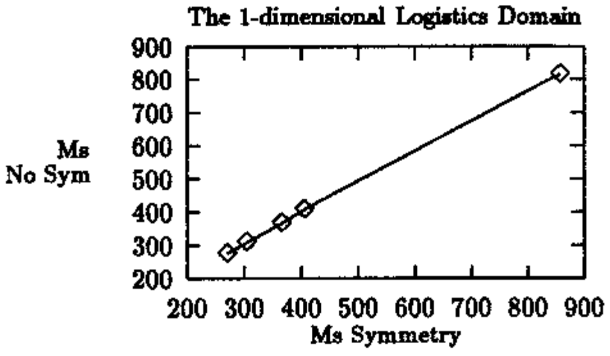

Figure 6: Comparison between STAN with Symmetry and STAN without Symmetry in the One-Dimensional Logistics Domain

interesting search involved in solving them so the maintained data structures play no useful role. As can be seen from Figure 6 the overhead in building and maintaining these structures is insignificantly low.

6 Further Developments

There are two important lines of further development under investigation. The first concerns the need to distinguish different search contexts in which symmetric action combinations can be considered. As discussed in Section 4, failure to manage contexts leads to the loss of parallel optimal plan completeness and the loss of any efficiency advantages in certain carefully constructed examples. We are experimenting with *un-marking* tried symmetry groups when backtracking over earlier action-selection choices. In the example considered in Section 4, the group *S* would be un-marked when backtracking over the choice of *o1*, on the grounds that the selection of an alternative to *o1* creates a different context in which o, might be usefully reconsidered.

The second line of development concerns the identification of symmetry states which would enable a fuller exploitation of both object and action symmetries. In order to pursue this we are examining the object state-transition networks which are generated automatically by TIM as part of the type inference process. Examination of these networks indicates the possible states that objects of the associated type can inhabit. The *brokenSym* structure, in our current implementation, is replaced by a data structure which records, for each object and for each symmetry state that object can inhabit, whether the symmetry of the object is currently broken in that state. Thus, the symmetry of objects can be restored as they traverse their state-transition networks. The information stored in the new data structure is level-dependent, reflecting the fact that state transitions are made by applications of actions. We expect the added overhead of initiating and maintaining the new data structures to be far outweighed, in problems where symmetry is significant, by the advantages obtained from being able to refresh symmetry as search progresses back from the goal state.

7 Conclusion

In this paper we discuss a way of exploiting the symmetry that is inherent in many planning problems to circumvent much of the search that makes these problems intractable using current planning technology. We have described the algorithm in terms of the modifications made to the basic Graphplan-style search procedure of STAN. Our results demonstrate the significant improvements obtained by detecting and using symmetry during the planning process. STAN with symmetry, STAN version 3, is available for experimental purposes from the STAN web page[2].

References

[Blum and Furst, 1997] A. Blum and M. Furst. Fast planning through planning graph analysis. *AIJ,* 90(1-2):279-298, 1997.

[Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. STRIPS: A New Approach to the Application of Theorem-Proving to Problem-Solving. *Artificial Intelligence,* 2(3), 1971.

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

[Long and Fox, 1999] D. Long and M. Fox. The efficient implementation of the plan-graph in STAN. *To appear in J AIR,* 10,1999.

[Nebel *et al.*, 1997] B. Nebel, Y. Dimopoulos, and J. Koehler. Ignoring irrelevant facts and operators in plan generation. In *ECP-97,* pages 338-350, 1997.

http: //www. *dur*.ac.uk/ =desOwww/research/stanstuff/planpage.html