Exploiting Symmetry in Lifted CSPs

David Joslin
Computational Intelligence Research Laboratory
University of Oregon
Eugene, OR 97403
joslin@cirl.uoregon.edu

Amitabha Roy
Dept. of Computer and Information Science
University of Oregon
Eugene, OR 97403
aroy@cs.uoregon.edu

Abstract

When search problems have large-scale symmetric structure, detecting and exploiting that structure can greatly reduce the size of the search space. Previous work has shown how to find and exploit symmetries in propositional encodings of constraint satisfaction problems (CSPs). Here we consider problems that have more compact "lifted" (quantified) descriptions from which propositional encodings can be generated. We describe an algorithm for finding symmetries in lifted representations of CSPs, and show sufficient conditions under which these symmetries can be mapped to symmetries in the propositional encoding. Using two domains (pigeonhole problems, and a CSP encoding of planning problems), we demonstrate experimentally that the approach of finding symmetries in lifted problem representations is a significant improvement over previous approaches that find symmetries in propositional encodings.

Introduction

As others have shown (Benhamou & Sais 1992; Benhamou 1994; Crawford *et al.* 1996; Brown, Finkelstein, & Purdom 1988), when search problems have large-scale symmetric structure, detecting and exploiting that structure can greatly reduce the size of the search space. Symmetries can arise in a variety of ways. In planning problems, for example, interchangeable resources are one source. In generating a plan for flying a package from one city to another, it makes no sense to waste time trying to decide between two interchangeable airplanes. If the planes do not differ in any significant way, the planner should just pick one. If it turns out that the goal cannot be achieved with the selected plane, there is no need to backtrack and try to generate a plan with the other airplane. The symmetry between the planes guarantees that any solution to the planning problem has a symmetric solution found by simply exchanging the two planes everywhere they are mentioned in the plan. Failing to recognize such symmetries can make the search very inefficient.

Previous work has shown how to find and exploit symmetries in propositional encodings of constraint satisfaction problems (CSPs) (Benhamou & Sais 1992; Benhamou 1994; Crawford *et al.* 1996). Here we consider problems that have "lifted" descriptions. Although most constraint solvers require propositional encodings, it is often natural to describe problems using axioms quantified over the objects in the domain. We might define a map coloring problem, for example, by identifying the countries (A, B and C), the available colors (*red* and *blue*), the variables (for each country c, $colorof(c) \in \{red, blue\}$), and the adjacencies ($adj(A,B)$, $adj(B,C)$); all we then need to add are the following axioms:

$$\forall x \forall y \ \ (adj(x,y) \rightarrow adj(y,x))$$

$$\forall x \forall y \ \ (adj(x,y) \rightarrow colorof(x) \neq colorof(y))$$

It is simple to expand these axioms over the finite domains of objects to generate a purely propositional theory. The advantage of working with lifted representations is that they are typically very compact, compared to the corresponding propositional theories that result from expanding the quantified axioms.

We present an algorithm for finding symmetries in lifted problem descriptions, and demonstrate experimentally, in two domains, that this approach is a significant improvement over finding symmetries in the expanded propositional theory, which in turn is a significant improvement over backtracking search that makes no use of symmetry. One of the domains examined is the pigeonhole problem, for which we compare our results to those of (Crawford *et al.* 1996). We also examine a logistics planning domain, using a CSP encoding of planning problems from (Kautz & Selman 1996). The same approach would be directly applicable to other planners that represent planning problems (or parts of them) as CSPs (Kautz & Selman 1996; Joslin 1996; Joslin & Pollack 1996; Yang 1992).

Detecting and exploiting symmetries

Planning problems, represented as CSPs, are good examples of problems that have natural, lifted descriptions. A number of planners have applied constraint

reasoning to some degree, but some recent approaches have focused on transforming entire planning problems into CSPs (Joslin 1996; Joslin & Pollack 1996; Kautz & Selman 1996; Ginsberg 1996).

A planning problem can be described by the initial and goal states, and the axioms that define the legal states and legal state transitions in a domain. For example, in a state-based encoding for a logistics domain, we might have airplanes *a1* and *a2*, and packages *p1* and *p2*, and have the boolean variable *in(p1,a1,3)* be true if and only if package *p1* is in plane *a1* at time $t = 3$. We would similarly have variables *in(x,y,t)* for all packages x, airplanes y, and time points t. Axioms would describe the legal states and state transitions. One axiom, for example, would be

$$\forall p \forall x \forall y \forall t \; ((at(p,x,t) \wedge at(p,y,t)) \rightarrow x = y)$$

i.e., a plane cannot be in two cities at the same time. Another axiom might require that if a package is moved from one location to another, it must have been on a plane that made the indicated flight. By grounding out the axioms over the finite domains, and adding the literals that define the initial and goal states, we can produce a propositional theory.

To generalize this construction, we define a *lifted constraint satisfaction problem* to be a tuple $\{D, S, L\}$ where D is a finite, colored set of atoms representing the necessary domains, with a distinct "color" assigned to each type of element, S is a set of ground literals, and L is a restricted first-order theory in which quantification is allowed only over the finite domains. (L is thus a compact representation of a propositional theory.)

In a planning problem, D identifies the "objects" relevant to the problem (airplanes, packages, and time points), S defines the initial and goal states (*at(a1,boston,0)* if plane *a1* is in Boston in the initial state), and L contains the axioms that describe the legal states and legal state transitions in the domain. For the map coloring example, D would define the sets of countries and colors, S would give the adjacency relations, and L would contain the axioms mentioned previously. We call S the *problem description* because, in general, the quantified axioms in L hold across all of the problems in a domain (an airplane can only be in one place at a time), while S describes a specific problem (airplane *a1* is in Boston at time $t = 0$).

When L is grounded over the atoms in D, yielding L_D, and combined with S, we get a propositional encoding of a theory, $\mathcal{T}$. Abusing the notation, we will write $\mathcal{T} = S \wedge L_D$.

Prior approaches such as (Crawford *et al.* 1996) find and break symmetries in the propositional theory, $\mathcal{T}$. Because $\mathcal{T}$ can be very large, finding symmetries in $\mathcal{T}$ can be computationally intensive. Our algorithm instead finds symmetries in S, which is much smaller than $\mathcal{T}$, and then uses D to map those symmetries to symmetries in $\mathcal{T}$. We show sufficient conditions for this mapping to be valid.

Symmetry detection in a boolean CSP is polynomial time equivalent to graph isomorphism (Crawford 1992). Though graph isomorphism is a difficult problem, it is not known to be NP complete nor to be in P. Despite the computational difficulty of the problem, there are group theoretic algorithms, such as NAUTY (McKay 1990), that work well in practice. Both the algorithm used in (Crawford *et al.* 1996) and our own algorithm find symmetries by generating a graph from $\mathcal{T}$ or S, respectively, and then invoking NAUTY. The symmetries are then used to generate symmetry-breaking constraints, which are added to the CSP.

We first introduce some notation from group theory. Let G be a group. We write $H \leq G$ to indicate that H is a subgroup of G. We denote by $P(\Omega)$ the symmetric group on the finite set Ω i.e. the set of all $|\Omega|!$ permutations of Ω. Refer to (Wielandt 1964) for definitions and elementary results on permutation groups.

Let $\mathcal{T}$ be a propositional theory. Let V be an ordered set of variables of $\mathcal{T}$. A permutation $\sigma \in P(V)$ is called a symmetry of $\mathcal{T}$ if it produces the same theory after it permutes the clauses and literals of $\mathcal{T}$. A symmetry σ is said to preserve $\mathcal{T}$. $Sym(\mathcal{T}) = \{\sigma | \sigma \in P(V)$ is a symmetry of $\mathcal{T}\} \leq P(V)$ is called the symmetry group of the theory $\mathcal{T}$. For results on symmetries of propositional theories, see (Crawford *et al.* 1996).

Consider a problem, $\mathcal{P} = \{D, S, L\}$, where the full propositional theory is $\mathcal{T} = S \wedge L_D$. Let V be the variables of $\mathcal{T}$ in some order. Note also that V consists of predicates over D: for example, in the logistics domain, a variable in $\mathcal{T}$ might be $at(a1, c1, 4)$, true iff plane $a1$ is at location $c1$ at $t = 4$.

A permutation of atoms in the domain D induces a permutation of variables in $\mathcal{T}$. For example, the permutation $(a1, a2)$ maps plane $a1$ to $a2$ and plane $a2$ to $a1$ in D, and will map the variable $at(a1, c1, 0)$ to $at(a2, c1, 0)$ (and vice versa) in V. It will permute all variables in V which mention planes $a1$ or $a2$ in this manner. It will map variables that don't mention planes $a1$ or $a2$ to themselves. This gives us a permutation in $P(V)$. This procedure gives us a mapping from $P(D)$ to $P(V)$ – we say that we can "extend" a permutation of D to a permutation of V.

We first compute a group of permutations, $H \leq P(D)$, of D whose extension $K \leq P(V)$ preserves S i.e. $K \leq Sym(S)$. The following theorem gives a sufficient condition for K to preserve L_D.

Theorem 1 *If L does not mention any domain elements in D, then* any *permutation of D when extended to a permutation of V will preserve L_D.*

The proof is an easy induction on the quantifier depth of L. An universal quantifier over a "colored set" in the domain (e.g airplanes) mentions the airplanes symmetrically in the conjunction that we obtain on grounding out L. An existential quantifier similarly mentions those elements symmetrically in the disjunction. So in particular, if we have found a set of permutations of D which preserve S when extended, they

also preserve L_D when extended, since any permutation of D preserves L_D. Therefore, they also preserve $\mathcal{T} = S \wedge L_D$ when extended, because if $T = T_1 \wedge T_2$ where T, T_1 and T_2 are propositional theories, then $Sym(T_1) \cap Sym(T_2) \leq Sym(T)$. Thus by looking at the action of $P(D)$ on the much smaller propositional theory S, we can find a group of symmetries of $\mathcal{T}$.

We now present an algorithm that, given a problem $\mathcal{P} = \{D, S, L\}$, computes symmetries of $\mathcal{T}$ by computing the subgroup $H \leq P(D)$ which preserve S, and extending H to a subgroup $K \leq P(V)$. We assume that L does not mention any elements in D. The algorithm constructs a colored graph that captures the symmetries of S. NAUTY then finds the automorphisms of the graph. Those automorphisms are restricted to the domain elements, D, which we then extend to permutations of V.

1. Let A be a graph, initially empty. The nodes of A will be colored.
2. For each element $x \in D$, add a vertex V_x to A. Two such vertices, V_x and V_y share the same color iff they are objects of the same color (same type) in D. For example, if there are three airplanes $a1$, $a2$ and $a3$ in the domain, we have three vertices with the same color in the graph.
3. For each ground literal in S, add a vertex with edges to the domain elements that the literal mentions. For example, if $at(a1, c1, 0)$ is a literal in S, we add a new vertex connected to the vertices representing $a1$, $c1$, and time point 0. A vertex for a literal never has the same color as a vertex for a domain element. Two literal vertices share the same color iff they have the same predicate. For example, $at(a1, c1, 0)$ and $at(a3, p5, 0)$ would get the same color.
4. Give the graph thus generated to NAUTY, which returns the generators for the graph's automorphism group. We restrict the generators to the vertices representing the domain elements, so the result will be in $P(D)$. Extend each generator to a permutation in V. Then $K \leq P(V)$ is generated by the extended generators. Since L does not mention any elements of D, the above theorem implies that $K \leq Sym(L_D)$.

In the discussion above, we have imposed the restriction that L not mention any elements of D. This is a sufficient condition for symmetries in the propositional part, S, extending to symmetries in the ground theory $\mathcal{T}$). In the pigeonhole problems used in the experiments discussed below, this condition is met. In the planning problems, however, time points are mentioned by the presence of a "successor" function, necessary to express that a change at one time point affects the world state at the successor time point. The successor function may be a "built-in" function, but it must be treated as if it were explicitly defined for the purpose of determining which domain elements are mentioned in L.

Let $D' \subseteq D$ be the set of elements in D that are not mentioned in L. We do not need to require that L mention no atoms, but only that it mention no atoms in D. Therefore, if we replace D with D' in the algorithm given above, we meet the sufficient condition that L mention no domain elements in the specified set. Any symmetries we find over D' will extend to symmetries in the fully-ground theory. We currently do not automatically restrict the set of elements in D to exclude those that are mentioned in L, but this could easily be automated.

In the worst case, every element in D is mentioned in L, and our approach fails to find any symmetries. That does not mean that no symmetries exist, and it may even be the case that elements are mentioned in L in a way that preserves symmetry. An alternative approach would be to find permutations of D that extend to symmetries in S, disregarding the mention of any elements of D in L. We can then check whether each permutation preserves L_D; this is equivalent to checking whether a given permutation of vertices of a graph is an automorphism of the graph, and can be computed in time quadratic in the size of L_D. Permutations that are not symmetries would then be discarded.

Example

Suppose we have three cities (*c1*, *c2*, and *c3*), two packages (*p1* and *p2*), and one airplane (*a*). The plane is initially at *c3*, and the packages are at *c1* and *c2*, respectively. The goal is to transfer package *p1* to city *c2*, and package *p2* to city *c1*. Symmetries should tell us, for example, that if there is no solution that starts by having the plane fly first to city *c1* to deliver package *p1*, then there cannot be a solution that instead starts with the plane flying first to city *c2* to deliver package *p2*.

Assuming that the bound on the length of the plan is $t = 3$, the initial and goal states might be described as follows:

$$\begin{array}{ll} at(p1, c1, 0) & at(p1, c2, 3) \\ at(p2, c2, 0) & at(p2, c1, 3) \\ at(a, c3, 0) & \end{array}$$

In addition, we have axioms that define the domain, as discussed previously. The axioms mention time points, so the domains we are interested in are restricted to planes, packages and cities.

The algorithm for finding symmetries in the problem begins by building a colored graph with vertices for each of the domain elements: *a*, *c1*, *c2*, *c3*, *p1*, and *p2*. Elements of the same type share the same color. We also have vertices for each of the ground literals that define the initial and goal states, with edges for each domain element mentioned. For example, the node for *at(p1,c1,0)* has edges to the nodes for *p1* and *c1*.

Finding symmetries in the planning problem has now been transformed into the problem of finding automorphisms in the resulting colored graph. In this

case, exchanging *c1* with *c2*, and *p1* with *p2*, and the corresponding ground literals as well, gives us a graph that is isomorphic to the original. Therefore, we know that we have a symmetry in the original problem that occurs when we exchange packages *and* exchange cities. Obviously this approach will also detect simpler symmetries, such those that would occur if we had some number of interchangeable airplanes.

To exploit this symmetry, we project it onto the ground version of the theory. For example, exchanging domain element *c1* with *c2* in the lifted theory corresponds to exchanging *at(a,c1,1)* and *at(a,c2,1)* in the ground theory. By making all of the appropriate exchanges in this fashion, we identify a symmetry in the ground theory.

We can then generate symmetry-breaking constraints, as described in (Crawford *et al.* 1996). For example, since we have a symmetry that exchanges *at(a,c1,1)* and *at(a,c2,1)*, one of the symmetry-breaking constraints might be:

$$at(a, c1, 1) \rightarrow at(a, c2, 1))$$

This would be added to the theory, with the effect of requiring that if we pick just one of these alternatives, it must not be *at(a,c1,1)*. In other words, if the solver proves that no solution has *at(a,c2,1)* (i.e., we can't solve the problem by going to city *c2* first), then the symmetric alternative of going to city *c1* first is eliminated also, since the new clause prevents us from having *at(a,c1,1)* without also having *at(a,c2,1)*. By adding the new constraints, symmetric choices are forced to be made in one particular way, thus allowing a systematic solver to avoid some redundant effort in the search.

Experimental results

The first domain we examine is the pigeonhole problem, in which N pigeons are to be placed in $N-1$ holes (Benhamou & Sais 1992). The lifted description consists of the domains for pigeons and holes, an axiom requiring that only one pigeon can be assigned to a given hole, and an axiom requiring that every pigeon be assigned to some hole. The boolean variables are *in(p,h)* for all pigeons, p, and all holes, h. Results were presented in (Crawford *et al.* 1996) comparing NTAB (Crawford 1996) (a solver based on Tableau) without using symmetry, to NTAB using symmetry-breaking constraints derived from symmetries found by NAUTY in the propositional theory. In figure 1 we show data for these two approaches, as well as our own results using symmetries discovered in the lifted description. The CPU time for each data point is the time required to prove that the theory is unsatisfiable. (All CPU times reported here are from experiments run on a SPARCstation 10.) As the graph shows, the lifted approach is significantly faster than finding symmetries in the propositional theory, and both easily outperform NTAB without symmetry breaking.

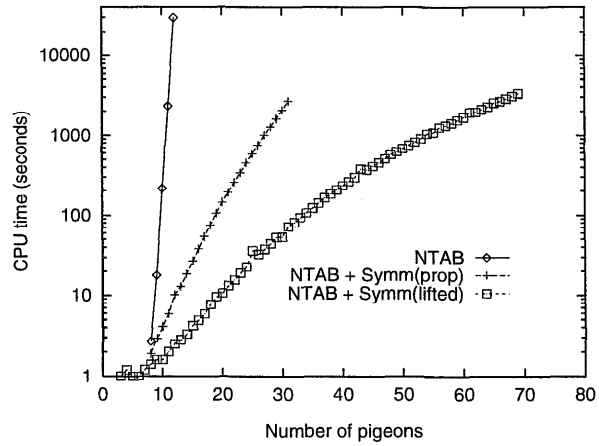

Figure 1: Pigeonhole problems (y-axis is log scaled)

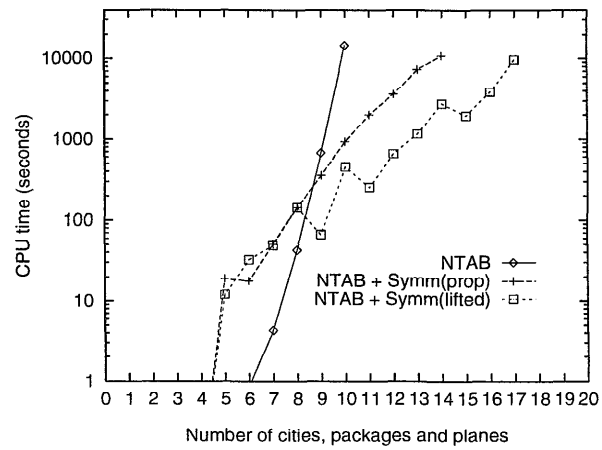

Figure 2: Logistics problems (y-axis is log scaled)

Our second set of experiments uses logistics planning problems based on one of the domains used in (Kautz & Selman 1996). Although Kautz and Selman solve only ground theories, they essentially use a lifted representation to generate those ground theories. For our experiments, we used a reconstruction of that lifted representation (Bedrax-Weiss 1996).

Each problem in our test set has N packages, each in a different city, and $N-1$ planes. Each package must be delivered to a city other than its starting point. The planes all start from a common location that is neither a starting point nor a destination for any of the packages. Because there are not enough airplanes for all of the packages, a valid plan will require at least one plane to make two trips. Problems were generated from $N = 4$ up to $N = 17$.

A polynomial-time preprocessing step compacts the problem by looking for "failed literals," where a failed literal is defined to be one that, when added to the theory, leads to a contradiction by unit resolution. NTAB uses dynamic backtracking (Ginsberg 1993) without

variable re-ordering. In combination, these techniques in the current version of NTAB improve considerably upon the version used by Kautz and Selman in (Kautz & Selman 1996). The logistics problems they describe that were unsolvable by the earlier version of NTAB with a search limit of ten hours are now solved within a few minutes (without using symmetries).

Non-systematic solvers such as WSAT have so far proven to be much more effective at solving satisfiable planning problems than systematic search techniques, with or without symmetry breaking. Non-systematic solvers are of no use, however, in proving that a problem is unsatisfiable. To find an optimal (minimum length) solution, one must find a solution of length t, *and* prove that no solution exists for length $t-1$. For this reason, we anticipate that systematic and non-systematic solvers would be used in parallel, and here we focus on unsatisfiable cases.

Figure 2 shows experimental results with a bound on plan length that allows the planes to make only one trip, making the problems unsatisfiable. The times shown do not include the preprocessing time, which was common to all three of the techniques employed here. Times shown for NTAB with symmetry breaking include the time required to generate the graph, run NAUTY, generate the symmetry breaking constraints, run another compression step on the augmented problem, and finally to run NTAB until the problem is shown to be unsatisfiable. In the propositional case, the graph is generated on the full, expanded theory. In the lifted case, the graph is generated on the propositional part of the lifted theory, as described previously, then the symmetries are mapped to the propositional theory, for which symmetry breaking constraints are generated. As the results show, breaking the symmetries can significantly improve upon the time required to prove that a problem is unsatisfiable.

Our approach of finding symmetries in the compact problem description allows us to find them very quickly; on the largest logistics problem, only about 30 seconds (out of 9650 seconds total) were spent finding symmetries.

Related work

The approach described here finds symmetries in the problem description, and the approach described in (Crawford *et al.* 1996) finds symmetries in the full propositional expansion of the problem, but both break those symmetries by adding new constraints. The augmented problem can then be solved by any standard CSP engine. An alternate approach is to exploit symmetries dynamically, during the search for a solution (Benhamou 1994; Brown, Finkelstein, & Purdom 1988; Lam & Thiel 1989; Lam 1993). In such approaches, the reasoning done about symmetries is very tightly coupled to the search techniques themselves. These approaches would likely benefit from our technique of finding symmetries quickly from the problem description.

One potential advantage to our approach is portability; if a new CSP search engine becomes available, the techniques presented here for detecting and breaking symmetries could be applied immediately, because they only depend on being able to augment the CSP with new constraints. We anticipate, however, that using the symmetry information more directly in the search engine will be far more efficient than our current approach of generating symmetry-breaking constraints. The symmetries discovered by our algorithm can be described very compactly, but on all but the smallest of problems it would be infeasible to generate the constraints needed to break all of those symmetries. Using the compact description of the symmetries directly is a much more promising approach.

Little has been done about automatic recognition of symmetries in planning problems, so both the lifted and non-lifted versions are improvements over previous work. Symmetries that arise from interchangeable resources have typically been handled through knowledge engineering. Some planners, for example, allow interchangeable resources to be assigned to resource pools; the planning algorithm implicitly recognizes that resources within a pool are interchangeable (Tate, Drabble, & Dalton 1994). Another option has been to provide domain-dependent search control information, causing the planner to select and commit to resource allocations in a way that makes sense for the particular domain. This paper shows that it is possible to find symmetries in planning problems automatically, including (but not limited to) those that arise because of interchangeable resources.

Summary and future work

We have shown that it is possible to find and exploit symmetries, such as those that arise from interchangeable resources in planning problems, by analyzing a lifted problem description. Finding these symmetries and adding constraints that break them can be highly effective in reducing the search space.

We hope to look at the possibility of using *approximate* symmetries. Strictly speaking, if two resources differ in any of their properties, they are not symmetric. Not all properties are equally important in all problems, however. For example, two planes that differ slightly in the weight of cargo they can carry may be interchangeable in most problems. Approximate symmetries have also been examined in (Ellman 1993). There, for function-finding problems, an approximate symmetry for goal function G must be an exact symmetry for some other goal function, $\hat{G}$ that is a "good approximation" of G, where "good approximation" is taken to mean either that G entails $\hat{G}$, or vice versa.

Although we have focused on planning and simple constraint satisfaction problems here, the approach we have taken can be easily applied to other types of problems as well. In manufacturing scheduling prob-

lems, for example, identical items in a production run and multiple production lines can introduce symmetries. Identifying approximate symmetries could be very helpful here as well; in a production run of cars on an assembly line, for example, there may be few exact duplicates, but some sets of cars may differ only in relatively minor ways. In searching for near-optimal schedules, when it is impossible to exhaustively examine the entire search space, identifying approximate symmetries could be very helpful in focusing the search on alternatives that are not just minor variations on solutions found so far.

As noted, on satisfiable problems local search typically outperforms systematic search, even when the systematic search is assisted by symmetry breaking. An interesting open question is whether symmetry-breaking constraints would be helpful or harmful (or neither) with local search techniques on satisfiable problems. On the one hand, these new clauses eliminate some solutions, so the solution density decreases. On the other hand, the new clauses may discourage local changes that merely move from one non-solution to a symmetric equivalent. Some preliminary experiments suggest that breaking symmetries hurts local search performance, but more work remains to be done.

We also hope to extend the approach taken here to find symmetries in the non-ground portion of lifted CSPs as well. The approach would be to convert the theory to a canonical form, then build a graph such that symmetries in the lifted theory would correspond to symmetries in the graph. There is no *a priori* reason to believe that this would not work, but the issues are subtle.

We hope to pursue these and other directions of research, and that the work presented here will provide a basis for efficient techniques for detecting and exploiting symmetry in problems that are naturally represented as lifted CSPs.

Acknowledgments: This research has been supported by ARPA/Rome Labs (F30602-95-1-0023), the AFOSR (F49620-96-1-0335), and an NSF CISE Postdoctoral Research award (CDA-9625755).

References

Bedrax-Weiss, T. 1996. personal communication.

Benhamou, B., and Sais, L. 1992. Theoretical study of symmetries in propositional calculus and applications. In Kapur, D., ed., *Automated Deduction: 11th International Conference on Automated Deduction (CADE-11)*, Lecture Notes in Artificial Intelligence, 281–294. Springer-Verlag.

Benhamou, B. 1994. Study of symmetry in constraint satisfaction problems. In *Principles and Practice of Constraint Programming (PPCP-94)*.

Brown, C. A.; Finkelstein, L.; and Purdom, Jr., P. W. 1988. Backtrack searching in the presence of symmetry. In Mora, T., ed., *Applied algebra, algebraic algorithms and error correcting codes, 6th international conference*, 99–110. Springer-Verlag.

Crawford, J.; Ginsberg, M.; Luks, E.; and Roy, A. 1996. Symmetry breaking predicates for search problems. In *Proc. KR-96.*

Crawford, J. 1992. A theoretical analysis of reasoning by symmetry in first-order logic (extended abstract). In *Workshop notes, AAAI-92 workshop on tractable reasoning*, 17–22.

Crawford, J. 1996. Satisfiability techniques for planning problems. Unpublished.

Ellman, T. 1993. Abstraction via approximate symmetry. In *Proc. IJCAI-93.*

Ginsberg, M. 1993. Dynamic backtracking. *Journal of Artificial Intelligence Research* 1:25–46.

Ginsberg, M. L. 1996. A new algorithm for generative planning. In *Proc. KR-96.*

Joslin, D., and Pollack, M. E. 1996. Is "early commitment" in plan generation ever a good idea? In *Proc. AAAI-96.*

Joslin, D. 1996. *Passive and active decision postponement in plan generation.* Ph.D. Dissertation, University of Pittsburgh, Pittsburgh, PA.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI-96.*

Lam, C. W. H., and Thiel, L. 1989. Backtrack search with isomorph rejection and consistency check. *Journal of Symbolic Computation* 7(5):473–486.

Lam, C. W. H. 1993. Applications of group theory to combinatorial searches. In Finkelstein, L., and Kantor, W. M., eds., *Groups and Computation, Workshop on Groups and Computation*, volume 11 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 133–138.

McKay, B. D. 1990. *Nauty* user's guide, version 1.5. Technical Report TR-CS-90-02, Department of Computer Science, Australian National University, Canberra.

Tate, A.; Drabble, B.; and Dalton, J. 1994. Reasoning with constraints within O-Plan2. Technical Report ARPA-RL/O-Plan2/TP/6 Version 1, Artificial Intelligence Applications Institute, University of Edinburgh.

Wielandt, H. 1964. *Finite Permutation Groups.* New York: Academic Press.

Yang, Q. 1992. A theory of conflict resolution in planning. *Artificial Intelligence* 58:361–392.