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 very different domains (pigeonhole problems, and Kautz and Selman’s 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. Moreover, 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. If planes, trucks, and packages are all interchangeable, and we discover that one package cannot be delivered on time with some particular truck and some particular plane, it is pointless to continue to try other combinations of packages, trucks and planes. An exhaustive search that fails to recognize and exploit symmetry will inflate the size of the search space by the number of planes times the number of trucks times the number of packages.

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 with a lifted representation, using axioms quantified over the objects in the domain. We might define a map coloring, for example, by identifying the countries (A, B and C), the available colors (red and blue), the variables (for each country c, colorof(c) ∈ {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). Information about symmetries found in the lifted description of planning problems could be of use to other types of algorithms as well.

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.

### Detecting and exploiting symmetries

Planning problems, represented as CSPs, are good examples of problems that have natural, lifted descriptions. Several techniques for representing planning problems as CSPs have been introduced recently. A number of planners have applied constraints in various ways in the context of a generative planning algorithm, but some more recent approaches have focused on transforming the entire planning problem into a CSP (Kautz & Selman 1996; Joslin 1996; Joslin & Pollack 1996; Yang 1992).

A planning problem can be described by the initial and goal states, and the axioms that define the legal states and state transitions in a domain. For example, in a state-based encoding for a logistics domain, we might have airplanes a1 and a2, and cargoes c1 and c2, and have the boolean variable \( in(c1, a1, 3) \) be true if and only if cargo \( c1 \) is in plane \( a1 \) at time \( t = 3 \). We would similarly have a variable \( in(x, y, t) \) for all packages \( x \), cargoes \( y \), and time points \( t \). Axioms, quantified over the sets of packages, cargoes, and so on, would describe the legal states and state transitions. One axiom, for example, would be

\[
\forall p \forall x \forall y \forall t \ (at(p, x, t) \land at(p, y, t) \Rightarrow x = y)
\]

i.e., a plane cannot be in two cities at the same time. Another axiom might require that if a cargo is moved from one location to another, it must have been on a plane that made the indicated flight. Other axioms would be required as well.

By grounding out the axioms over the appropriate domains, and adding the literals that define the initial and goal states, we can produce a propositional theory, that can then be given to a solver. If a solution is found, a plan can be extracted from it, as discussed in (Kautz & Selman 1996), which takes us from the initial state to the desired goal.

To generalize this construction, we define a lifted constraint satisfaction problem \( P \) 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 possibly-negated ground literals, and \( L \) is a first order theory without function symbols. In a planning problem, \( D \) identifies the “objects” relevant to the domain (airplanes, cargoes, 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 \( L \) are describing general truths that hold across many problems (an airplane can only be in one place at a time), while \( S \) describes aspects of a single 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, \( T \). Abusing the notation, we will write \( T = S \land L_D \).

Prior approaches such as (Crawford et al. 1996) find and break symmetries in the propositional theory, \( T \). Because \( T \) can be very large, finding symmetries in \( T \) can be computational intensive. Our algorithm instead finds symmetries in \( S \), which is much smaller than \( T \), then uses \( D \) to map those symmetries to symmetries in \( T \). 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 hard 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. In fact, both the algorithm used in (Crawford et al. 1996) and our own algorithm find symmetries by generating a graph from \( T \) or \( S \), respectively, and then invoking NAUTY. The symmetries are then used to generate symmetry-breaking constraints, which are added to the CSP. Finally, the augmented problem is given to the solver.

In order to discuss the sufficient conditions for being able to map symmetries in the problem description \( S \) to symmetries in the propositional theory \( T \), we 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 \( Sym(\Omega) \) the symmetric group on the finite set \( \Omega \), i.e., the set of all \( |\Omega| \) permutations of \( \Omega \). For a set of permutations \( X \), by \( (X) \) we will mean group generated by \( X \). Refer to (Wielandt 1964) for elementary results on permutation groups.
Let \( \mathcal{T} \) be a propositional theory. Let \( V \) be the ordered set of variables of \( \mathcal{T} \). A permutation \( \sigma \in \text{Sym}(V) \) 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 \( \sigma \) is also said to preserve \( \mathcal{T} \). Then \( \text{Sym}(\mathcal{T}) = \{ \sigma \mid \sigma \text{ is a symmetry of } \mathcal{T} \} \leq \text{Sym}(V) \) is called group of symmetries 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 \( \mathcal{T} = S \land L_D \). 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 typical variable in \( \mathcal{T} \) is \( at(a_1, p_1, 4) \).

A permutation of atoms in the domain \( D \) induces a permutation of variables in \( \mathcal{T} \). For example, the permutation \( (a_1, a_2) \) (which maps plane \( a_1 \) to \( a_2 \) and plane \( a_2 \) to \( a_1 \) will map the variable \( at(a_1, p_1, 0) \) to \( at(a_2, p_1, 0) \) (and vice versa) in \( \mathcal{T} \). It will permute all variables in \( V \) which mention planes \( a_1 \) or \( a_2 \) in this manner. It will map variables that don’t mention planes \( a_1 \) or \( a_2 \) to themselves. This gives us a permutation in \( \text{Sym}(V) \).

We first find a group of permutations \( H \leq \text{Sym}(D) \) which when lifted preserve just the problem description i.e. \( S \). Let the group of such lifted permutations be denoted by \( K \) where \( K \leq \text{Sym}(S) \). The following claim gives a sufficient condition for \( K \) to preserve \( L_D \).

**Theorem 1** If \( L \) does not mention any particular domain element, then any permutation of \( D \) when lifted will preserve \( L_D \).

The proof is an easy induction on the quantifier depth of \( L \). So in particular, if we have found a set of permutations of \( D \) which preserve \( S \) when lifted, since they also preserve \( L_D \) when lifted (since any permutation of \( D \) preserves \( L_D \)), they preserve \( \mathcal{T} = S \land L_D \). Thus by looking at the action of \( \text{Sym}(D) \) on the much smaller propositional theory \( S \), we can find a group of symmetries of \( \mathcal{T} \).

As a further note, we may observe that \( L \) may mention some domain elements and we can still apply our methods with some modifications. We say that an element \( d \in D \) is distinguished if \( L \) mentions \( d \). Let \( X \subset D \) be all the elements that \( L \) mentions. We can then find permutations in \( \text{Sym}(D - X) \) which preserve \( S \) - the rest of the arguments are similar. Our algorithm presented below takes care of this.

We now present an algorithm that, given a problem \( \mathcal{P} = \{D, S, L\} \), computes a subgroup of symmetries of \( \mathcal{T} \) by computing the subgroup of \( \text{Sym}(D) \) which preserve \( S \) and lifting it to permutations of \( V \). The algorithm constructs a colored graph that captures the symmetries of \( S \). NAUTY then finds the symmetries of the graph. Those symmetries are restricted to the domain elements, \( D \) which we lift to symmetries in \( \mathcal{T} = S \land L_D \).

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 (1) they are objects of the same color (same type) in \( D \), and (2) neither \( x \) nor \( y \) are distinguished elements in \( L \), as defined previously. For example, if there are three airplanes \( a_1, a_2 \) and \( a_3 \) in the domain, none of them distinguished in \( L \), we have three vertices with the same color in the graph. Note that each distinguished element gets a unique color.
3. For each ground literal in \( S \), add a vertex with edges to the domain elements that the clause mentions. For example, if \( at(a_1, p_1, 0) \) is a literal in \( S \), we add a new vertex connected to the vertices representing \( a_1, p_1 \), and time point 0. A vertex for a clause never has the same color as a vertex for a domain element. Two clause vertices share the same color iff they have the same predicate. For example, \( at(a_1, p_1, 0) \) and \( at(a_3, p_5, 0) \) would get the same color.
4. Give the graph thus generated to NAUTY. NAUTY gives us generators for the graph’s automorphism group. We restrict the generators to the vertices representing the domain elements, so the result will be in \( \text{Sym}(D) \). Let the set of restricted generators be \( \mathcal{R} \). Then \( H = \langle \mathcal{R} \rangle \). Lift \( H \) to permutations in \( \text{Sym}(V) \) by lifting each generator \( r \in \mathcal{R} \). Then the lifted subgroup of \( \text{Sym}(V) \) is generated by the lifted generators.

Currently we require the user to specify what the distinguished elements are. However it is easy to automate finding the distinguished elements in \( L \). If no function symbols are allowed in \( L \), then elements not mentioned in \( L \) are not distinguished, and this is sufficient for the pigeonhole case examined in the following section. In the planning example, function symbols are used, but in a manner that allows us to detect that a certain class of elements is distinguished, although they are not mentioned directly. We also hope in the future to be able to more precisely characterize the conditions under which elements are distinguished by \( L \), with fewer restrictions on \( L \) than we currently require. With some modifications, we can extend our method to handle the most general case when \( L \) has no restrictions on its properties. We could check whether a lifted permutation actually preserves \( L_D \) and keep those permutations that are symmetries and throw out those that are not. For complicated first order theories, such checking would indeed be necessary.

**Example**

To illustrate how symmetries are detected in the lifted representation of a problem, and how these symmetries are exploited to prune the search, consider the following example. Suppose we have three cities \((C_1, C_2, \text{ and } C_3)\), two packages \((P_1 \text{ and } P_2)\), and one airplane \((A)\). The plane is initially at \( C_3 \), and the packages are at \( C_1 \) and \( C_2 \), respectively. The goal is to transfer
package \textit{P1} to city \textit{C2}, and package \textit{P2} to city \textit{P1}. The length of the plan will be restricted so that the problem has no solution. We want symmetries here to tell us that if there is no solution that starts by having the plane fly to city \textit{C1} to deliver package \textit{P1} first, then there can't be a solution that instead starts with the plane flying to city \textit{C2} to deliver package \textit{P2} first.

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

\begin{align*}
  & \text{at}(P1, C1, 0) \quad \text{at}(P1, C2, 3) \\
  & \text{at}(P2, C2, 0) \quad \text{at}(P2, C1, 3) \\
  & \text{at}(A, C3, 0)
\end{align*}

In addition, we have axioms that define the results of the various actions (i.e., the changes that occur when a plane flies from one city to another), and other axioms that describe invariants in the domain (i.e., a plane is never in two cities at the same time). The axioms will not need to mention any of the atoms (planes, packages and cities) by name, so they will satisfy the sufficient conditions for preserving any symmetries found in the propositional part of the problem. The fully-ground version of the problem can be generated by applying the axioms to the domain elements. For example, from the axiom:

\[ \forall p \forall x \forall y \forall t \ (\text{at}(p, x, t) \land \text{at}(p, y, t) \rightarrow x = y) \]

(a plane cannot be two cities at the same time), we can write:

\[ \neg \text{at}(A, C1, 1) \lor \neg \text{at}(A, C2, 1) \]

as well as other ground clauses for other cities and time points.

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 \( P2 \). (To keep the discussion simple, we ignore time points here.) 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 \( \text{at}(P1, C1, 0) \) 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 \textit{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 \( \text{at}(A, C1, 1) \) and \( \text{at}(A, C2, 1) \) in the ground theory. By making all of the appropriate exchanges in this fashion, we describe a symmetry in the ground theory.

We can then generate symmetry-breaking predicates, as described in (Crawford et al. 1996). For example, since we have a symmetry that exchanges \( \text{at}(A, C1, 1) \) and \( \text{at}(A, C2, 1) \), one of the symmetry-breaking predicates might be:

\[ \text{at}(A, C1, 1) \rightarrow \text{at}(A, C2, 1) \]

This would be added to the theory as a new clause, with the effect of requiring that if we pick just one of these alternatives, it must not be \( \text{at}(A, C1, 1) \). In other words, if the solver proves that no solution has \( \text{at}(A, C2, 1) \) (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 \( \text{at}(A, C1, 1) \) without also having \( \text{at}(A, C2, 1) \). The actual symmetry-breaking predicates will tend to be more complex than this, but the effect is the same. By adding the new clauses, symmetric choices are forced to be made in one particular way, thus allowing a systematic solver to avoid some redundant effort in the search.

\section*{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 \( \text{in}(p, h) \) for all pigeons, \( p \), and all holes, \( h \). Results were presented in (Crawford et al. 1996) comparing \texttt{ntab}\footnote{The solver we use in all of these experiments is \texttt{ntab} (Crawford 1996), based on Tableau.}
without using symmetry, and NTAB using symmetry-breaking predicates 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.

Our second set of experiments uses logistics planning problems based on one of the domains used in (Kautz & Selman 1996). Each problem 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, an optimal 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, attempting to eliminate the easy parts of the problem. The preprocessing step improves upon the simplification techniques used by (Kautz & Selman 1996). NTAB itself 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.

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 (quicker) 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 preprocessed, propositional theory, for which symmetry breaking predicates are generated. As the results clearly 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. The full propositional theory may have symmetries that do not show up in the problem description, but it is important to note that in both sets of experiments presented here, our approach is able to find symmetries that really matter.

**Related work**

The approaches we have described here can be used to find symmetries in either the problem description or the CSP generated by SATPLAN, DESCARTES, or similar planners. These symmetries are broken by adding constraints to the problem, which 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 predicates. 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 predicates needed to break all of those symmetries. As a result, generating the symmetry breaking predicates increase the size of the representation to such an extent that for practical reasons we end up losing information. Using the compact description of the symmetries directly is a much more promising approach.

As mentioned in the introduction, the most common technique for handling interchangeable resources efficiently in planning is to push the burden onto the knowledge engineer. While providing mechanisms such as resource pools, as provided in O-Plan (Tate, Drabble, & Dalton 1994) and other planners, is helpful, we would prefer to provide the option of having some of this work done automatically.

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. Two planes that differ slightly in the weight of cargo they can carry may be interchangeable in all but the most tightly-constrained problems. We have sketched out several approaches for identifying such approximate symmetries.

A more difficult problem is to recognize symmetries that arise as a result of decisions made by the planner. If two planes start out at different locations, they are unlikely to be interchangeable for any problems, even using the approximation techniques. If, however, at some point in the plan both airplanes end up at the same location, then a symmetry arises that could potentially be exploited. One possibility would be to generate constraints that, when a symmetry arises at some time point in the CSP generated by a planner such as SATPLAN, break that symmetry at successive time points.

We also note that 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 problems, 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.

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 (F30602-93-C-0031 and F30602-95-1-0023), the AFOSR (F49620-92-J-0384), and an NSF CISE Postdoctoral Research award (CDA-9625755). The authors wish to James Crawford for providing us with the most recent version of NTAB, and Tania Bedrax-Weiss for her encoding of Kautz and Selman-style planning problems.

References


