Symmetry-Breaking Predicates for Search Problems

James Crawford
Matthew Ginsberg
Computational Intelligence Research Laboratory
1269 University of Oregon
Eugene, OR 97403-1269
{jc, ginsberg}@cirl.uoregon.edu

Eugene Luks
Amitabha Roy
Department of Computer Science
The University of Oregon
Eugene, OR 97403-1202
{luks, aroy}@cs.uoregon.edu

Abstract

Many reasoning and optimization problems exhibit symmetries. Previous work has shown how special purpose algorithms can make use of these symmetries to simplify reasoning. We present a general scheme whereby symmetries are exploited by adding "symmetry-breaking" predicates to the theory. Our approach can be used on any propositional satisfiability problem, and can be used as a pre-processor to any (systematic or non-systematic) reasoning method. In the general case adding symmetry-breaking axioms appears to be intractable. We discuss methods for generating partial symmetry-breaking predicates, and show that in several specific cases symmetries can be broken either fully are partially using a polynomial number of predicates. These ideas have been implemented and we include experimental results on two classes of constraint-satisfaction problems.

1 Introduction

Human artifacts from chess boards to aircraft exhibit symmetries. From the highly regular patterns of circuitry on a microchip, to the interchangeable pistons in a car engine, or seats in a commercial aircraft, we are drawn esthetically and organizationally to symmetric designs. Part of the appeal of a regular or symmetric design is that it allows us to reason about and understand larger and more complex structures than we could otherwise handle. It follows that if we are to build computer systems that configure, schedule, diagnose, or otherwise reason about human artifacts, we need to endow these reasoning systems with the ability to exploit structure in general and symmetries in particular.

Automated reasoning is a huge area, so for purposes of this paper we will focus on search. Abstractly, a search problem consists of a large (usually exponentially large) collection of possibilities, the *search space*, and a predicate. The task of the search algorithm is to find a point in the search space that satisfies the predicate. Search problems arise naturally in many areas of Artificial Intelligence, Operations Research, and Mathematics.

The use of symmetries in search problems is conceptually simple. If several points in the search-space are related by a symmetry then we never want to visit more than one of them. In order to accomplish this we must solve two problems. First, the symmetries need to be discovered; e.g., we need to realize that we can interchange the five ships without changing the basic form of the problem. Second, we need to make use of the symmetries.

This paper focuses on the second of these problems. It was shown by Crawford [Crawford, 1992] that detecting symmetries is equivalent to the problem of testing graph isomorphism, a problem that has received a substantial amount of study (see, *e.g.*, [Babai, 1995]).

With regard to taking computational advantage of the symmetries, past work has focused on specialized search algorithms that are guaranteed to examine only a single member of each symmetry class [Brown *et al.*, 1988; Crawford, 1992]. Unfortunately, this makes it difficult to combine symmetry exploitation with other work in satisfiability or constraint satisfaction, such as flexible backtracking schemes [Gaschnig, 1979; Ginsberg, 1993] or nonsystematic approaches [Minton *et al.*, 1990; Selman *et al.*, 1992]. Given the rapid progress in search techniques generally over the past few years, tying symmetry exploitation to a specific search algorithm seems premature.

The approach we take here is different. Rather than modifying the search algorithm to use symmetries, we will use symmetries to modify (and hopefully simplify) the problem being solved. In tic-tac-toe, for example, we can require that the first move be in the middle, the upper left hand corner, or the upper middle (since doing this will not change our analysis of the game in any interesting way). In general, our approach will be to add additional constraints, *symmetry-breaking predicates*, that are satisfied by exactly one member of each set of symmetric points in the search space. Since these constraints will be in the same language as the original problem (propositional satisfiability for purposes of this paper) we can run the symmetry detection and utilization algorithm as a preprocessor to any satisfiability checking algorithm.

Of course there is a catch. In this case two catches: First, there is no known polynomial algorithm for detecting the symmetries. Symmetry detection is equivalent to graph isomorphism which is believed to be easier than NP-complete, but is not known to be polynomial. Nevertheless, graph isomorphism is rarely difficult in practice, as has been profoundly demonstrated by the efficient nauty system [McKay, 1990]. Furthermore, it has been shown that, on average, graph isomorphism is in linear time using even naive methods [Babai and Kučera, 1979]. The second catch is that even after detection is complete, computing the full symmetry-breaking predicate appears to be intractable.

However, there is generally no reason to generate the full symmetry breaking predicate. We can generate a partial symmetry-breaking predicate without affecting the soundness or completeness of the subsequent search. We will show that in several interesting cases we can break symmetries either fully or partially using a polynomial number of predicates.

The outline of the rest of this paper is as follows: we first define symmetries of search problems, and discuss how predicates can be added to break symmetries. We then discuss both exact and partial methods for controlling the size of the symmetry-breaking predicate. Finally we discuss experimental results and related work.

2 Definitions and Preliminaries

For purposes of this paper we will assume that we are working in clausal propositional logic. The symmetries of a propositional theory will be defined to be the permutations of the variables in the theory that leave the theory unchanged. These symmetries form a group and we use techniques and notation from computational group theory throughout the paper.

Let L be a set of propositional variables. As usual, literals are variables in L, or negations of variables in L. If $x \in L$, then we write the negation of x as $\overline{x}$. A clause is then just a disjunction of literals (written, *e.g.*, $x \vee y \vee z$), and a theory is a conjunction of clauses. One basic observation that will be critical to the definitions below is that two clauses are considered to be identical iff they involve the same set of literals (*i.e.*, order is not significant) and two theories are identical iff they involve the same set of clauses.

A *truth assignment* for a set of variables L is a function $A : L \rightarrow \{t, f\}$ (on occasion we write $1, 0$ for t, f respectively). In the usual way, A extends by the semantics of propositional logic to a function on the set of theories over L, and, by abuse of notation, we continue to denote the extended function by A.[1] A truth assignment A of L is called a *model* of the theory T if $A(T) = t$. The set of models of T is denoted $\mathcal{M}(T)$.

The propositional satisfiability problem is then just (see, *e.g.*, [Garey and Johnson, 1979]):

Instance: A theory T.

Question: Is $\mathcal{M}(T)$ non-empty (*i.e.*, does T have a model)?

Clearly one can determine whether such an assignment exists by trying all possible assignments. Unfortunately, if the set L is of size n then there are 2^n such assignments. All known approaches to determining propositional satisfiability are computationally equivalent (in the asymptotically worst case) to such a complete search. Propositional satisfiability is thus one of the simplest "canonical" examples of a search problem.

To formally define symmetries we need some additional notation. Consider a set L. The group of all permutations of L is denoted by $\mathrm{Sym}(L)$.[2] This is a group under composition; the product $\theta\phi$ of $\theta, \phi \in \mathrm{Sym}(L)$ is taken to be the result of performing θ and then ϕ. If $v \in L$ and $\theta \in \mathrm{Sym}(L)$, the image of v under θ is denoted v^θ (it is standard to write the permutation as a superscript so that we can make use of the natural equality $v^{\theta\phi} = (v^\theta)^\phi$). A permutation θ

[1]That is, $A(\overline{x})$ is the negation of $A(x)$, A is true of a clause iff it is true of at least one of the terms in the clause, and A is true of a theory iff it is true of all the clauses in the theory.

[2]Recall that a permutation of a finite set L is a one-to-one mapping $\theta : L \rightarrow L$.

of a set L of variables naturally extends to a permutation of negated variables such that $\overline{v}^\theta = \overline{v^\theta}$, and thus to a permutation of the set of clauses over L, wherein if $C = \bigvee_{i=1}^r v_i$, then $C^\theta = \bigvee_{i=1}^r v_i^\theta$, and finally to a permutation of the theories over L, namely, if $T = \{C_i\}_{1\le i\le m}$, then $T^\theta = \{C_i^\theta\}_{1\le i\le m}$.

Let T be a theory over L and let $\theta \in \mathrm{Sym}(L)$. We say that θ is a *symmetry*, or automorphism, of L iff $T^\theta = T$. The set of symmetries of T is a subgroup of $\mathrm{Sym}(L)$ and is denoted by $\mathrm{Aut}(T)$.

For example, consider the following theory: $a\vee\overline{c}$, $b\vee\overline{c}$, $a\vee b\vee c$, $\overline{a}\vee\overline{b}$. Notice that if we interchange a and b the theory is unchanged (again, only the order of the clauses and the order of literals within the clauses is affected). It is customary to denote this particular symmetry by $(a\ b)$.

Permutations of variables in general, and symmetries in particular, can be viewed as acting on assignments as well as theories. If $\theta \in \mathrm{Sym}(L)$ then θ acts on the set of truth assignments by mapping $A \mapsto {}^\theta A$, where ${}^\theta A(v) = A(v^\theta)$ for $v \in L$.[3] Hence, if T is a theory over L, $A(T^\theta) = {}^\theta A(T)$. Thus, we have the immediate consequence that any symmetry of T maps models of T to models of T, and non-models of T to non-models:

Proposition 2.1 *Let T be a theory over L, $\theta \in \mathrm{Aut}(T)$, and A a truth assignment of L. Then $A \in \mathcal{M}(T)$ iff ${}^\theta A \in \mathcal{M}(T)$.*

More generally, $\mathrm{Aut}(T)$ induces an equivalence relation on the set of truth assignments of L, wherein A is equivalent to B if $B = {}^\theta A$ for some $\theta \in \mathrm{Aut}(T)$; thus, the equivalence classes are precisely the *orbits* of $\mathrm{Aut}(T)$ in the set of assignments. Note, further, that any equivalence class either contains only models of T, or contains no models of T. This indicates why symmetries can be used to reduce search: we can determine whether T has a model by visiting each equivalence class rather than visiting each truth assignment.

3 Symmetry-Breaking Predicates

The symmetry-breaking predicates are chosen such that they are true of exactly one element in each of the equivalence classes of assignments generated by the symmetry equivalence. For example, for the small example theory discussed in section 2, the two models are (t, f, f) and (f, t, f). The theory has one non-trivial symmetry – the interchange of a and b. As required by proposition 2.1, applying this perturbation to a model yields a model. We can "break" the symmetry by adding the axiom $a \to b$ which eliminates one of the models, (t, f, f), leaving us with only one model from the equivalence class.

In general, we introduce an ordering on the set of variables, and use it to construct a lexicographic order on the set of assignments. We will then add predicates that are true of only the smallest model, under this ordering, within each equivalence class.[4] Intuitively we do this by viewing each model as a binary number (*e.g.,* (t, f, f) would be seen as 100). We then add predicates saying that θ does not map M to a smaller model (for all symmetries θ).

If $\theta \in \mathrm{Sym}(L)$, then θ acts on any sequence of variables in L: if $V = (v_1, \ldots, v_m)$ then $V^\theta = (v_1^\theta, \ldots, v_m^\theta)$. For a sequence $V = (v_1, \ldots, v_m)$ and $0 \le i \le m$, it is convenient to denote by V_i the initial segment $(v_1, \ldots, v_i)$ (of course, this is the empty sequence () if $i = 0$).

If $v, w \in L$, we write $v \le w$ as a shorthand for the clause $v \to w$. If $V = (v_1, \ldots, v_m)$ and $W = (w_1, \ldots, w_m)$ are sequences of variables in L and $0 \le i \le m$, we let $P_i(V, W)$ abbreviate the predicate

$$V_{i-1} = W_{i-1} \to v_i \le w_i.$$

Finally, we write $V \le W$ as shorthand for

$$\bigwedge_{i=1}^m P_i(V, W),$$

that is,

$$\begin{aligned} v_1 \le w_1 \wedge \\ (v_1 = w_1) \to v_2 \le w_2 \wedge \\ (v_1 = w_1 \wedge v_2 = w_2) \to v_3 \le w_3 \wedge \\ \cdots \end{aligned}$$

The intuition behind this definition is that if we have an assignment A, then the predicate $V \le W$ will be true of A iff $A(V) = (A(v_1), \ldots, A(v_m))$, viewed as a binary number, is less than or equal to $A(W) = (A(w_1), \ldots, A(w_m))$.

Henceforth, we fix an ordering $V = (v_1, \ldots, v_m)$ of the variables in L. Then the set of truth assignments of L inherit a lexicographic ordering, i.e., $A < B$ if, for some i, $A(v_j) = B(v_j)$ for $j < i$, while $A(v_i) < B(v_i)$.

[3] It is natural to write this as a "left action," e.g., we have ${}^{\theta\phi}A = {}^\theta({}^\phi A)$, whereas expressing the image of A under θ by A^θ would lead to the awkward relation $A^{\theta\phi} = (A^\phi)^\theta$.

[4] We note that this is surely not the *only* way to create symmetry-breaking predicates. One can break symmetries by adding any predicate that is true of one member of each equivalence class.

In other words, viewed as binary numbers, $A(V) < B(V)$.

Now consider a symmetry θ of a theory T. The predicate $V \leq V^\theta$ rules out any model M for which $M > {}^\theta M$. It is immediate that

Proposition 3.1 *Let T be a theory, and V be an ordering of it's variables. Then the predicate*

$$\bigwedge_{\theta \in \mathrm{Aut}(T)} V \leq V^\theta$$

is true only of the lexicographically least model in each equivalence class of truth assignments. Hence, it is a symmetry-breaking predicate for T.

Returning to the example we have been tracking, we take $V = (a, b, c)$. Recall that θ swaps a and b. Thus $V^\theta = (b, a, c)$, so $V \leq V^\theta$ is:

$$a \to b$$
$$a = b \to (b \to a)$$
$$(a = b \wedge b = a) \to (c \to c)$$

In which the only non-tautologous term is $a \to b$. This rules out the model (t, f, f). This model is ruled out because θ maps it to the lexicographically smaller model (f, t, f).

By addition of auxiliary variables the predicates given by $V \leq V^\theta$ can be represented by a linear number of clauses. We do this by introducing a new variable e_i defined to be true exactly when $v_i = v_i^\theta$. This can be done by adding the following clauses:

$$(v_i \wedge v_i^\theta) \to e_i$$
$$(\overline{v_i} \wedge \overline{v_i^\theta}) \to e_i$$
$$(e_i \wedge v_i) \to v_i^\theta$$
$$(e_i \wedge \overline{v_i}) \to \overline{v_i^\theta}$$

Nevertheless, since $\mathrm{Aut}(T)$ may be of exponential size, the entire symmetry-breaking predicate given by this theorem may be quite large. In general we have the following negative result:

Theorem 3.2 *The problem of computing, for any theory T, a predicate true of only the lexicographic leader in each equivalence class of models is NP-hard.*

The proof of this theorem is technical and is given in the appendix. The proof includes showing the NP-completeness of the following question: Given an incidence matrix A of a graph Γ, can one reorder the vertices and edges of Γ so as to produce an incidence matrix B that exceeds A lexicographically?

Despite this negative worst-case result, it is still possible to generate, either exactly or approximately, symmetry-breaking predicates for interesting problems. In the next two sections we focus on exact methods and show that in some cases where $\mathrm{Aut}(T)$ is exponential it may still be possible to generate tractable symmetry-breaking predicates. Then, in section 6, we turn to approximate symmetry breaking.

4 The Symmetry Tree

For problems, like n-queens, with a relatively small number of symmetries we are done: one simply computes the symmetries and then calculates the predicate for each symmetry. However, many interesting problems have many symmetries, and computing the predicates for each symmetry yields unnecessary duplication. For example, if $\theta, \phi \in \mathrm{Aut}(T)$ agree on the first i variables then $P_i(V, V^\theta) = P_i(V, V^\phi)$. In order to attack problems with a large number of symmetries, we first organize the symmetries into a *symmetry tree*, and then show how the tree can be "pruned". To describe the symmetry tree and pruning methods we need some notation for permutations.

Again, let T be a theory over L and $V = (v_1, \ldots, v_m)$ be a fixed ordering of L. We can describe a permutation of the variables by listing the image of V under the permutation. For example, the permutation taking v_1 to v_2, v_2 to v_3, and v_3 to v_1 can be written as $[v_2, v_3, v_1]$. The notation is extended to *partial permutations*, which are 1-1 maps of initial segments of V into V, thus the partial permutation taking v_1 to v_3 and v_2 to v_1 is written $[v_3, v_1]$. Note that initial segment could be empty, giving rise to the partial permutation [ ].

For purposes of the formal development to follow, it is useful to describe these partial permutations with a standard group-theoretic construction. Let $G = \mathrm{Aut}(T)$. For $0 \leq i \leq n$, let G_i be the set of permutations in G that do not move the first i variables. That is, $G_i = \{\theta \in G \mid v_j^\theta = v_j, \text{ for } 1 \leq j \leq i\}$. Thus,

$$G = G_0 \supseteq G_1 \supseteq \cdots \supseteq G_n = 1$$

(the last being the identity subgroup). For $0 \leq i \leq n$, let $\mathcal{C}_i$ denote the set of *right cosets* of G_i in G. A right coset $C \in \mathcal{C}_i$ is a set that is of the form $G_i\theta$ for some $\theta \in G$ (note that G is the disjoint union $\bigcup_{C \in \mathcal{C}_i} C$). For purposes of this paper, one can think of a right coset as a partial permutation. For this, let $\theta \in C$, then the partial permutation of length i associated with θ

is $[v_1^\theta, \ldots, v_i^\theta]$. Note that this i-tuple is independent of the choice of $\theta \in C$.

We can now describe the structure of the symmetry tree, $\mathrm{SB}(T)$, for T. The root of $\mathrm{SB}(T)$, considered to be at level 0, is G. The set $\mathcal{C}_i$ comprises the nodes at level i. Furthermore, $C \in \mathcal{C}_i$ is a parent of $C' \in \mathcal{C}_{i+1}$ iff $C' \subseteq C$. Equivalently, in terms of partial permutations, the root is $[\ ]$ and each node $[w_1, \ldots, w_{i-1}]$ will have one child $[w_1, \ldots, w_{i-1}, x]$ for each x that is the image of v_i under a symmetry mapping V_{i-1} to $(w_1, \ldots, w_{i-1})$.

To illustrate this construction, recall the example discussed in section 2. Assume that $V = a, b, c$. There are two symmetries of this theory: the identity operation, and the exchange of a and b. The first of these takes a to itself, and the second takes a to b. There will thus be two children of the root node: $[a]$ and $[b]$. Given that a is mapped to a, b is forced to map to b, so the node $[a]$ has only the child $[a\ b]$. Similarly the node $[b]$ has only the child $[b\ a]$. The final symmetry tree is shown in figure 1.

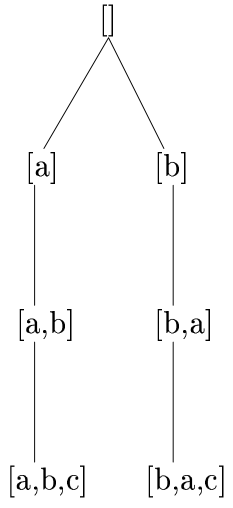

Figure 1: Symmetry trees for simple example.

The duplication previously observed in the symmetry-breaking predicate of Proposition 3.1 arose precisely because $P_i(V, V^\theta) = P_i(V, V^\phi)$ whenever θ and ϕ belong to the same right coset of G_i. With this in mind, for $C \in \mathcal{C}_i$, we define $Q(C, i)$ to be $P_i(V, V^\theta)$ for any (all) $\theta \in C$. (The "i" in the notation "$Q(C,i)$" is not superfluous, that is, it is not determined by C; it is possible that $G_i = G_j$ for $j \neq i$ in which case $C \in \mathcal{C}_j(G)$.) Finally, we associate the predicate $Q(C, i)$ to the corresponding node $C \in \mathcal{C}_i$ in $\mathrm{SB}(T)$. It is now clear that the conjunction of the predicates assigned to the nodes of $\mathrm{SB}(T)$

$$\bigwedge_{i=1}^{n} \bigwedge_{C \in \mathcal{C}_i} Q(C, i)$$

remains a symmetry-breaking predicate for T.

5 Pruning the Symmetry Tree

Working from the symmetry tree to generate symmetry-breaking predicates eliminates a certain amount of duplication. However, there are cases in which the symmetry tree is of exponential size. For example, the theory $(x \vee y \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$ admits all $3! = 6$ permutations of $\{x, y, z\}$. Although, we do not typically expect to see all $n!$ permutations of the variables appearing in practical problems, it is not unusual to see theories with exponentially large symmetry groups. Furthermore, we would surely want to take advantage of the symmetry-breaking opportunities afforded by such a group.

In this section we show that pruning rules can achieve a drastic reduction in size of the symmetry tree in some important cases while still breaking all the symmetries. To see how this is done consider the symmetry tree shown in figure 2. Here, we suppose that $\mathrm{Aut}(T)$ includes the permutation $(x_1\ x_i)$, exchanging x_1 and x_i. Then, for any nodes in the symmetry of T of the form $[x_1, x_j]$ and $[x_i, x_j]$, $j \neq 1, i$, the subtree rooted at $[x_i, x_j]$ (namely, the tree T_2 in the diagram below) can be pruned.

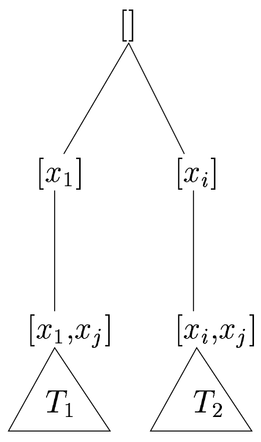

Figure 2: Symmetry trees for pruning example.

To see why this is so, consider any node $[x_i, x_j, \ldots, x_k]$ in T_2. Composing the symmetry creating this branch with $(x_1\ x_i)$, we must find a corresponding node $[x_1, x_j, \ldots, x_k]$ in T_1. In this example, it is not difficult to show that $Q([x_1, x_j, \ldots], k) \to Q([x_i, x_j, \ldots], k)$. Thus we can prune T_2 without loss of inferential power.

The example can be generalized as follows.

Consider a right-coset $C \in \mathcal{C}_i$. Take $\theta \in C$ (the constructions to follow are independent of the choice of θ). Let $\sim_{C,i}$ be the smallest equivalence relation on L such that $v_j \sim_{C,i} v_j^\theta$ for $1 \le j \le i$, i.e., $v \sim_{C,i} w$ if v and w are the endpoints of a sequence $u, u^\theta, u^{\theta\theta}, u^{\theta\theta\theta}, \ldots$ such that, with the possible exception of one of these endpoints, all terms are in V_i. Observe that, if w is in neither of the sequences V_i nor V_i^θ, then the $\sim_{C,i}$ equivalence class of w is a singleton.

From the definition of Q, one can see that if $x \sim_{C,i} y$ then for any child $C', i+1$ of C, i in $SB(T)$, the antecedent of $Q(C', i+1)$ forces $x = y$. From this observation one can show:

Theorem 5.1 *Let $C \in \mathcal{C}_i$. Suppose that $\theta \in G = \mathrm{Aut}(T)$ stabilizes the equivalence classes of $\sim_{C,i}$ (i.e., $v \sim_{C,i} v^\theta$ for all $v \in L$). Then, for all $j > i$ and $C' \in \mathcal{C}_j$ with $C' \subset C$,*

$$Q(C'\theta, j) \to Q(C', j).$$

Hence, all descendents of the level i node C' in $\mathrm{SB}(T)$ may be pruned.

Proof: Let C', j be as indicated. Let $g \in C'$. We must show that the conjunction of the predicates

$$(V^{\phi\theta})_{j-1} = V_{j-1} \quad \to \quad v_j \le v_j^{\phi\theta}, \tag{1}$$
$$(V^{\phi})_{j-1} \quad = \quad V_{j-1} \tag{2}$$

imply $v_j \le v_j^\phi$.

Now, (2) implies $V_i^\phi = V_i$, which implies equality of all the variables in each the equivalence classes of $\sim_{C,i}$. But, since θ stabilizes these classes, this implies $V^\theta = V$ (i.e., the conjunction of all $v^\theta = v$), and, therefore, $V^{\phi\theta} = V^\phi$. Hence, (2) and (3) imply $v_j \le v_j^\phi$ as required. □

In applying the condition globally, we need to be sure that we do not overprune.

It is safe to prune in all instances where $\sim_{C\theta,i} \ne \sim_{C,i}$, in which case, $\sim_{C\theta,i}$ is a proper refinement of $\sim_{C,i}$ (for C, θ, i as in the theorem, $v \sim_{C\theta,i} w$ always implies $v \sim_{C,i} w$).

In the case that $\sim_{C\theta,i} = \sim_{C,i}$, one could prune provided that $C\theta \prec C$ (there is an induced lexicographic ordering on $\mathcal{C}_i$, which is easily seen via the partial-permutation interpretation of cosets). Although some technical extensions to this theorem are possible, this formulation is particularly useful because suitable θ can be found using standard tools of computational group theory. The computation employs "set-stabilizer" techniques that are closely related to graph-isomorphism methods. In particular, methods of Luks [1982] are guaranteed to exhibit suitable θ, if such elements exist, in polynomial time under various conditions, including boundedness of the equivalence classes of $\sim_{C,i}$. As it turns out, in practical computation, this is rarely a difficult problem anyway and is are generally considered to have efficient implementations at least for the cases corresponding to $i \le 10000$ [Butler, 1991].

One case in which pruning is particularly effective is when the symmetry group of the theory is the full symmetric group (that is, any permutation of L is a symmetry of the theory). In this case the symmetry tree is of size $n!$, but after pruning only n^2 nodes remain. To see why this happens, note that since any perturbation is a symmetry, for any $C \in \mathcal{C}_i$ there will always be a $\theta \in \mathrm{Aut}(T)$ that stabilizes the equivalence classes of $\sim_{C,i}$. So for any node C we can always delete *all* it's descendents (so long as we have not already deleted the node $C\theta$!). If one prunes while the tree is being generated, the entire pruned tree (and thus the symmetry-breaking predicate) can be generated in n^2 time. The resultant predicate is not minimal. It turns out that if $V = (v_1, \ldots, v_m)$ then the predicate that one generates consists of a clause $v_i \to v_j$ for any i, j such that $1 \le i < j \le n$. There are obvious polynomial time simplifications that will reduce this to a linear number of clauses, but it is not clear how useful these simplifications will be in the general case.

Remark. Other, less extreme, cases can be constructed in which pruning is effective. However, there are also cases in which the symmetry tree is not prunable to polynomial size. The existence of such cases is a consequence (assuming P≠NP) of theorem 3.2; in fact, the theorem suggests, more strongly, that for some theories, there is no tractable lex-leader predicate since lex-leader *verification* is NP-hard. However, we can also directly construct theories where the symmetry tree cannot be pruned to polynomial size even though the lex-leader problem is in polynomial time (the algorithm uses the "string canonization" procedure of [Babai and Luks, 1983], applicable because the group turns out to be abelian). The existence of the polynomial time algorithm, in turn, guarantees we can find *some* symmetry-breaking predicate in polynomial time even though $\mathrm{SB}(T)$ is useless for this purpose. Details will appear in a future paper.

6 Approximation

If the symmetry tree is of exponential size, and no pruning is possible, then a natural approach is to generate just a part of the tree, and from this smaller tree generate partial symmetry-breaking predicates. We call a predicate P a *partial symmetry-breaking predicate* for a theory T if the models of P consist of at least one member of each of the symmetry equivalence classes of the truth assignments of the variables in T.

We can thus add P without changing the soundness or completeness of the subsequent search. The trade-off here is that the search engine may visit multiple nodes that are equivalent under some symmetry of T. In essence, then, approximating the symmetry-breaking predicate trades time spent generating symmetry-breaking predicates for time in the search engine.

In the next section we discuss various approaches to generating partial symmetry-breaking predicates.

7 Experimental Results

We have implemented a prototype system that takes a propositional theory in clausal form and constructs an approximate symmetry breaking formula from it. The implementation consists of the following steps:

1. The input theory is converted into a graph such that the automorphisms of the graph are exactly the symmetries of the theory. This is done using the construction given in [Crawford, 1992]. There are three "colors" of vertices in this graph, the vertices representing positive literals, those representing negative literals, and those representing clauses. Graph automorphisms are constrained to always map nodes to other nodes of the same color. We also add edges from each literal to each clause that it appears in. These edges (together with the node colorings) guarantee that automorphisms of the graph are symmetries of the theory.[5]

2. We find the generators of the automorphism group of the graph using McKay's graph isomorphism package, nauty [McKay, 1990]. nauty is very fast in practice though there are known examples of infinite classes of graphs which drive nauty to provably exponential behavior [Miyazaki, 1996].

3. From the generators of the automorphism group we construct the symmetry tree and then generate the symmetry-breaking predicate. As expected, in many cases computing the entire symmetry-breaking predicate is computationally infeasible. We use several approximations to compute partial symmetry-breaking predicates:

   - generating predicates for just the generators returned by nauty,
   - building the symmetry tree to some small depth and generating predicates for this smaller tree, and
   - generating random group elements and writing predicates for only those elements.

   An alternative, not yet implemented, would be to use pruning rules such as those from section 5 (though these obviously will not work in all cases).

In the experiments below we generally compare the run time for testing the satisfiability of the input theory alone and conjoined with the symmetry-breaking predicate. In all cases SAT checking was done using the TABLEAU algorithm [Crawford and Auton, 1996] and run times are "user" time. All code is written in C.

7.1 Experiment 1: The pigeonhole problem

The pigeonhole problem $PHP(n, n-1)$ is the following: place n pigeons in $n-1$ holes such that each pigeon is assigned to a hole and each hole holds at most one pigeon. This problem is obviously unsatisfiable. We study this problem because it is provably exponentially hard for any resolution based method, but is tractable using symmetries. A typical encoding of the problem is to have variables $\{P_{ij} | 1 \le i \le n, 1 \le j \le (n-1)\}$ where P_{ij} is taken to mean that pigeon i in hole j. $PHP(n, n-1)$ is then:

$$\begin{aligned}(\forall i \forall j \forall k\ (j \neq k) \Rightarrow (\overline{P_{ij}} \vee \overline{P_{ik}})) \wedge \\ (\forall i \vee_{1 \le j \le (n-1)} P_{ij}) \wedge \\ (\forall j \vee_{1 \le i \le n} P_{ij})\end{aligned}$$

Since all the pigeons are interchangeable and all the holes are interchangeable, the automorphism group of $PHP(n, n-1)$ is the direct product of 2 symmetric groups. The order of this group ($n!(n-1)!$) prohibits the full use of the symmetry tree. Furthermore, as we

[5]For efficiency we special-case binary clauses by representing $x \vee y$ with a link directly from x to y (instead of creating a node for the binary clause and linking x and y to it). This is important because some of the instances we consider have a huge number of binary clauses and some of the algorithms that follow are quadratic, or worse, in the number of nodes.

demonstrate in the Appendix (see final remark), pruning as in section 5 cannot help in this case. Hence, for these experiments, we generate only those predicates that are associated with the generators of the automorphism group (or, more specifically, the set of generators returned by nauty). For PHP, such generators are not only determined in polynomial time, but also serve to break all symmetries. (Of course, in general, the predicates associated with generators of $\mathrm{Aut}(T)$ do not suffice to break all symmetries.)

The run times for various sizes of the n are shown in figure 3. Run times are on a Sparc 10:51

It is difficult to tell from the run-time data what the scaling is, but it turns out that we can show analytically that every step of our implementation is in polynomial time. The input theory can be represented as a graph with $3n^2 - 2n + 2$ vertices[6]. nauty takes this graph as input and finds the generators of its automorphism group. To do this nauty builds a search tree in which each node is a coloring of the vertices which is a suitable refinement of the coloring of the parent node.[7] The time that nauty spends on each node is polynomial in the size of the input graph. So to show that nauty runs in polynomial time it suffices to show that the number of nodes is polynomial. The proof requires a discussion of the details of the internals of nauty that is beyond the scope of this paper, but one can show that for this problem nauty expands exactly $2n^2 - 3n - 1$ nodes. Computing symmetry-breaking predicates for the generators is obviously in polynomial time. The last step is SAT checking which is, in general, exponential, but for these theories, augmented with the symmetry-breaking predicates, there was no need to run TABLEAU: a proof of unsatisfiability was obtained by a polynomial time simplification procedure that is used as a front-end to TABLEAU.

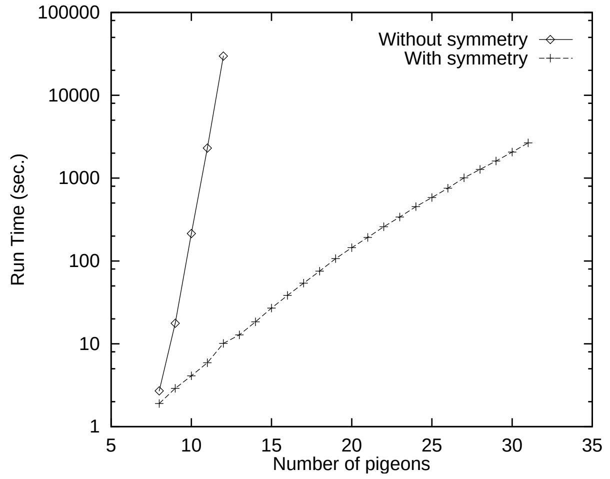

Figure 3: Run times for the pigeon-hole problem with and without symmetry. Note that the y axis is log scaled.

nodes at depth i in the symmetry tree generate clauses of length linear in i. It turns out that long clauses are of very little use to a satisfiability checker like TABLEAU, so for these experiments we cut off the symmetry tree at depth 20 and generate predicates only up to this depth. The results are shown in figure 4. Run times are for a Sparc 5.

7.2 Example 2: N-queens

The n-queens problem has been well studied in the CSP literature, but we include it here as a prototypical example of a problem with a small number of geometric symmetries. The problem is to place n queens on a n by n chess board such that no two queen can attach each other. N-queens has 8 symmetries and for any size board the full symmetry tree has eight leaves.

As one can see from the construction in section 4,

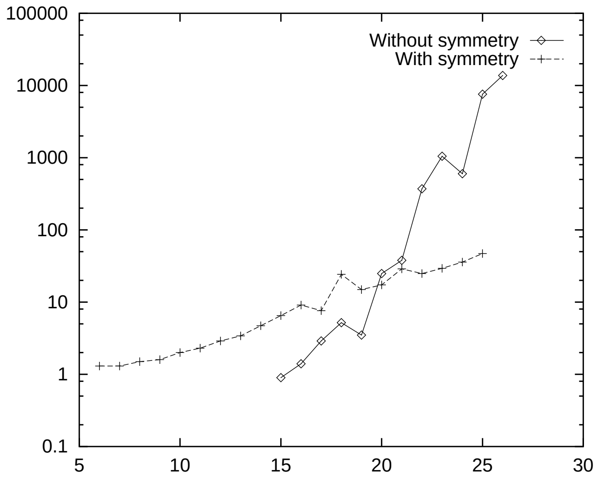

Figure 4: Run times for n-queens with and without symmetry. Note that the y-axis is log scaled.

We know that N-queens is a somewhat delicate problem in that reordering the clauses in the input can drastically change the behavior of SAT checkers (especially as n is increased). Therefore, we took each theory (with and without symmetry-breaking predicates)

[6]The theory actually has $O(n^3)$ clauses, but many of these clauses are binary clauses that become edges in the graph rather than nodes

[7]A refining of a vertex coloring C is another vertex coloring $\hat{C}$ such that if vertex i and j have the same color in $\hat{C}$ then they have the same color in C.

and randomly perturbed the order of the clauses (and of the variables within the clauses) 50 times.[8][9] We then ran TABLEAU on each permuted theory. Figure 5 shows the average run times. As can be seen, the qualitative nature of the results has not changed but a fair amount of noise has been removed.

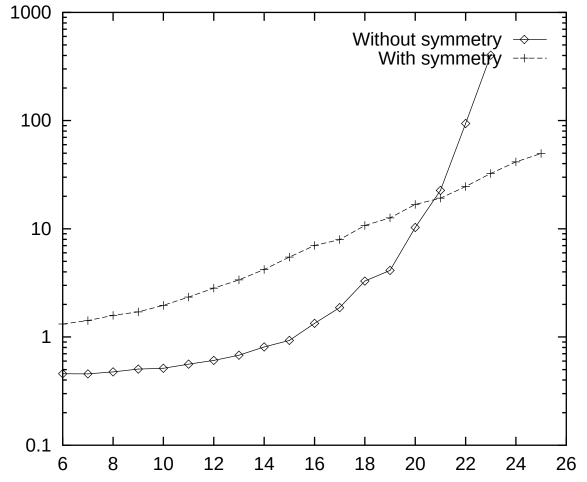

Figure 5: Average run times over 50 random permutations with and without symmetry. Note that the y-axis is log scaled.

8 Related Work

[Freuder, 1991] discusses the elimination of interchangeable values in constraint satisfaction problems. [Brown *et al.*, 1988] discuss an algorithm for backtracking search in the presence of symmetry. In their approach the search engine is modified so that at each node in the search tree a test is done to determine whether the node is lex-least under the symmetries of the theory. [Krishnamurthy, 1985] discusses the idea of using symmetries to reduce the lengths of resolution proofs. He uses the "rule of symmetry" which asserts that if θ is symmetry of a theory Th, and one can show that p follows from Th, then $\theta(p)$ also follows from Th (since the proof could be repeated with each step s replaced by $\theta(s)$). Tour and Demri [1995] show that Krishnamurthy's method is NP-complete in general, but that a restriction of it is equivalent to graph isomorphism. Their restricted method appears to be the same as that discussed by Crawford [Crawford, 1992], and Tour and Demri's proof of graph isomorphism uses essentially the same construction used by Crawford. Benhamour and Sais [1992] also discuss techniques for making use of symmetries within specially designed search engines. The most successful uses of symmetry in reducing search spaces is surely in the large and growing literature on the application of automorphism groups to combinatorial problems (see, *e.g.*, [Butler and Lam, 1985; Lam and Thiel, 1989; Lam, 1993; Laue, 1995]). This work has made impressive contributions to the discovery and classification of designs and to the study of combinatorial optimization problems.

[8]Obviously these perturbations have nothing to do with symmetries of the theory. The idea here is only to average out the cases where the SAT checker gets "lucky" and stumbles on a model almost immediately.

[9]For 23 queens the data given is for 20 perturbations.

9 Conclusion

This work has shown how symmetries can be utilized to add additional constraints, symmetry-breaking predicates, to search problems. These constraints ensure that the search engine never visits two points in the search space that are equivalent under some symmetry of the problem. Complexity results suggest that generating symmetry-breaking predicates will be intractable in the general case. However, partial symmetry breaking can be done in polynomial time (assuming the associated graph isomorphism problem is tractable). Preliminary experiments have been completed showing that partial symmetry breaking is effective on prototypical constraint-satisfaction problems. Work continues on more realistic applications, such as applying these techniques to propositional encodings of planning problems [Joslin and Roy, 1996].

Appendix

Theorem 3.2 is a direct consequence of the NP-completeness of

Problem. MAXIMUM IN MODEL CLASS (MMC)

Input: A theory T over L; an ordering of L; $M \in \mathcal{M}(T)$.

Question: Does there exist $\theta \in \mathrm{Aut}(T)$ such that ${}^{\theta}M < M$?

We shall demonstrate completeness by a reduction from CLIQUE [Garey and Johnson, 1979]. We make use of an intermediate problem, which is interesting in its own right. For lexicographic ordering, we consider $m \times n$ matrices, as mn-tuples, taking the rows in succession.

Problem. MAXIMUM INCIDENCE MATRIX. (MIM)

Input: An incidence matrix A for a graph Γ.

Question: Is there an incidence matrix B for Γ such that $B > A$ with respect to lexicographic ordering.

Recall that a $|V| \times |E|$ incidence matrix of $\Gamma = (V, E)$ is determined by specified orderings of V and E, wherein $A_{ij} = 1$ or 0 according to whether or not the ith vertex lies on the jth edge. Two such matrices are then related by permutations of the rows and columns. Thus, in particular, the NP-completeness of MIM establishes the NP-completeness of the problem of the existence of a matrix B, obtained from a given {0,1}-matrix A by permuting rows and columns, such that $B > A$.

Lemma 9.1 *MIM is NP-complete.*

Proof: MIM is in NP since suitable orderings of V and E can be guessed and verified. For the completeness, we reduce CLIQUE to MIM. Suppose we are given an instance (Γ, K) of CLIQUE, wherein $\Gamma = (V, E)$ is a graph and K is a positive integer; the relevant question is whether Γ contains a K-clique (i.e., a complete subgraph on K vertices). We may assume $|V| > K > 3$.

We augment Γ to a graph $\hat{\Gamma} = (\hat{V}, \hat{E})$ as follows. Fix an ordering $v_1, v_2, \ldots, v_{|V|}$ of V. The additional vertices comprise sets W, X, and Y, disjoint from V and from one another. We describe these along with new edges:

1. W is a complete graph on K vertices, $\{w_1, w_2, \ldots, w_K\}$, ordered as indicated by the subscripts.

2. For each $w_i \in W$, join w_i to each vertex in a new set X_i of size $|V|(|V|+1)/2$; $X = \bigcup_i X_i$ is ordered so that X_i precedes X_j for $i < j$. Only X_1 is joined to V and this is done by joining the first $|V|$ elements of X_1 to v_1, the next $|V| - 1$ elements to v_2, the next $|V| - 2$ to v_3, etc.

3. The set Y, joined only to V, provides a new common neighbor for each pair of vertices V and also ensures the degree of each $v \in V$ is maximal in $\hat{\Gamma}$. Thus, for $1 \leq i < j \leq |V|$, we create a new vertex y_{ij} and add edges joining it to v_i and v_j. Finally, for each $v_i \in V$, join v_i to the vertices in a new set Y_i chosen so as to bring the total degree of v_i to precisely $d = K - 1 + |V|(|V|+1)/2$ (which is also the degree of each $w \in W$). Let $Y = \{ y_{ij} \mid 1 \leq i < j \leq |V| \} \cup \bigcup_i Y_i$. We order Y so that: for $i < j$, y_{ij} precedes Y_i; for $i < i' < j$, Y_i precedes $y_{i'j}$; and for $i < j < j'$, y_{ij} precedes $y_{ij'}$.

The vertices $\hat{V} = W \cup X \cup V \cup Y$ in the resulting graph $\hat{\Gamma}$ are ordered in the sequence $WXVY$ with the orders within each of the four segments as indicated above. By construction, $\hat{\Gamma}$ has a K-clique, W. Observe, however, that $\hat{\Gamma}$ has a *second* K-clique iff Γ had a K-clique, since the vertices in $X \cup Y$ have degree at most 2.

For the instance of MIM, we take A to be the lexicographically greatest incidence matrix of $\hat{\Gamma}$ with respect to the indicated ordering of the vertices. In this regard, note that the maximum incidence matrix for a *given* vertex ordering is obtainable in polynomial time.

We claim that A is the maximum incidence matrix for $\hat{\Gamma}$ if W is first in the ordering of $\hat{V}$. For, suppose A' is the maximum such matrix. Then rows $K+1$ through $K + |X|$ of A' must again correspond to X, the only other vertices directly joined to W; which forces these rows to duplicate their counterparts in A. The next $|V|$ rows in A' must now correspond to V, since these are the only remaining vertices directly joined to Y; suppose these rows correspond to the ordering $v_{i_1}, v_{i_2}, v_{i_3}, \ldots$ of V. Since the v_{i_1} row in A' cannot be exceeded by the v_1 row in A, v_{i_1} must also be joined to $|V|$ elements of X, in fact, to the first $|V|$ elements; so $i_1 = 1$. Similarly, $i_2 = 2$, $i_3 = 3$, etc., so that V remains ordered as before. But, given this ordering of V, the maximality of A' requires Y to be ordered as specified above (except for irrelevant reorderings within each Y_i). Thus, $A' = A$, proving the claim.

We need to show that $\hat{\Gamma}$ has a second K-clique iff $\hat{\Gamma}$ has an incidence matrix $B > A$.

Suppose $\hat{\Gamma}$ has a K-clique $W' \subseteq V$. Reorder $\hat{V}$ so that W' comprises the first K elements and the $(K+1)$st element is the common neighbor in Y of the first two elements of W'. The matrix B induced by the new order agrees with A in the first K rows but its $(K+1)$st row exceeds that of A (the $(K+1)$st row of A begins $0^{K-1}10^{d-2}0$, while the $(K+1)$st row of B begins $0^{K-1}10^{d-2}1$). Hence $B > A$.

Conversely, let B be the maximum incidence matrix for $\hat{\Gamma}$ and suppose that $B > A$. Since the first K rows of A recorded a K-clique consisting entirely of vertices of maximum degree d in $\hat{\Gamma}$, this segment cannot be strictly exceeded. Hence, the first K vertices in the ordering that produces B must form a clique. Since $B > A$, this clique is not W. □

Remark. We trust that the above demonstration will discourage finding-lex-leading-incidence-matrices as an approach to finding canonical forms for graphs and, thereby, to graph isomorphism.

Lemma 9.2 *MMC is NP-complete.*

Proof: MMC is in NP since, one can guess θ, if it exists, and verify both $\theta \in \mathrm{Aut}(T)$ and ${}^{\theta}M < M$ in polynomial time. We reduce MIM to MMC as follows. Suppose the $m \times n$ matrix A constitutes an instance of MIM. Let X and Y be sets of size m, n respectively We describe a theory T on the set $L = X \times Y$ of variables, namely,

$$T = \bigwedge_{\substack{x,x' \in X \\ y \in Y}} \left((x,y) \vee (x',y) \vee \overline{(x',y)}\right) \quad \wedge$$
$$\bigwedge_{\substack{x \in X \\ y,y' \in Y}} \left(\overline{(x,y)} \vee (x,y') \vee \overline{(x,y')}\right)$$

Trivially, T is a tautology. One verifies that $\mathrm{Aut}(T)$ is $\mathrm{Sym}(X) \times \mathrm{Sym}(Y)$ acting on $X \times Y$ in the natural way. We fix orderings $x_1, x_2, \ldots, x_m$ and $y_1, y_2, \ldots, y_n$ of X and Y respectively, and let $X \times Y$ be ordered lexicographically. The $m \times n$ {0,1}-matrix A yields a model M of T wherein $M((x_i, y_j)) = 1 - A_{ij}$ (the $0, 1$ reversal to accommodate a conversion from maximal matrices to minimal models). There is a natural correspondence between row (respectively, column) permutations and $\mathrm{Sym}(X)$ (respectively, $\mathrm{Sym}(Y)$). Thus, if B is obtained from A via a row permutation and a column permutation, then the permutation pair yield $\theta \in \mathrm{Sym}(X) \times \mathrm{Sym}(Y)$. It follows directly that $B > A$ iff ${}^{\theta}M < M$, establishing the reduction of MIM to MMC. □

Remark. This particular proof of the NP-completeness of MMC was chosen so as to show that the problem remains NP-complete even when L can be identified with some $X \times Y$ and $\mathrm{Aut}(T) = \mathrm{Sym}(X) \times \mathrm{Sym}(Y)$. Note that the pigeonhole problem engenders a theory of exactly this type. (the fact that $|X| = |Y| - 1$ in PHP is not significant – routine padding could be used to force this restriction in MIM). Thus, we have demonstrated the futility, for PHP, of pruning via methods, like that of section 5, which rely solely on information contained in $\mathrm{Aut}(T)$ and $\mathrm{SB}(T)$.

Acknowledgements

This work was supported in part by ARPA/Rome Labs under grant numbers F30602-93-C-0031 and F30602-95-1-0023, by AFOSR under grant numbers F49620-92-J-0384 and F49620-96-1-0335, and by NSF under grant number IRI-94 12205. The work benefited from various discussions over the years with many people including Bart Selman, Steve Minton, Takunari Miyazaki, David Etherington, David Joslin, and all the members of CIRL.

References

[Babai and Kučera, 1979] L. Babai and L. Kučera. Canonical labelling of graphs in linear average time. In *Proceedings of the Twentieth IEEE Conference on Foundations of Computer Science*, pages 46–49, 1979.

[Babai and Luks, 1983] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 171–183, Boston, Massachusetts, 25–27 April 1983.

[Babai, 1995] L. Babai. Automorphism groups, isomorphism, reconstruction. In L. Lovász R. L. Graham, M. Grötschel, editor, *Handbook of Combinatorics*, chapter 27, pages 1447–1540. North-Holland – Elsevier, 1995, 1995.

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

[Brown *et al.*, 1988] Cynthia A. Brown, Larry Finkelstein, and Paul W. Purdom. Backtrack searching in the presence of symmetry. In T. Mora, editor, *Applied algebra, algebraic algorithms and error correcting codes, 6th international conference*, pages 99–110. Springer-Verlag, 1988.

[Butler and Lam, 1985] G. Butler and C. W. H. Lam. A general backtrack algorithm for the isomorphism problem of combinatorial objects. *Journal of Symbolic Computation*, 1(4):363–382, 1985.

[Butler, 1991] G. Butler. *Fundamental algorithms for permutation groups.* Lecture notes in computer science. Springer-Verlag, 1991.

[Crawford and Auton, 1996] James Crawford and Larry Auton. Experimental results on the crossover point in random 3sat. *Artificial Intelligence*, 81, 1996.

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

[de la Tour and Demri, 1995] Thierry Boy de la Tour and Stéphane Demri. On the complexity of extending ground resolution with symmetry rules. In *Pro-*

ceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), volume 1, pages 289–295, 1995.

[Freuder, 1991] Eugene G. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 227–233, 1991.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Co., New York, 1979.

[Gaschnig, 1979] John Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, 1979.

[Ginsberg, 1993] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[Joslin and Roy, 1996] David Joslin and Amitabha Roy. Exploiting symmetry in plan generation. Unpublished manuscript, 1996.

[Krishnamurthy, 1985] B. Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22:253–275, 1985.

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

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

[Laue, 1995] R. Laue. Construction of groups and the constructive approach to group actions. In S. Walcerz T. Lulek, W. Florek, editor, *Symmetry and Structural Properties of Condensed Matter in Proceedings of the Third International School on Theoretical Physics*, pages 404–416. World Scientific - Singapore, New Jersey, London, Hong Kong, 1995.

[Luks, 1982] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comp. Sys. Sci.*, 25:42–65, 1982.

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

[Minton *et al.*, 1990] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 17–24, 1990.

[Miyazaki, 1996] Takunari Miyazaki. The complexity of McKay's canonical labeling algorithm. In L. Finkelstein and W. M. Kantor, editors, *Groups and Computation II, Workshop on Groups and Computation*, volume to appear of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 1996.

[Selman *et al.*, 1992] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, 1992.