Implementing a Generalized Version of Resolution

Heidi E. Dixon
CIRL/CIS
1269 University of Oregon
Eugene, OR 97403 USA
dixon@cirl.uoregon.edu

Matthew L. Ginsberg
On Time Systems, Inc.
1850 Millrace, Suite 1
Eugene, OR 97403 USA
ginsberg@otsys.com

David K. Hofer
CIS
University of Oregon
Eugene, OR 97403 USA
dhofer@cs.uoregon.edu

Eugene M. Luks
CIS
University of Oregon
Eugene, OR 97403 USA
luks@cs.uoregon.edu

Andrew J. Parkes
CIRL
1269 University of Oregon
Eugene, OR 97403 USA
parkes@cirl.uoregon.edu

Abstract

We have recently proposed augmenting clauses in a Boolean database with groups of permutations, the augmented clauses then standing for the set of all clauses constructed by acting on the original clause with a permutation in the group. This approach has many attractive theoretical properties, including representational generality and reductions from exponential to polynomial proof length in a variety of settings. In this paper, we discuss the issues that arise in implementing a group-based generalization of resolution, and give preliminary results describing this procedure's effectiveness.

Introduction

We often say that real-world constraint satisfaction problems contain *structure*. The term 'structure' is somewhat vague, but generally means that a problem contains recognizable patterns. Structured problems might best be described as problems in which subproblems or symmetric variants of subproblems recur throughout the search.

Usually, a problem is originally described using some high level language in which problem structure is explicit. A planning problem, for example, might be represented using first order logic. The Boolean satisfiability methods generally used for solving CSPs are variants of the DPLL "Davis-Putnam" algorithm and are based on conjunctive normal form (CNF) encodings. In practice, the high level encoding is used to generate a much larger CNF or *ground* encoding that unfortunately obscures the original problem structure, making it impossible for solvers to exploit this extra information in solving the problem.

A high price is paid for this approach, since many classes of problems cannot be solved efficiently without appealing to the problem structure in some way. As an example, the pigeonhole problem occurs naturally in many planning and scheduling domains, but DPLL is too weak to produce short proofs of unsatisfiability for these instances. Traditional methods using CNF encodings thus suffer from exponential scaling on these problems.

This roadblock for traditional methods has led some researchers to adapt DPLL to use stronger representations (Barth 1995; Chatalic & Simon 2000; Li 2000). When strong representations are combined with strong inference, the structure of the problem can be leveraged to produce shorter proofs. These lifted methods appear able to overcome the drawbacks of weak encodings without sacrificing the performance achieved by CNF-based methods. The disadvantage is that no single representation has proven sufficient to capture the range of structures present in naturally occurring problem instances.

We have shown (Dixon *et al.* 2004b) that permutation groups provide a general and efficient way of representing problem structure. Group-based axiomatizations generalize cardinality constraints, parity constraints and constraints quantified over finite domains. Inference among different constraint types becomes possible within this unified framework, using a general inference rule called *augmented resolution*. Augmented resolution allows multiple resolutions to be performed in parallel, leading to a powerful yet simple proof system that is still practical for automation. As we will see, the computational issues that arise in this setting can be solved by drawing on the large body of theoretical and algorithmic work that already exists for groups.

Groups

The collection of permutations on a set L will be denoted $\mathrm{Sym}(L)$. If the elements of L can be labeled $1, 2, \ldots, n$ in some obvious way, $\mathrm{Sym}(L)$ is often denoted simply S_n. If we take L to be the integers from 1 to n, a particular permutation can be denoted by a series of disjoint cycles, so that the permutation $\omega = (135)(26)$, for example, would map 1 to 3, then 3 to 5, then 5 back to 1. It would also exchange 2 and 6. The order in which the disjoint cycles are written is irrelevant, as is the choice of first element within a particular cycle. If ω_1 and ω_2 are two permutations, it is obviously possible to compose them; we will write the composition as $\omega_1\omega_2$ where the order means that we operate first with ω_1 and then with ω_2.

While composition is associative, it is not necessarily commutative. As an example, $(123)(23) = (12)$ but $(23)(123) = (13)$. The composition operator also has an inverse, since any permutation can obviously be inverted by mapping x to that y with $\omega(y) = x$. In our example, it is easy to see that $\omega^{-1} = (153)(26)$. If we have a set $S \subseteq L$ and a permutation $\omega \in \mathrm{Sym}(L)$, we will denote by $\omega(S)$ the set generated by applying ω to each element of S. We call

this set the *image* of S under ω.

We see, then, that the set S_n is equipped with a binary operation that is associative, and relative to which there is an identity element and each element has an inverse. Thus S_n is a *group*. There are many excellent references for group theory generally (Rotman 1994, and others) and computational group theory specifically (Seress 2003).

Definition 1 *A subset S of a group G is called a* subgroup *of G, denoted $S \leq G$, if S is closed under the group operations of inversion and multiplication.*

Groups can be described without enumerating all of their elements; consider the group S_n, which is of size $n!$. We can represent a group G by giving only a set S of permutations that *generate* G in that any element of G can be expressed as a product of elements of S. If S is such a generating set for G, we will write $G = \langle S \rangle$.

The number of generators required to describe any group $G \leq S_n$ is easily shown to be at most $\log_2 |G|$. The reason is that the size of a subgroup always divides the size of the group and so, if $x \notin \langle S \rangle$, then adding x to S at least doubles the size of $\langle S \rangle$. Thus the number of generators needed can never exceed $\log_2 |G|$; a more sophisticated analysis shows that $\lfloor \frac{n}{2} \rfloor$ also serves as a bound for any subgroup of S_n if $n > 3$ (McIver & Neumann 1987). Permutation representations of groups provide highly compact specifications of large objects.

Axiom Structure as a Group

Cardinality Constraints Cardinality constraints have the form

$$x_1 + \cdots + x_m \geq k \tag{1}$$

asserting that at least k of the x_i's must be true. The single axiom (1) is equivalent to $\binom{m}{k-1}$ conventional disjunctions. The representational strength of cardinality constraints allows polynomial length proofs of the pigeonhole problem (Cook, Coullard, & Turan 1987) which is known to be exponentially difficult for any resolution-based method (Haken 1985). Consider the cardinality constraint

$$x_1 + x_2 + x_3 + x_4 + x_5 \geq 3 \tag{2}$$

which can be encoded in conjunctive normal form as

$$\begin{array}{ll} x_1 \vee x_2 \vee x_3 & x_1 \vee x_4 \vee x_5 \\ x_1 \vee x_2 \vee x_4 & x_2 \vee x_3 \vee x_4 \\ x_1 \vee x_2 \vee x_5 & x_2 \vee x_3 \vee x_5 \\ x_1 \vee x_3 \vee x_4 & x_2 \vee x_4 \vee x_5 \\ x_1 \vee x_3 \vee x_5 & x_3 \vee x_4 \vee x_5 \end{array} \tag{3}$$

The constraint (2) could also be encoded by the first ground axiom $x_1 \vee x_2 \vee x_3$, together with the set of permutations

$$\text{Sym}(\{x_1, x_2, x_3, x_4, x_5\}). \tag{4}$$

The remaining ground axioms (3) can be generated by applying the set of permutations (4) to the first axiom. More generally, a cardinality constraint of form (1) can be written as a single clause $x_1 \vee x_2 \vee \cdots \vee x_{m-k+1}$ together with the group $G = \text{Sym}(\{x_i\})$. Operating on $x_1 \vee x_2 \vee \cdots \vee x_{m-k+1}$ with elements of G allows us to generate the full set of CNF clauses that is logically equivalent to (1).

Parity Constraints We now consider constraints that are most naturally expressed using modular arithmetic or exclusive or's, such as

$$x_1 \oplus \cdots \oplus x_k = 1 \tag{5}$$

Axiom sets consisting of parity constraints in isolation can be solved in polynomial time using Gaussian elimination, but there are examples that are exponentially difficult for resolution-based methods (Tseitin 1970).

As in the cardinality example, single axioms such as (5) reveal structure that a Boolean axiomatization obscures. In this case, (5) with $k = 3$ is equivalent to:

$$\begin{array}{ll} x_1 \vee x_2 \vee x_3 & x_1 \vee \neg x_2 \vee \neg x_3 \\ \neg x_1 \vee x_2 \vee \neg x_3 & \neg x_1 \vee \neg x_2 \vee x_3 \end{array} \tag{6}$$

These four axioms can be generated from the first using the three permutations in the set

$$\{(x_1, \neg x_1)(x_2, \neg x_2), (x_1, \neg x_1)(x_3, \neg x_3), (x_2, \neg x_2)(x_3, \neg x_3)\}$$

Although literals are now being exchanged with their negations, this set, too, is closed under the group inverse and composition operations. Since each element is a composition of disjoint transpositions, each element is its own inverse. The composition of the first two elements is the third. In general, a constraint of the form (5) can be written as the clause $x_1 \vee x_2 \vee \cdots \vee x_k$ together with the group

$$G = \langle (x_1, \neg x_1)(x_2, \neg x_2), \ldots, (x_{k-1}, \neg x_{k-1})(x_k, \neg x_k) \rangle.$$

First Order Structure Consider next a first-order constraint such as

$$P(x, y) \vee Q(y, z) \vee R(x, z) \tag{7}$$

where each variable is universally quantified over a finite domain of size d, so that (7) corresponds to d^3 ground axioms. If x, y and z are all chosen from the two element domain $\{a, b\}$, the single lifted axiom (7) corresponds to the set of ground instances:

$$\begin{array}{c} P(a,a) \vee Q(a,a) \vee R(a,a) \\ P(a,a) \vee Q(a,b) \vee R(a,b) \\ P(a,b) \vee Q(b,a) \vee R(a,a) \\ P(a,b) \vee Q(b,b) \vee R(a,b) \\ P(b,a) \vee Q(a,a) \vee R(b,a) \\ P(b,a) \vee Q(a,b) \vee R(b,b) \\ P(b,b) \vee Q(b,a) \vee R(b,a) \\ P(b,b) \vee Q(b,b) \vee R(b,b) \end{array}$$

If we introduce ground literals l_1, l_2, l_3, l_4 for the instances of $P(x, y)$ and so on, we get:

$$\begin{array}{c} l_1 \vee l_5 \vee l_9 \\ l_1 \vee l_6 \vee l_{10} \\ l_2 \vee l_7 \vee l_9 \\ l_2 \vee l_8 \vee l_{10} \\ l_3 \vee l_5 \vee l_{11} \\ l_3 \vee l_6 \vee l_{12} \\ l_4 \vee l_7 \vee l_{11} \\ l_4 \vee l_8 \vee l_{12} \end{array} \tag{8}$$

at which point the structure implicit in (7) has apparently been obscured.

But note that the set of axioms (8) is "generated" by a set of transformations on the underlying variables. For example, we are allowed to swap the values of a and b for variable x, and still have a sanctioned ground clause. Since the variable x appears as the first argument of both P and R, the permutation

$(P_{(a,a)}, P_{(b,a)})(P_{(a,b)}, P_{(b,b)})(R_{(a,a)}, R_{(b,a)})(R_{(a,b)}, R_{(b,b)})$

corresponds to a swap of a and b as the first argument of both P and R, producing the correct action on the relevant ground literals.

In terms of the literals in (8), this becomes

$$\omega_x = (l_1 l_3)(l_2 l_4)(l_9 l_{11})(l_{10} l_{12})$$

In a similar way, swapping the two values for y corresponds to $\omega_y = (l_1 l_2)(l_3 l_4)(l_5 l_7)(l_6 l_8)$ and z produces $\omega_z = (l_5 l_6)(l_7 l_8)(l_9 l_{10})(l_{11} l_{12})$. Now consider the subgroup $G = \langle \omega_x, \omega_y, \omega_z \rangle$ of Sym$(\{l_i\})$ generated by ω_x, ω_y and ω_z. As in our earlier examples, this group allows all of the clauses in (8) to be generated from any such clause. Thus operating on the first axiom in (8) with ω_x produces $l_3 \vee l_5 \vee l_{11}$. This is the fifth axiom, exactly as it should be, since we have swapped $P(a, a)$ with $P(b, a)$ and $R(a, a)$ with $R(b, a)$. Alternatively, a straightforward calculation shows that

$$\omega_x \omega_y = (l_1 l_4)(l_2 l_3)(l_5 l_7)(l_6 l_8)(l_9 l_{11})(l_{10} l_{12})$$

mapping the first axiom in (8) to the next to last, and so on.

Augmented Clauses Before proceeding, note that any "reasonable" permutation that maps a literal l_1 to another literal l_2 should respect the semantics of the axiomatization and map $\neg l_1$ to $\neg l_2$ as well.

Definition 2 *Given a set of n variables, we will denote by W_n that subgroup of S_{2n} that maps the literal $\neg l_1$ to $\neg l_2$ whenever it maps l_1 to l_2.*

Definition 3 *An* augmented clause *in an n-variable Boolean satisfiability problem is a pair (c, G) where c is a Boolean clause and $G \leq W_n$. A ground clause c' is an* instance *of an augmented clause (c, G) if there is some $g \in G$ such that $c' = g(c)$. Two augmented clauses (c_1, G_1) and (c_2, G_2) will be called* equivalent *if they have identical sets of instances. This will be denoted $(c_1, G_1) \equiv (c_2, G_2)$.*

Proposition 4 *Let (c, G) be an augmented clause. Then if c' is any instance of (c, G), $(c, G) \equiv (c', G)$.* ■

Definition 5 *If C is a set of augmented clauses, we will say that C* entails *an augmented clause (c, G), writing $C \models (c, G)$, if every instance of (c, G) is entailed by the set of instances of the augmented clauses in C.*

Augmented clauses provide highly compact descriptions of structured clause sets. An augmented clause can be exponentially more concise than its equivalent set of ground clauses.

Proposition 6 *Let S be a set of ground clauses over n variables, and (c, G) an equivalent augmented clause. Then a set of generators for G can be expressed in $O(n^2)$ space.* ■

Generalizing Resolution

Of course, presenting a more compact representation is not progress in and of itself; it must also be possible to *reason* with the representation. Augmented clauses possess a natural generalization of the classical Boolean idea of resolution; the essential idea is that the "resolvent" of two augmented clauses (c_1, G_1) and (c_2, G_2) should be, as nearly as possible, the set of all resolvents that can be obtained by resolving an instance of (c_1, G_1) with an instance of (c_2, G_2).

Definition 7 *Let (c, G) be an augmented clause. By $G(c)$ we will mean the union of all instances $g(c)$ of the augmented clause (c, G). For a permutation p and set S with $p(S) = S$, by $p|_S$ we will mean the restriction of p to the given set, and we will say that p is a* pullback *of $p|_S$ back to the original set on which p acts.*

Note that it is not possible to restrict a group to an arbitrary set; one cannot restrict the permutation (xy) to the set $\{x\}$ because you need to add y as well.

Definition 8 *Let (c_1, G_1) and (c_2, G_2) be two augmented clauses. A permutation p is called an* extension of (c_1, G_1) and (c_2, G_2) *if there are $g_i \in G_i$ such that for $i = 1, 2$, $p|_{c_i} = g_i|_{c_i}$. We will denote the set of extensions of (c_1, G_1) and (c_2, G_2) by* $\texttt{extn}(c_i, G_i)$.

An extension will be called stable *if there are $g_i \in G_i$ such that for $i = 1, 2$, $p|_{G_i(c_i)} = g_i|_{G_i(c_i)}$. We will denote the set of stable extensions of (c_1, G_1) and (c_2, G_2) by* $\texttt{stab}(c_i, G_i)$.

Note that the only difference between an extension and a stable extension is the domain for which the permutation p is required to match elements of the groups G_i. For an extension, the match must be only on the given clause c; stability requires that the match be on the entire image of c under the G_i.

If (c_1, G_1) and (c_2, G_2) are augmented clauses such that the Boolean clauses c_1 and c_2 resolve to give resolve(c_1, c_2), the set of all clauses of the form $p($resolve$(c_1, c_2))$ where $p \in \texttt{extn}(c_i, G_i)$ is analogous to the set of all resolvents that can be obtained by resolving an instance of (c_1, G_1) and one of (c_2, G_2). Unfortunately, the set of extensions p is not closed under composition and is not a group; it is only for groups that the representational and other efficiencies of the augmented approach can be realized. But the set of stable extensions *is* a group, leading to:

Definition 9 *Let (c_1, G_1) and (c_2, G_2) be augmented clauses. Then the* resolvent *of (c_1, G_1) and (c_2, G_2), to be denoted by* resolve$((c_1, G_1), (c_2, G_2))$, *is the augmented clause* $(\text{resolve}(c_1, c_2), \texttt{stab}(c_i, G_i))$.

The above definition resolves many of the instances of (c_1, G_1) and of (c_2, G_2) in "parallel", allowing us to conclude at a stroke many of the clauses that would have been obtained had we resolved the individual instances of (c_1, G_1) with the instances of (c_2, G_2).

Proposition 10 *Augmented resolution is sound, in that if $(c, G) =$ resolve$((c_1, G_1), (c_2, G_2))$ and c' is an instance of (c, G), then $(c_1, G_1) \wedge (c_2, G_2) \models c'$.* ■

Proposition 11 *Augmented resolution is complete, in that if* (c_1, G_1) *and* (c_2, G_2) *are augmented clauses with instances* c_1' *and* c_2' *respectively and* $c' = \text{resolve}(c_1', c_2')$*, then* c' *is an instance of* $\text{resolve}((c_1', G_1), (c_2', G_2))$. ■

Recall that $(c_1', G_1) \equiv (c_1, G_1)$ by virtue of Proposition 4, and similarly for c_2'.

In some cases, computing the group of stable extensions is easy:

Lemma 12 *If* (c_1, G) *and* (c_2, G) *are augmented clauses and* $G(c_1) = G(c_2)$*, then* $\text{resolve}((c_1, G), (c_2, G)) \equiv (\text{resolve}(c_1, c_2), G)$. ■

But what can be said about the more general case? We now address this difficulty.

Augmented resolution

The essence of the augmented resolution computation involves computing the group of stable extensions of the groups in the resolvents. Specifically, we have augmented clauses (c_1, G_1) and (c_2, G_2) and need to compute the group G of stable extensions of G_1 and G_2. Recalling Definition 8, this is the group of all permutations ω with the property that there is some $g_1 \in G_1$ such that $\omega|_{c_1^{G_1}} = g_1|_{c_1^{G_1}}$ and similarly for $g_2 \in G_2$ and c_2. We have adjusted notation here, replacing the $G_i(c_i)$ in the original Definition 8 with $c_i^{G_i}$. The reason for the notational shift is that the composition of two group elements fg acts with f first and then with g. By replacing $g(f(x))$ with x^{fg}, the original definition of function composition $g(f(x)) = (fg)(x)$ (note the awkward variable order) becomes the more natural $x^{fg} = (x^f)^g$. As remarked in Definition 7, $c_i^{G_i}$ is the union of the images of c_i under permutations in G_i.

As an example, consider the two clauses

$$\begin{aligned}(c_1, G_1) &= (a \vee b, \langle (ad), (be), (bf), (xy)\rangle)\\ (c_2, G_2) &= (c \vee b, \langle (be), (bg)\rangle)\end{aligned}$$

The image of c_1 under G_1 is $\{a, b, d, e, f\}$ (the x and y appearing in the group are irrelevant), and $c_2^{G_2} = \{b, c, e, g\}$. We therefore need to find those permutations ω such that ω restricted to $\{a, b, d, e, f\}$ is an element of $\langle (ad), (be), (bf), (xy)\rangle$, and ω restricted to $\{b, c, e, g\}$ is an element of $\langle (be), (bg)\rangle$.

From the second condition, we know that c cannot be moved by ω, and any permutation of b, e and g is acceptable because (be) and (bg) generate the symmetric group $\text{Sym}\{b, e, g\}$. This second restriction does not impact the image of a, d or f under ω.

From the first condition, we know that a and d can be swapped or left unchanged, and any permutation of b, e and f is acceptable. But recall from the second condition that we must also permute b, e and g. These conditions combine to imply that we cannot move f or g, since to move either would break the condition on the other. We can swap b and e or not, so the group of stable extensions is $\langle (ad), (be)\rangle$, and that is what our construction should return.

As a preliminary, we need the following:

Definition 13 *Given a group* G *acting on a set* L*, the* pointwise stabilizer *of* L*, denoted* G_L*, is the subgroup of all* $g \in G$ *such that* $l^g = l$ *for every* $l \in L$*. The* set stabilizer *of* L*, denoted* $G_{\{L\}}$*, is that subgroup of all* $g \in G$ *such that* $L^g = L$*.*

Point stabilizers can be computed in polynomial time. There is no known polynomial algorithm for set stabilizers in general (see (Luks 1993) for a discussion of the complexity of this and related problems), although set stabilizer is not NP-hard unless the polynomial hierarchy collapses to Σ_2^p (Babai & Moran 1988).

Procedure 14 *Given augmented clauses* (c_1, G_1) *and* (c_2, G_2)*, to compute* $\texttt{stab}(c_i, G_i)$*:*

1 $\texttt{c\_image}_i \leftarrow c_i^{G_i}$ for $i = 1, 2$
2 $\texttt{g\_restrict}_i \leftarrow G_i|_{\texttt{c\_image}_i}$ for $i = 1, 2$
3 $C_\cap \leftarrow \texttt{c\_image}_1 \cap \texttt{c\_image}_2$
4 $\texttt{g\_stab}_i \leftarrow \texttt{g\_restrict}_{i\{C_\cap\}}$ for $i = 1, 2$
5 $\texttt{g\_int} \leftarrow \texttt{g\_stab}_1|_{C_\cap} \cap \texttt{g\_stab}_2|_{C_\cap}$
6 $\{g_j\} \leftarrow \{\text{generators of } \texttt{g\_int}\}$
7 $\{l_{ij}\} \leftarrow \{g_j, \text{pulled back to } \texttt{g\_stab}_i\}$ for $i = 1, 2$
8 $\{l'_{2j}\} \leftarrow \{l_{2j}|_{\texttt{c\_image}_2 - C_\cap}\}$
9 **return** $\langle \texttt{g\_restrict}_{1C_\cap}, \texttt{g\_restrict}_{2C_\cap}, \{l_{1j} \cdot l'_{2j}\}\rangle$

Proposition 15 *Procedure 14 returns* $\texttt{stab}(c_i, G_i)$.

Space prohibits our both proving this result and providing an example of the procedure in action; we work through our example and remark that the proof follows it closely.

1. $\texttt{c\_image}_i \leftarrow c_i^{G_i}$. As described earlier, we have $\texttt{c\_image}_1 = \{a, b, d, e, f\}$ and $\texttt{c\_image}_2 = \{b, c, e, g\}$.
2. $\texttt{g\_restrict}_i \leftarrow G_i|_{\texttt{c\_image}_i}$. We restrict each group to the corresponding $\texttt{c\_image}_i$. We get $\texttt{g\_restrict}_2 = G_2$ but $\texttt{g\_restrict}_1 = \langle (ad), (be), (bf)\rangle$ as the irrelevant points x and y are removed.
3. $C_\cap \leftarrow \texttt{c\_image}_1 \cap \texttt{c\_image}_2$. The construction considers three separate sets – the intersection of the images of the original clauses (where the computation is interesting because the various g_i must agree), and the points in only the image of c_1 or only the image of c_2. The analysis on these latter sets is straightforward; we just need ω to agree with any element of G_1 or G_2 on the set in question. Here we compute the intersection region $C_\cap = \{b, e\}$.
4. $\texttt{g\_stab}_i \leftarrow \texttt{g\_restrict}_{i\{C_\cap\}}$. We find the subgroup of $\texttt{g\_restrict}_i$ that set stabilizes $C_\cap = \{b, e\}$. For $\texttt{g\_restrict}_1 = \langle (ad), (be), (bf)\rangle$, this is $\langle (ad), (be)\rangle$ because we can no longer swap b and f, while for $\texttt{g\_restrict}_2 = \langle (be), (bg)\rangle$, we get $\texttt{g\_stab}_2 = \langle (be)\rangle$.
5. $\texttt{g\_int} \leftarrow \texttt{g\_stab}_1|_{C_\cap} \cap \texttt{g\_stab}_2|_{C_\cap}$. Since ω must simultaneously agree with both G_1 and G_2 when restricted to $C_\cap$ (and thus with $\texttt{g\_restrict}_1$ and $\texttt{g\_restrict}_2$ as well), the restriction of ω to $C_\cap$ must lie within this intersection. In our example, $\texttt{g\_int} = \langle (be)\rangle$.
6. $\{g_i\} \leftarrow \{\text{generators of } \texttt{g\_int}\}$. Any element of $\texttt{g\_int}$ will lead to an element of the group of stable extensions provided that we extend it appropriately from $C_\cap$ back to the full set $c_1^{G_1} \cup c_2^{G_2}$; this step begins the process of building up these extensions. It suffices to work with just the generators of $\texttt{g\_int}$, and we construct those generators here. We have $\{g_i\} = \{(be)\}$.

7. $\{l_{ki}\} \leftarrow \{g_i, \text{pulled back to } \texttt{g\_stab}_k\}$. Our goal is now to build up a permutation on $\texttt{c\_image}_1 \cup \texttt{c\_image}_2$ that, when restricted to $C_\cap$, matches the generator g_i. We do this by pulling g_i separately back to $\texttt{c\_image}_1$ and to $\texttt{c\_image}_2$. Any such pullback suffices, so we can take (for example) $l_{11} = (be)(ad)$ and $l_{21} = (be)$. In the first case, the inclusion of the swap of a and d is neither precluded nor required; we could just as well have used $l_{11} = (be)$.

8. $\{l'_{2i}\} \leftarrow \{l_{2i}|_{\texttt{c\_image}_2 - C_\cap}\}$. We cannot simply compose l_{11} and l_{21} to get the desired permutation on $\texttt{c\_image}_1 \cup \texttt{c\_image}_2$ because the part of the permutations acting on the intersection $\texttt{c\_image}_1 \cap \texttt{c\_image}_2$ will have acted twice. In this case, we would get $l_{11} \cdot l_{21} = (ad)$ which no longer captures our freedom to exchange b and e.

We deal with this by restricting l_{21} *away* from $C_\cap$ and only then combining with l_{11}. Here, restricting (be) away from $C_\cap = \{b, e\}$ produces the trivial permutation $l'_{21} = ()$.

9. Return $\langle \texttt{g\_restrict}_{1C_\cap}, \texttt{g\_restrict}_{2C_\cap}, \{l_{1i} \cdot l'_{2i}\}\rangle$. We compute the final answer from three sources: The combined $l_{1i} \cdot l'_{2i}$ that we have been working to construct, along with elements of $\texttt{g\_restrict}_1$ and $\texttt{g\_restrict}_2$ that fix every point in $c_\cap$. These latter two sets consist of stable extensions, since an element of $\texttt{g\_restrict}_1$ pointwise stabilizes the image of c_2 if and only if it pointwise stabilizes the points that are in both the image of c_1 (to which $\texttt{g\_restrict}_1$ has been restricted) and the image of c_2; in other words, if and only if it pointwise stabilizes $C_\cap$.

In our example, we have

$$\begin{aligned} \texttt{g\_restrict}_{1C_\cap} &= \langle (ad) \rangle \\ \texttt{g\_restrict}_{2C_\cap} &= 1 \\ \{l_{1i} \cdot l'_{2i}\} &= \{(be)(ad)\} \end{aligned}$$

The group of stable extensions is $\langle (ad), (be)(ad) \rangle$, identical to the "obvious" $\langle (ad), (be) \rangle$. We can swap either the (a, d) pair or the (b, e) pair, as we see fit. The first swap (ad) is sanctioned for the first "resolvent" $(c_1, G_1) = (a \vee b, \langle (ad), (be), (bf) \rangle)$ and does not mention any relevant variable in the second $(c_2, G_2) = (c \vee b, \langle (be), (bg) \rangle)$. The second swap (be) is sanctioned in both cases.

Computational issues We conclude this section by discussing some of the computational issues that arise when we implement Procedure 14, including the complexity of the various operations required.

1. $\texttt{c\_image}_i \leftarrow c_i^{G_i}$. Efficient algorithms exist for computing the image of a set under a group. The basic method is to use a flood-fill like approach, adding and marking the result of acting on the set with a single group element, and recurring until no new points are added.

2. $\texttt{g\_restrict}_i \leftarrow G_i|_{\texttt{c\_image}_i}$. A group can be restricted to a set that it stabilizes by restricting the generating permutations individually.

3. $C_\cap \leftarrow \texttt{c\_image}_1 \cap \texttt{c\_image}_2$. Set intersection is straightforward.

4. $\texttt{g\_stab}_i \leftarrow \texttt{g\_restrict}_{i\{C_\cap\}}$. Although set stabilizer is not known to be in P, implementations exist that are efficient in practice.

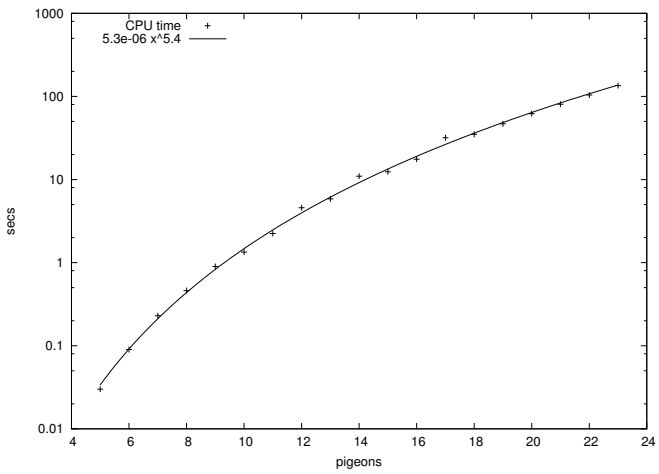

Figure 1: CPU time for a pigeonhole resolution

5. $\texttt{g\_int} \leftarrow \texttt{g\_stab}_1|_{C_\cap} \cap \texttt{g\_stab}_2|_{C_\cap}$. Group intersection is also not known to be in polynomial time (and is in fact polynomial time equivalent to set stabilizer (Luks 1993)); once again, practical and efficient implementations exist.

6. $\{g_i\} \leftarrow \{\text{generators of } \texttt{g\_int}\}$. Groups are typically represented in terms of their generators, so reconstructing a list of those generators is trivial.

7. $\{l_{ki}\} \leftarrow \{g_i, \text{pulled back to } \texttt{g\_stab}_k\}$. Suppose that we have a group G acting on a set S, a subset $T \subseteq S$ and a permutation h acting on T such that we know that h is the restriction to T of some $g \in G$. Finding such a pullback is polynomial in the number of variables acted on by G.

8. $\{l'_{2i}\} \leftarrow \{l_{2i}|_{\texttt{c\_image}_2 - C_\cap}\}$. Restriction is still easy.

9. Return $\langle \texttt{g\_restrict}_{1C_\cap}, \texttt{g\_restrict}_{2C_\cap}, \{l_{1i} \cdot l'_{2i}\}\rangle$. Since groups are represented by their generators, we need simply take the union of the generators for the three arguments. The pointwise stabilizers needed for the first two arguments can be computed in polynomial time.

Experimental results

We have implemented the procedure described in the last section as one of the necessary first steps to building a theorem prover for an augmented resolution system.

The results for the pigeonhole problem are shown in Figure 1. This particular example involves resolving the two basic axioms in a pigeonhole problem containing n pigeons and $n - 1$ holes. (The axioms state that every pigeon is in some hole and no hole contains two pigeons.) We produced the results in the figure by encoding the axioms in a way that obscured the fact that the groups were identical and by also disabling the check to see if the groups were the same.

The solid line gives the observed time (in seconds) for the resolution as a function of the number of pigeons involved, plotted on a log scale. The experiments were performed on a 1.7GHz Pentium-M with 1GB of main memory, although the implementation used only 5MB. Interestingly, the CPU usage was *not* dominated by the non-polynomial steps in Lines 4 and 5 of Procedure 14, but instead by the need to

compute "stabilizer chains" in support of the algorithm generally. Stabilizer chains (Sims 1970) are data structures that support a variety of computational manipulations on groups, and can be constructed in time $O(n^5)$ if n is the number of variables being manipulated by the group in question (Knuth 1991). The solid curve in the figure is the best fit between the data and a CPU utilization of ax^b, which occurs for $b = 5.4$.

The pigeonhole problem was chosen for our experiments because, although the groups involved are very simple (the direct product of a symmetry over n pigeons and one over $n-1$ holes), the standard stabilizer chain construction generally works poorly on full symmetry groups or products thereof. For this reason, we expect that the computational needs of Procedure 14 in a pigeonhole setting are in fact greater than those that will be encountered in other augmented problems. The running time can be reduced to $O(n^3)$ by using stabilizer chain algorithms designed to work well on full symmetry groups (and only slightly worse than the standard algorithms in the general case). The running time can be reduced to $O(1)$ by recognizing that all of the groups involved are identical and applying Lemma 12 to produce the resolvent group directly.

We have shown elsewhere (Dixon *et al.* 2004b) that actually *solving* the pigeonhole problem in this framework involves a total of $O(n^2)$ resolutions and a similar number of unit propagations. The unit propagations require a single stabilizer chain computation costing $O(n^3)$ (Dixon *et al.* 2004a) but potentially shared among all of the propagations required, since the group is unchanged throughout the problem.[1] The overall scaling on this problem instance can therefore be expected to be $O(n^3)$, which compares favorably with the $O(n^5)$ scaling obtained by exploiting the symmetry directly using symmetry-breaking axioms (Crawford *et al.* 1996), or with the exponential scaling needed by conventional methods. The augmented approach is also far more flexible, since it can deal with theories where no global symmetry is present, or with structure in other forms.

Conclusion

Augmented resolution is a completely new mechanism for solving satisfiability problems, and has extremely attractive theoretical properties including exponential reductions in both the space needed to store a clausal database and the number of inference steps needed to derive conclusions from it. In this paper, we have begun to investigate the computational properties of this new approach, presenting an algorithm for computing augmented resolution and describing the results of experiments that appear likely to measure that algorithm's worst case performance. The result of those experiments indicate that the time needed for a single resolution remains polynomial in the number n of domain elements being considered, although the time does grow as $O(n^5)$ if full symmetry groups are involved. In practice, it should be possible to reduce the resolution time substantially by exploiting the specific structure of the groups in question.

[1] As with resolution, there are non-polynomial elements to the unit propagation procedure (Dixon *et al.* 2004a) but the evidence is that the stabilizer chain construction dominates.

References

Babai, L., and Moran, S. 1988. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. System Sci.* 36:254–276.

Barth, P. 1995. A Davis-Putnam based enumeration algorithm for linear pseudo-boolean optimization. Technical Report MPI-I-95-2-003, Max Planck Institut für Informatik, Saarbrücken, Germany.

Chatalic, P., and Simon, L. 2000. Zres: the old Davis-Putnam meets ZBDDs. In McAllester, D., ed., *17th International Confernece on Automated Deduction (CADE'17)*, number 1831 in Lecture Notes in Artificial Intelligence (LNAI), 449–454.

Cook, W.; Coullard, C.; and Turan, G. 1987. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics* 18:25–38.

Crawford, J. M.; Ginsberg, M. L.; Luks, E.; and Roy, A. 1996. Symmetry breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*.

Dixon, H. E.; Ginsberg, M. L.; Hofer, D.; Luks, E. M.; and Parkes, A. J. 2004a. Generalizing Boolean satisfiability III: Implementation. Technical report, CIRL, Eugene, Oregon.

Dixon, H. E.; Ginsberg, M. L.; Luks, E. M.; and Parkes, A. J. 2004b. Generalizing Boolean satisfiability II: Theory. Technical report, CIRL, Eugene, Oregon.

Haken, A. 1985. The intractability of resolution. *Theoretical Computer Science* 39:297–308.

Knuth, D. E. 1991. Notes on efficient representation of permutation groups. *Combinatorica* 11:57–68.

Li, C. M. 2000. Integrating equivalency reasoning into Davis-Putnam procedure. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 291–296.

Luks, E. M. 1993. *Permutation Groups and Polynomial-Time Computation*, volume 11 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. Amer. Math. Soc. 139–175.

McIver, A., and Neumann, P. 1987. Enumerating finite groups. *Quart. J. Math.* 38(2):473–488.

Rotman, J. J. 1994. *An Introduction to the Theory of Groups*. Springer.

Seress, A. 2003. *Permutation Group Algorithms*, volume 152 of *Cambridge Tracts in Mathematics*. Cambridge, UK: Cambridge University Press.

Sims, C. C. 1970. Computational methods in the study of permutation groups. In Leech, J., ed., *Computational Problems in Abstract Algebra, Proc. Conf. Oxford, 1967*. Pergamon Press.

Tseitin, G. 1970. On the complexity of derivation in propositional calculus. In Slisenko, A., ed., *Studies in Constructive Mathematics and Mathematical Logic, Part 2*. Consultants Bureau. 466–483.