Efficient Symmetry Breaking for Boolean Satisfiability

Fadi A. Aloul **Karem A. Sakallah** **Igor L. Markov**

Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122

{faloul, karem, imarkov}@umich.edu

Abstract

Identifying and breaking the symmetries of CNF formulae has been shown to lead to significant reductions in search times. In this paper we describe a more systematic and efficient construction of symmetry-breaking predicates (SBPs). In particular, we use the cycle structure of symmetry generators, which typically involve very few variables, to drastically reduce the size of SBPs. Furthermore, our new SBP construction grows linearly with the number of relevant variables as opposed to the previous quadratic constructions. Our empirical data suggest that these improvements reduce search run times by one to two orders of magnitude on a wide variety of benchmarks with symmetries.

1 Introduction

Modern Boolean satisfiability (SAT) solvers, based on backtrack search, are now capable of attacking instances with thousands of variables and millions of clauses [Velev and Bryant, 2001] and are being routinely deployed in a wide range of industrial applications [Biere *et al*., 1999], [Nam *et al*., 2001], [Stephan *et al*., 1996]. Their success can be credited to a combinational of recent algorithmic advances and carefully-tuned implementations [Silva and Sakallah, 1999], [Moskewicz *et al*., 2001], [Goldberg and Novikov, 2002]. Still, there are problem instances that remain beyond the reach of most SAT solvers.

One aspect of intractability is the presence of symmetry in the conjunctive normal form (CNF) of a SAT instance. Intuitively, a symmetry of a discrete object is a transformation, e.g., a permutation, of its components that leaves the object intact. The symmetries of a CNF formula are permutations of its literals (variables and their negations) that result in a re-ordering of its clauses (and the literals within clauses) without changing the formula itself. Such symmetries induce an equivalence relation on the set of variable assignments such that two assignments are equivalent if and only if the formula assumes the same truth value (either 0 or 1) at each of these assignments. A search algorithm that is oblivious to the existence of these symmetries may end up, wastefully exploring a set of equivalent unsatisfying assignments before moving on to a more promising region of the search space. On the other hand, knowledge of the symmetries can be used to significantly prune the search space. Symmetries are studied in abstract algebra in terms of *groups*. We assume the reader to be familiar with the basics of group theory; in particular, we assume familiarity with permutation groups and their representation in terms of irredundant sets of generators. A good reference on the subject is [Fraleigh, 2000].

The rest of the paper is organized in five sections. Section 2 provides a brief review of permutations and permutation groups. Section 3 describes pervious work on symmetry breaking for SAT. Our main contribution on efficient constructions of symmetry-breaking predicates is detailed in Section 4. These constructions are evaluated empirically in Section 5, and we end with conclusions in Section 6.

2 Notation and Preliminaries

We will be concerned with permutations on the literals of a set of n Boolean variables $\{x_1, \ldots, x_n\}$ which we assume to be totally ordered according to $x_1 \prec x_2 \prec \cdots \prec x_n$. We use I_n to denote the set of integers between 1 and n inclusive, and denote non-empty subsets of I_n by upper-case "index variables" I and J as appropriate. Given an index set I and an index $i \in I$, we define the "index selector" functions:

$$\begin{array}{ll} \text{pred}(i, I) = \{j \in I \mid j < i\} & \text{prev}(i, I) = \max(\text{pred}(i, I)) \\ \text{succ}(i, I) = \{j \in I \mid j > i\} & \text{next}(i, I) = \min(\text{succ}(i, I)) \end{array} \quad (1)$$

where min and max return, respectively, the least and greatest element in the given index set. For completeness, we also let $\min(\varnothing) = n + 1$, and $\max(\varnothing) = 0$.

A permutation π of the set of $2n$ literals $L = \{x_1, x'_1, \ldots, x_n, x'_n\}$ (where x'_i denotes the logical negation of x_i) is a function $\pi: L \to L$ that is both one-to-one and onto. We will denote that x_j is the image of x_i under π by writing $x_j = x_i^\pi$. To preserve Boolean consistency, whenever π maps x_i to x_j it must simultaneously map x'_i to x'_j. Such implied mappings will be assumed whenever not explicitly specified. A permutation π is a *phase-shift* permutation if $x_i^\pi = x'_i$ for some $i \in I_n$, i.e., π maps some literal to its complement.

Permutations will be expressed either in tabular form or in cyclic notation. For example,

$$\pi = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ x_1^\pi & x_2^\pi & \cdots & x_n^\pi \end{pmatrix} \tag{2}$$

denotes a permutation that maps x_1 to x_1^π etc. The same permutation can be expressed as a set of disjoint cycles such as

$$\pi = (x_i, x_i^\pi, (x_i^\pi)^\pi, \ldots)(x_j, x_j^\pi, (x_j^\pi)^\pi, \ldots)\ldots \tag{3}$$

Here a cycle $(a\ ,\ b\ ,\ \ldots\ ,\ z)$ is a shortcut for "*a* maps to *b*, *b* maps to *c*, . . . and *z* maps to *a*." The length of a cycle is equal to the number of literals in it; we will refer to a cycle whose length is *k* as a *k*-cycle. We define the support of a permutation π, $\mathrm{supp}(\pi)$, to be the set of indices appearing in its cyclic representation, i.e.,

$$\mathrm{supp}(\pi) = \{ i \in I_n | x_i^\pi \neq x_i \} \tag{4}$$

The number of cycles in a permutation π will be denoted by $\mathrm{cycles}(\pi)$. We also define $\mathrm{phase\text{-}shift}(\pi)$ to be the index of the *smallest* variable (according to the assumed total ordering) that is mapped to its complement by π:

$$\mathrm{phase\text{-}shift}(\pi) = \min \{ i \in I_n | x_i^\pi = x_i' \} \tag{5}$$

We should note that a phase-shift permutation must have one or more *phase-shift cycles,* i.e., length-2 cycles that have the form $(x_i\ ,\ x_i')$. Finally, we define $\mathrm{ends}(\pi)$ as follows:

$$\mathrm{ends}(\pi) = \{ i \in I_n | i \text{ is largest index of a variable in a non-phase-shift cycle of } \pi \} \tag{6}$$

A *permutation group G* is a group whose elements are permutations of some finite set and whose binary operation is function composition, also referred to as permutation multiplication. The order of a group is the number of its elements. A *subgroup H* of a group *G*, denoted $H \leq G$ is a subset of *G* that is closed under the group's binary operation. The *cyclic subgroup* of $\pi \in G$, denoted by $<\pi>$, is the subgroup consisting of π and its integer powers:

$$<\pi> = \{ \pi^i | i \in \mathbb{Z} \} \tag{7}$$

and π is said to generate $<\pi>$. A set of permutations $\pi_1 \in G\ ,\ \ldots\ ,\ \pi_k \in G$ *generates G* if the subgroup resulting from taking all possible products of the integer powers of these permutations is equal to *G.* The permutations $\{\pi_1\ ,\ \ldots\ ,\ \pi_k\}$ are called *generators* of *G.* A set of generators is *irredundant* if it is not possible to express any of its permutations as a product of powers of its other permutations. A set of irredundant generators serves as an implicit representation of the group it generates and, in general, guarantees exponential compression in the size of the representation. Note that a set of irredundant generators is not a group since it is not closed under multiplication and taking inverse. In the sequel, a set of permutations *G* that is not necessarily closed will be indicated by placing a "hat" on the variable denoting the set, i.e., $\hat{G}$. Additionally, and with a slight abuse of notation, we will indicate that *G* is the group generated by $\hat{G}$ by writing $G = <\hat{G}>$.

3 Previous Work

The basic framework for utilizing the symmetries in a CNF instance to prune the search space explored by a SAT solver was laid out in [Crawford *et al*., 1996]. This framework was extended later, in [Aloul *et al*., 2002], to account for phase-shift symmetries, take advantage of the cycle structure of permutations, and consider only generators of the group of symmetries. In outline, the procedure consists of the following steps:

1. Convert a CNF formula φ to a colored graph whose symmetries are isomorphic to the symmetries of the formula.
2. Find the symmetries of the graph in terms of a set of irredundant generators $\hat{G} = \{\pi_1, \cdots, \pi_k\}$ using a suitable graph automorphism program [McKay, 1981], [Spitznagel, 1994].
3. Map the graph symmetries back to symmetries of the formula.
4. Construct an appropriate *symmetry-breaking predicate* (SBP) ρ and conjoint it to the formula.
5. Solve $\varphi \wedge \rho$ using a suitable SAT solver [Moskewicz *et al*., 2001].

Our concern in this paper is step 4. Noting that the group of symmetries induces an equivalence relation on the set of assignments in the *n*-dimensional Boolean space, the basic idea is to construct a "filter" that picks out a single representative from each equivalence class. In particular, choosing the *lexicographically smallest representative*—according to the assumed total ordering on the variables—leads to the following *Lex-Leader* SBP [Crawford *et al*., 1996]:

$$\rho^{\mathrm{LL}}(<\hat{G}>) = \bigwedge_{\pi \in <\hat{G}>} \mathrm{PP}(\pi) \tag{8}$$

$$\mathrm{PP}(\pi) = \bigwedge_{i \in I} \mathrm{BP}(\pi, i) \tag{9}$$

$$\mathrm{BP}(\pi, i) = \left[\bigwedge_{j \in \mathrm{pred}(i,I)} \left(x_j = x_j^\pi \right) \right] \rightarrow \left(x_i \leq x_i^\pi \right) \tag{10}$$

where the index set *I* in (9) and (10) is equal to I_n. In these equations, the Lex-Leader SBP is expressed as a conjunction of *permutation predicates* (PPs) each of which is a conjunction of *bit predicates* (BPs)[1]. Introducing *n* auxiliary "equality" variables $e_i \equiv (x_i = x_i^\pi)$ makes it possible to express the *ith* BP as an $(i + 1)$-literal CNF clause. This leads to a CNF representation of the PP in (9) that has *n* clauses with a total literal count of $0.5(n^2 + 3n)$. Additionally, each of the introduced equality constraints yields 4 3-literal clauses bringing the total CNF size of (9) to:

$$\begin{aligned} \mathrm{clauses}(\mathrm{PP}(\pi)) &= 5n \\ \mathrm{literals}(\mathrm{PP}(\pi)) &= 0.5(n^2 + 27n) \end{aligned} \tag{11}$$

In its present form, the lex-leader SBP in (8)-(10) can lead to an exponentially large CNF formula because the order of the symmetry group can be exponential in the number of variables. Thus, its value in pruning the search space is negated by the need of the SAT solver to process a much larger CNF formula. To remedy this problem [Crawford *et al*., 1996] suggested the construction of a symmetry tree to eliminate some redundant permutations. However, in the worst case the number of symmetries in the tree remains exponential. Empirical evidence in [Aloul *et al*., 2002]

[1] Note that $x \leq y$ in the bit predicate means "*x* implies *y*".

$$\pi = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\ x_4 & x_2 & x_8 & x_1 & x_5' & x_3' & x_7 & x_6' & x_9 & x_{10} \end{pmatrix}$$
$$= (x_1\,, x_4)(x_3\,, x_8\,, x_6')(x_5\,, x_5')$$

(a) Permutation in tabular and cyclic notation

$$\begin{aligned}
&\mathrm{supp}(\pi) = \{1\,,3\,,4\,,5\,,6\,,8\}\\
&\text{phase-shift}(\pi) = 5\\
&\mathrm{succ}(\text{phase-shift}(\pi)\,, I_{10}) = \{6\,,7\,,8\,,9\,,10\}\\
&\mathrm{ends}(\pi) = \{4\,,8\}\\
&\mathrm{supp}(\pi) \setminus \mathrm{ends}(\pi) = \{1\,,3\,,5\,,6\}\\
&\mathrm{supp}(\pi) \setminus \mathrm{ends}(\pi) \setminus \mathrm{succ}(\text{phase-shift}(\pi)\,, I_{10}) = \{1\,,3\,,5\}
\end{aligned}$$

(b) Various index sets associated with permutation

$$\begin{aligned}
&\mathrm{BP}(\pi\,,1) = x_1 \le x_4\\
&\boxed{\mathrm{BP}(\pi\,,2) = (x_1 = x_4) \to (x_2 \le x_2)}\\
&\mathrm{BP}(\pi\,,3) = (x_1 = x_4)(x_2 = x_2) \to (x_3 \le x_8)\\
&\boxed{\mathrm{BP}(\pi\,,4) = (x_1 = x_4)(x_2 = x_2)(x_3 = x_8) \to (x_4 \le x_1)}\\
&\mathrm{BP}(\pi\,,5) = (x_1 = x_4)(x_2 = x_2)(x_3 = x_8)(x_4 = x_1) \to (x_5 \le x_5')\\
&\mathrm{BP}(\pi\,,6) = (x_1 = x_4)(x_2 = x_2)(x_3 = x_8)(x_4 = x_1)(x_5 = x_5') \to (x_6 \le x_3')\\
&\boxed{\mathrm{BP}(\pi\,,7) = (x_1 = x_4)(x_2 = x_2)(x_3 = x_8)(x_4 = x_1)(x_5 = x_5')(x_6 = x_3') \to (x_7 \le x_7)}\\
&\boxed{\mathrm{BP}(\pi\,,8) = (x_1 = x_4)(x_2 = x_2)(x_3 = x_8)(x_4 = x_1)(x_5 = x_5')(x_6 = x_3')(x_7 = x_7) \to (x_8 \le x_6')}\\
&\boxed{\mathrm{BP}(\pi\,,9) = (x_1 = x_4)(x_2 = x_2)(x_3 = x_8)(x_4 = x_1)(x_5 = x_5')(x_6 = x_3')(x_7 = x_7)(x_8 = x_6') \to (x_9 \le x_9)}\\
&\boxed{\mathrm{BP}(\pi\,,10) = (x_1 = x_4)(x_2 = x_2)(x_3 = x_8)(x_4 = x_1)(x_5 = x_5')(x_6 = x_3')(x_7 = x_7)(x_8 = x_6')(x_9 = x_9) \to (x_{10} \le x_{10})}
\end{aligned}$$

(c) Bit predicates according to (10). BPs enclosed in boxes with square corners are tautologous because π maps the corresponding bits to themselves. BPs enclosed in boxes with rounded corners are tautologous because they correspond to cycle "ends." The BPs for bits 6 to 10 are tautologous because π maps bit 5 to its complement.

$$\begin{aligned}
\mathrm{PP}(\pi) &= (p_1)(p_1 \to l_1 p_3)(p_3 \to g_1 \to l_3 p_5)(p_5 \to g_3 \to l_5)\\
&= (p_1)(p_1 \to (x_1 \le x_4)p_3)(p_3 \to (x_1 \ge x_4) \to (x_3 \le x_8)p_5)(p_5 \to (x_3 \ge x_8) \to x_5')
\end{aligned}$$

(d) Linear formulation of the permutation predicate according to (18), based only on irredundant bits

Figure 1: Illustration of different formulations of the permutation predicate.

showed that full symmetry breaking, i.e., insuring that the SBP selects only the lex-leader from each equivalence class, is not necessary to obtain significant pruning of the search space. An SBP that breaks some, but not necessarily all, the symmetries of the formula can, in fact, provide a much better space/time trade-off during the search. This is accomplished by replacing the group of symmetries in (8) by a suitable, and much smaller, set of permutations $\hat{H} \subseteq <\hat{G}>$:

$$\rho^{\mathrm{LL}}(\hat{H}) = \bigwedge_{\pi \in \hat{H}} \mathrm{PP}(\pi) \tag{12}$$

In particular, [Aloul *et al.*, 2002] advocated the use of the set of generators $\hat{G}$ returned by the graph automorphism program in step 2.

4 Efficient Formulation of Permutation Predicate

Even when only a small number of permutations is used in constructing an SBP as in (12), the corresponding CNF formula may still be too large because each PP requires a CNF formula whose size is quadratic in the number of variables *n*. In this section we introduce two refinements that lead to much smaller PPs. The first refinement utilizes the *cycle structure* of a permutation to eliminate redundant bit predicates and can be viewed as replacing *n* in (11) by a much smaller number *m* and represents a more comprehensive and systematic treatment of cycles than that in [Aloul *et al.*, 2002]. The second refinement takes advantage of the *recursive bit-by-bit structure* in (10) to yield a CNF formula whose size is linear, rather than quadratic, in *m*. Figure 1 provides an example illustrating these refinements.

Elimination of redundant BPs. Careful analysis of (10) reveals three cases in which a BP is tautologous, and hence redundant. The first corresponds to bits that are mapped to themselves by the permutation, i.e., $x_i^\pi = x_i$. This makes the consequent of the implication in (10), and hence the whole bit predicate, unconditionally true. Removal of such BPs is easily accomplished by setting the index set *I* in (9) and (10) to $\mathrm{supp}(\pi)$ rather than I_n. For sparse permutations, i.e., permutations for which $|\mathrm{supp}(\pi)| \ll n$, this change alone can account for most of the reduction in the CNF size of the PP.

The second case corresponds to the BP of the last bit in each cycle of π. "Last" here refers to the assumed total ordering on the variables. Assume a cycle involving the variables $\{x_j | j \in J\}$ for some index set *J* and let $i = \max(J)$. Then

$$\left[\bigwedge_{j \in J \setminus \{i\}} (x_j = x_j^\pi)\right] \to (x_i \le x_i^\pi) = 1 \tag{13}$$

causing the corresponding bit predicate $\mathrm{BP}(\pi\,, i)$ to be tautologous. Elimination of these BPs is accomplished by restricting the index set *I* in (9) and (10) further to just $\mathrm{supp}(\pi) \setminus \mathrm{ends}(\pi)$ and corresponds to a reduction in the number of BPs from *n* to $m \equiv |\mathrm{supp}(\pi)| - \mathrm{cycles}(\pi)$.

The third and last case corresponds to the BPs of those bits that occur *after* the first "phase-shifted variable." Let *i* be the index of the first variable for which $x_i^\pi = x_i'$. Thus,

$e_i = 0$ and all BPs for $j > i$ have the form $0 \rightarrow (x_j \leq x_j^\pi)$ making them unconditionally true.

Taken together, the redundant BPs corresponding to these three cases can be easily eliminated by setting the index set in (9) and (10) to:

$$I = \text{supp}(\pi) \setminus \text{ends}(\pi) \setminus \text{succ}(\text{phase-shift}(\pi), I_n) \quad (14)$$

In the sequel we will refer to the bits in the above index set as "irredundant bits." Note that the presence of a phase-shifted variable early in the total order can lead to a drastic reduction in the number of irredundant bits. For example, if $\pi = (x_1, x_1')\ldots$ then $\text{PP}(\pi)$ is simply (x_1') regardless of how many other variables are moved by π.

Linear construction of PPs through chaining. The PP in (9) and (10) has a recursive structure that can be utilized to produce a CNF formula whose size is linear, rather than quadratic, in the cardinality of the index set *I*. Specifically, we introduce the "ordering" predicates $l_i = (x_i \leq x_i^\pi)$ and $g_i = (x_i \geq x_i^\pi)$ and, after algebraic manipulation, write the following equivalent expressions for the permutation predicate:

$$\begin{aligned} \text{PP}(\pi) &= g_0 \rightarrow \bigwedge_{i \in I} \left\{ \left(\bigwedge_{j \in \text{pred}(i,I)} g_j \right) \rightarrow l_i \right\} \\ &= g_0 \rightarrow l_k \wedge \left[g_k \rightarrow \bigwedge_{i \in K} \left\{ \left(\bigwedge_{j \in \text{pred}(i,K)} g_j \right) \rightarrow l_i \right\} \right] \end{aligned} \quad (15)$$

where $g_0 \equiv 1$, $k = \text{next}(0, I)$ and $K = \text{succ}(k, I)$. Noting that, except for the index set used, the parenthesized expression on the second line of the above equation is identical to the expression on the first line, we introduce a sequence of *chaining predicates* $\{p_i | i \in I\}$ defined according to:

$$p_i = g_{\text{prev}(i,I)} \rightarrow \bigwedge_{k \in K} \left[\bigwedge_{j \in \text{pred}(k,K)} g_j \right] \rightarrow l_k \quad (16)$$

where $K = \{i\} \cup \text{succ}(i, I) = \{k \in I | k \geq i\}$. The recursive structure of (15), now, makes it possible to express each chaining predicate in terms of the one that follows it

$$p_i = g_{\text{prev}(i, I)} \rightarrow l_i \, p_{\text{next}(i, I)} \qquad i \in I, p_{n+1} \equiv 1 \quad (17)$$

and yields the following alternative representation of the permutation predicate:

$$\text{PP}(\pi) = p_{\min(I)} \wedge \bigwedge_{i \in I} \left[p_i = g_{\text{prev}(i, I)} \rightarrow l_i \, p_{\text{next}(i, I)} \right]$$

which can be simplified further by replacing the equalities by one-way implications leading, finally, to:

$$\text{PP}(\pi) = p_{\min(I)} \wedge \bigwedge_{i \in I} \left[p_i \rightarrow g_{\text{prev}(i, I)} \rightarrow l_i \, p_{\text{next}(i, I)} \right] \quad (18)$$

The CNF representation of each conjunct in (18) is obtained by substituting the definitions of the *l* and *g* variables and using the distributive law. Thus, using this construction the permutation predicate requires $|I|$ additional variables (the chaining predicates) and consists of $2|I|$ 3-literal and $2|I|$ 4-literal clauses for a total of $14|I|$ literals.

5 Experimental Evaluation

We conducted a number of experiments to evaluate the effectiveness of the symmetry breaking constructions described above in reducing search times. We ran the experiments on representative CNF instances from the following seven benchmark families:

1. **Hole-n**: Unsatisfiable pigeon-hole instances [DIMACS]
2. **Urq**: Unsatisfiable randomized instances based on expander graphs [Urquhart, 1987]
3. **GRoute**: Difficult satisfiable instances that model global wire routing in integrated circuits [Aloul *et al.*, 2002]
4. **FPGARoute** and **ChnlRoute**: Large satisfiable and unsatisfiable, instances that model the routing of wires in the channels of field-programmable integrated circuits [Nam *et al.*, 2001]
5. **XOR**: Various exclusive-or chains [SAT 2002]
6. **2pipe**: Difficult unsatisfiable instances that model the functional correctness requirements of modern out-of-order microprocessor CPUs [Velev and Bryant, 2001]

Each of the benchmarks was converted to a colored graph as described in [Aloul *et al.*, 2002] and processed by the graph automorphism program Nauty (version 2) [McKay, 1981] using the GAP package (version 4, release 3) [Spitznagel, 1994]. The symmetries returned by Nauty were then mapped back to symmetries of the benchmark and appropriate SBPs constructed and added. The mChaff SAT solver [Moskewicz *et al.*, 2001] was then run on the original and SBP-augmented versions of each benchmark. All experiments were run on a Linux workstation with a 1.2Ghz AMD Athlon processor and 1GB of RAM. A time-out limit of 1000 seconds was set for all runs, and run time results represent averages over 10 to 200 independent runs.

Table 1 lists, for each benchmark family, the number of instances tested (col. 2), their total CNF sizes (cols. 3 and 4), the order of their symmetry groups (col. 5), the total number of generators returned by Nauty (col. 6), and the number of those that include phase shifts (col. 7). Columns 8 and 9 list the cardinality of the generators' support and the degree of sparsity present in these generators. The remaining columns in the table list the CNF sizes of three SBP constructions based on generators:

- The quadratic construction (using extra equality variables) based on all bits; this represents the previous state-of-the-art
- The quadratic construction based only on irredundant bits
- The linear construction (using extra chanining variables) based only on irredundant bits

Several observations can be made about the data in Table 1. The number of symmetries in these benchmarks is large, but all symmetries, including phase shifts in benchmark families Urq, XOR and 2pipe, can be represented by fairly small sets of generators. The generators returned by Nauty appear very sparse on average, i.e., a typical generator affects only a small number of variables.[2] This explains the reduction, by 1-2 orders-of-magnitude, in the size of symmetry-breaking predicates in column 13 (our first construction) versus column 11 ([Crawford *et al.*, 1996]). Both the number of variables and the number of literals are reduced. While our construction in column 13 only slightly extends the qua-

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bench-mark Family	**# Instances**	**Total Instance Sizes**		**Symmetries**					**Total SBP Sizes**					
				Total	**Generators**				**All bits**		**Irredundant bits**			
									Quadratic Const.				**Linear Const.**	
		Vars	**Lits**		**Tot**	**PS**	**Avg |supp|**	**S**	**Extra Vars**	**Lits**	**Extra Vars**	**Lits**	**Extra Vars**	**Lits**
Hole-n	6	616	6818	3E+18	108	0	20.6	79%	11788	865018	1112	20903	1112	14380
Urq	4	162	7212	6E+08	94	94	1.0	98%	3876	133047	0	94	0	94
GRoute	5	4704	93262	2E+11	138	0	46.6	95%	130032	63307944	3216	87312	3216	43506
FPGARoute	9	1580	19612	1E+21	241	0	24.2	86%	43429	4657249	2922	66603	2922	38257
ChnlRoute	6	1710	24366	2E+52	270	0	29.5	89%	79782	13705131	3987	138079	3987	52848
XOR	4	316	2512	7E+10	108	107	1.0	99%	9240	544092	1	121	1	110
2pipe	3	2620	61566	168	15	10	12.3	82%	13154	5954599	1031	186708	1031	14329
Total	**37**	**11708**	**215348**	**2E+52**	**974**	**211**	**24749**	**90%**	**291301**	**89167080**	**12269**	**499820**	**12269**	**47751**

KEY: Vars: # variables; Lits: # literals; Extra Vars: # additional variables used in SBPs; Tot: total # generators;
PS: # phase-shift generators; Avg. |supp|: average support of generators; S: sparsity of generators (1-Avg |supp|/Vars)

Table 1: Symmetry statistics for various benchmark families and size comparisons of three SBP constructions based on group generators.

1	2	3	4	5	6	7	8	9	10	11	12
Bench-mark Family	**# Instances**	**# Generators**	**# Generators & their compositions**	**Time to find symmetries (sec)**	**Time to solve original instances (sec)**	**Time to solve instances and SBPs (sec)**				**Speedup**	
						Generators only			**Generators & their comp.**	**$\frac{\text{col 6}}{\text{col 5 + col 9}}$**	**$\frac{\text{col 7}}{\text{col 5 + col 9}}$**
						All bits	**Irredundant bits**				
						Quadratic Construction		**Linear Construction**			
Hole-n	6	108	1157	0.38	1246.81	18.75	0.06	0.07	0.65	8832	171
Urq	4	94	94	0.76	895.65	3.72	1.17	1.17	1.17	2084	11
GRoute	5	138	2037	38.76	132.30	960.10	15.55	5.08	38.32	15	99
FPGARoute	9	241	3595	3.24	5070.30	4832.70	0.31	0.21	4.08	13391	12835
ChnlRoute	6	270	6578	25.86	2755.00	2675.85	0.37	0.17	5.87	588	569
XOR	4	108	108	11.43	1900.16	20.52	2.41	2.41	2.41	531	5
2pipe	3	15	18	23.50	7.84	16.88	3.19	8.01	8.31	0.8	2
Total	**37**	**974**	**13587**	**103.93**	**12008.06**	**8528.52**	**23.06**	**17.12**	**60.81**	**3635**	**1956**

Table 2: Comparison of search run times for various choices of SBP constructions and symmetries to break.

dratic-size construction in [Aloul *et al.*, 2002], our more advanced linear-size construction (column 15) offers an additional reduction by up to an order of magnitude. Note, however, that the number of variables is unchanged—the extra variables added by the two constructions have different function, but can be mapped to each other one-to-one.

Table 2 empirically compares the effectiveness of the symmetry-breaking predicates described in Table 1. First, in most cases it takes much less time to find symmetries of a CNF instance than to solve it. The 2pipe instances are an exception, but we believe that further advances in symmetry-finding can rectify this exception. Second, the all-bits quadratic construction due to [Crawford *et al.*, 1996] is dramatically slower than our variants based on the cycle structure. Our linear-sized construction provides a further speed-up. Exceptions are the Hole-n benchmarks, where the difference between our two constructions is small, and the 2pipe benchmarks where the quadratic-sized SBPs led to shorter run times despite being almost thirteen times larger. Further investigation showed that this was a side-effect of mChaff's VSIDS decision heuristic which tended to choose the auxiliary variables first because they occur in many more clauses than the original instance variables. Using BerkMin [Gold-

[2] While our experiments clearly confirm that such generators exist, finding them is a function of the graph automorphism program. Since we used Nauty as a black box, we do not have a clear explanation of what causes Nauty to find such sparse generators, whether even sparser generators can be found and whether Nauty will in some cases return less sparse generators.

Instance	2-cycle generators		Generators with long cycles						
	Tot	Time (sec)	Max cycle length	Generators only		Generators and their powers		Generators and their comp.	
				Tot	Time (sec)	Tot	Time (sec)	Tot	Time (sec)
Hole-7	13	0.01	21	12	0.01	94	0.08	52	0.07
Hole-8	15	0.01	12	14	0.03	127	0.05	24	0.17
Hole-9	17	0.01	63	16	0.55	161	2.95	77	0.97
Hole-10	19	0.01	42	18	8.2	207	30	116	8.3

Table 3: mChaff search run times of "randomized" Hole-n instances augmented with linear SBPs based on different sets of permutations. Tot denotes the total number of permutations used in constructing each SBP.

berg and Novikov, 2002] or GRASP [Silva and Sakallah, 1999] in the same experiment yielded opposite results as expected.

Table 2 offers additional data to evaluate symmetry-breaking by generators, which may not be complete. We added symmetry-breaking predicates built for pairwise products of generators, but the overall runtimes increased. While additional SBPs may break more symmetries, their overhead does not justify their use.

Table 3 describes experiments with generators that have long cycles in which we evaluated extensions to symmetry-breaking by generators. Namely, we tried adding powers of all generators and, alternatively, adding pairwise products of generators. Neither extension proved useful, which supports our main symmetry-breaking approach.

6 Conclusions

The main contribution of our work is a better construction of symmetry-breaking predicates for Boolean satisfiability. We empirically demonstrate improvements both in the size of predicates and the run time of SAT solvers after these predicates are added to the original CNF instances. We also show that (1) symmetry-breaking by generators is difficult to improve upon, and that (2) the efficiency of symmetry-breaking does not improve when larger cycles are found in generators.

Our work articulates that better symmetry finding algorithms would be useful, especially if tailored to CNF formulas and, perhaps, the kinds of symmetry groups commonly found in structured CNF instances.

Acknowledgments

This work was funded by the DARPA/MARCO Gigascale Silicon Research Center and by an Agere Systems/SRC Research fellowship.

References

[Aloul *et al.*, 2002] F. Aloul, A. Ramani, I. Markov, and K. Sakallah, "Solving Difficult SAT Instances in the Presence of Symmetries," in *Proc. Design Automation Conf.* (DAC), pp. 731-736, 2002.

[Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. Luks, and A. Roy, "Symmetry-breaking predicates for search problems," in *Proc. Intl. Conf. Principles of Knowledge Representation and Reasoning,* pp. 148-159, 1996.

[Biere *et al.*, 1999] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu, "Symbolic Model Checking Using SAT Procedures Instead of BDDs," in *Proc. Design Automation Conf.* (DAC), pp. 317-320, 1999.

[DIMACS] DIMACS Challenge benchmarks in *ftp://Dimacs.rutgers.EDU/pub/challenge/sat/benchmarks/cnf.*

[Fraleigh, 2000] J. B. Fraleigh. A First Course in Abstract Algebra. 6th ed. Addison Wesley Longman, Reading, Massachusetts, 2000.

[Goldberg and Novikov, 2002] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust SAT-solver," in *Proc. Design, Automation, and Test in Europe Conf.* (DATE), pp. 142-149, 2002.

[McKay, 1981] B. McKay, "Practical Graph Isomorphism," in *Congressus Numerantium*, vol. 30, pp. 45-87, 1981. *http://cs.anu.edu.au/~bdm/nauty/*

[Moskewicz *et al.*, 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proc. Design Automation Conf.* (DAC), pp. 530-535, 2001.

[Nam *et al.*, 2001] G. Nam, F. Aloul, K. Sakallah, and R. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints," in *Proc. Intl. Symp. on Physical Design* (ISPD), pp. 222-227, 2001.

[SAT 2002] SAT 2002 Competition, *http://www.satlive.org/SATCompetition/submittedbenchs.html*

[Silva and Sakallah, 1999] J. Silva and K. Sakallah, "GRASP: A New Search Algorithm for Satisfiability," in *IEEE Trans. on Computers*, 48(5), pp. 506-521, 1999.

[Spitznagel, 1994] E. Spitznagel, "Review of Mathematical Software, GAP," in *Notices Amer. Math. Soc.*, 41(7), pp. 780-782, 1994.

[Stephan *et al.*, 1996] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "Combinational Test Generation Using Satisfiability," in *IEEE Trans. on Computer-Aided Design*, 15(9), pp. 1167-1175, 1996.

[Urquhart, 1987] A. Urquhart, "Hard Examples for Resolution," in *Journal of the ACM*, 34(1), pp. 209-219, 1987.

[Velev and Bryant, 2001] M. N. Velev and R. E. Bryant, "Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors", in *Proc. Design Automation Conf.* (DAC), pp. 226-231, 2001.