Notre Dame Journal of Formal Logic
Volume 34, Number 4, Fall 1993

BREAKUP: A Preprocessing Algorithm for Satisfiability Testing of CNF Formulas

ROBERT COWEN and KATHERINE WYATT

Abstract An algorithm called BREAKUP, which processes CNF formulas by separating them into "connected components," is introduced. BREAKUP is then used to speed up the testing of some first-order formulas for satisfiability using Iwama's **IS** Algorithm. The complexity of this algorithm is shown to be on the order of $O(nc \cdot nv)$, where nc is the number of clauses and nv is the number of variables.

1 Introduction Connectedness is an attribute of graphs that can be useful in deciding the satisfiability of propositional formulas in conjunctive normal form (CNF). Preprocessing to separate a formula into "connected components" is generally fast and easy, and can significantly speed up testing of the formula for satisfiability. BREAKUP, the algorithm introduced here, will be used in conjunction with Iwama's **IS** Algorithm [4],[5] and will be shown to significantly speed up (as compared to using **IS** alone) the processing of some formulas from Gilmore [3] and Davis and Putnam [2].

A CNF formula is a conjunction of clauses where each clause is a disjunction of literals. A literal is a propositional variable or the negation of a propositional variable. Given a pair of literals $\{p, \neg p\}$, each will be called the *mate* of the other (cf. Davis [1]). Clearly, we can assume that no clause contains both a literal and its mate. There are two ways to think of clauses in a formula as connected. First, two clauses c, d, are *connected* if there exists a chain of clauses starting with c and ending with d such that for any two adjacent clauses x, y, either y contains a literal that occurs in x or y contains the mate of a literal that occurs in x. Alternatively, c and d can be said to be *connected* if there is a chain of clauses starting with c and ending with d such that if x and y are adjacent then y contains the mate of a literal that occurs in x. In either case, a set of clauses is said to be *connected* if every pair in the set is connected and a *component* is a maximally connected set of the clauses of the formula.

Received July 15, 1992; revised March 15, 1993

The two definitions of connectedness will yield different component structures of a formula S only if S contains one or more pure literals (a literal is *pure* in S if its mate is absent from S). If each literal has a mate in S, S is called a *linked conjunct* ([1], p. 326); in a linked conjunct all clauses containing the literal or its mate end up in the same component under either definition. However, if a formula contains a pure literal, then all the clauses containing this pure literal will be in the same component under the first definition, but they may be in different components under the second definition. The second definition shall be adopted since it is slightly simpler; in fact, pure literals can easily be eliminated (pure literal rule, see [2]), in which case both definitions yield the same components.

Theorem 1 *A CNF formula is satisfiable if and only if each of its components is satisfiable.*

Proof: If a CNF formula is satisfiable, then certainly each component of the formula is satisfiable, since any assignment satisfying the formula will satisfy each component. Conversely, assume each component is satisfiable. Then clearly a satisfying assignment for each component can be chosen which in addition makes all the mateless literals in the component true. If there is a literal which occurs in more than one component this must be because its mate is absent, hence it will have been assigned the value T in each component in which it occurs. Therefore the union of the assignments chosen for each component will be an assignment satisfying the formula.

The algorithm BREAKUP, which finds the components, will be described in more detail below. Satisfiability testing on each component "delivered" by BREAKUP is done by Iwama's **IS** algorithm [4],[5]. The component testing is done serially, but obviously could be done simultaneously (in parallel). For comparison, Iwama's algorithm is used alone on the entire formula.

The following two first-order formulas, found in [3] and [2], were tested for validity using BREAKUP and the **IS** algorithm.

1. $(\exists x)(\exists y)(\forall z)((F(x,y) \Rightarrow (F(y,z) \wedge F(z,z))) \wedge ((F(x,y) \wedge G(x,y)) \Rightarrow (G(x,z) \wedge G(z,z))))$
2. $(\exists x)(\forall y)(\forall z)((((F(y,z) \Rightarrow (G(y) \Rightarrow H(x))) \Rightarrow F(x,x)) \wedge ((F(z,x) \Rightarrow G(x)) \Rightarrow H(z)) \wedge F(x,y)) \Rightarrow F(z,z))$.

The formulas were first negated to test them for validity; that is, their negations were tested for (un)satisfiability. As in Davis [1],[2], hand computation first produced a set of ground clauses from the first-order formulas. Satisfiability testing was then done by computer (a rather slow IBM PC clone) using BREAKUP, followed by our implementation of Iwama's algorithm.

In case (1), 75 clauses were generated initially (the same set of clauses shown to be unsatisfiable in [2]); after eliminating those with only pure literals, 24 clauses remained. There were 30 variables in the reduced set, and two connected components: one with 18 clauses and one with 6 clauses. The first component was unsatisfiable. In case (2), 114 clauses were needed. Fifty-eight remained after eliminating clauses with only pure literals; there were 59 variables in the reduced

set of clauses. The formula broke into 10 components and the second component was unsatisfiable.

There was a dramatic decrease in the runtime of the **IS** algorithm on components of a formula as compared to the whole formula. In the case of Formula 1, the two components were tested in 68 seconds, while checking the whole formula took 1822 seconds. Testing the second formula in its entirety proved to be too long a job for the PC. The program was aborted after 78 hours as it appeared to be only midway through the search. However, testing the 10 components of this formula was accomplished in 38 seconds. Therefore, it seems that including an algorithm like BREAKUP in testing formulas for satisfiability with Iwama's algorithm is well worthwhile. We feel other methods of testing of CNF formulas could also benefit from preprocessing with BREAKUP.

2 Description of Routine BREAKUP

2.1 Structures The algorithm used by BREAKUP to separate CNF formulas into components is an adaptation of a breadth-first search, as used to find components of a graph, or strongly connected components of a digraph. The CNF formula is read in from a disk file. It has *nc* clauses and *nv* variables. The clauses are indexed $c_1, c_2, \ldots, c_{nc}$, and the variables are indexed $v_1, v_2, \ldots, v_{nv}$.

The CNF is read in as a packed string, and the program builds the following data arrays.

PUR: a $2 \times nv$ array, where PUR(1,i) and PUR(2,i) count the number of positive and negative occurrences of v_i. If v_i occurs only negated or unnegated (i.e., as a pure literal), then one of PUR(1,i) or PUR(2,i) will be zero.

IM: a $nc \times nv$ incidence matrix, with

$$\begin{aligned} \mathrm{IM}(i,j) &= +1, \text{ if } v_j \text{ is not negated in } c_i \\ &= -1, \text{ if } v_j \text{ is negated in } c_i \\ &= 0, \text{ if } v_j \text{ does not appear in } c_i. \end{aligned}$$

VIN: a $nv \times (nc + 1)$ array, in which the i-th row is a list of the clauses in which v_i occurs, followed by at least one zero to indicate the end of the list and fill out the row. The extra column of VIN can be eliminated by minor changes in the program, but it seemed convenient and has little effect on the complexity.

COMP: the list of component indices of the clauses.

CQ: a queue of the clauses in a particular component. A clause is not added to CQ unless the corresponding index in COMP is zero; this eliminates unnecessary repetition. The search uses two queue pointers, bot and top. Bot points to the cell in the queue that holds the number of the row in the incidence matrix that is currently being scanned; top points to the last clause added to the queue.

2.2 Outline of Algorithm Initially put $m = 1$ and component index $ci = 1$. Clause c_m is chosen as the root of the first component and is entered in CQ. Bot

and top point to c_m and row 1 of the incidence matrix IM is scanned. Each v_k which occurs as a nonpure literal in c_m is selected in turn: the nonzero entries of row k in VIN (other than c_m) are added to the queue, and the value of ci is stored in COMP for each new queue member. The pointer to the top of the queue is incremented by one after each addition. Moreover, when c_j is added to the queue, IM(j,k) is changed to zero to avoid examining row k of VIN more than once. Variables which appear as pure literals are ignored, except that IM(j,k) is changed to zero.

The pointer to the bottom of the queue moves to the second cell in the queue, which contains some c_j, and the jth row of IM is scanned as above. This pointer continues to move from cell to cell in the queue; when the bottom pointer reaches the cell top is pointing to, the connected component is complete and ci is incremented. A search for a new connected component is begun, using as root c_m the next clause in the sequence of clauses which still has a zero in COMP. When these have been exhausted, all components have been found.

2.3 Analysis of Complexity Initializing array entries to zero in IM, VIN, PUR, COMP, and CQ accounts for $(nc \times nv) + ((nc + 1) \times nv) + 2nv + 2nc$ operations. Let $p(i)$ denote the number of occurrences of v_i and let $S = \Sigma p(i)$. S is at most $nc \times nv$, and unpacking the CNF and recording it in IM, VIN, and PUR involve $4S$ steps.

COMP is checked once for every clause in the outer loop of the algorithm, and again for every occurrence of a nonpure literal, or at most $nc + S$ times. The bottom pointer on CQ is compared with the top pointer nc times. Every row of IM is scanned once, for a total of $nc \times nv$ steps; and checking PUR for pure literals adds nv steps. At most S nonzero entries of VIN are checked for addition to the queue, since pure literals are not considered. One zero in each row of VIN, or nc in all, is also scanned. Entries (c_i, v_j) in IM are changed to zero when clause c_i is added to the queue and when variable v_j is a pure literal, or in less than S instances. Therefore, running BREAKUP involves at most T operations:

$$\begin{aligned} T &= 2(nc \times nv) + ((nc + 1) \times nv) + 3nv + 5nc + 7S \\ &\leq 10((nc + 1) \times nv) + 3nv + 5nc \leq 10(nc + 1)(nv + 1). \end{aligned}$$

Even though not all operations take the same time, the complexity is on the order of $O(nc \cdot nv)$.

REFERENCES

[1] Davis, M., "Eliminating the irrelevant from mechanical proofs," *Proceedings, Symposia of Applied Mathematics*, vol. 15 (1963), pp. 15–30. Reprinted in [6], pp. 315–330.

[2] Davis, M. and H. Putnam, "A computing procedure for quantification theory," *Journal of the Association for Computing Machinery*, vol. 7 (1960), pp. 201–215.

[3] Gilmore, P., "A proof method for quantification theory: its justification and realization," *IBM Journal of Research and Development*, vol. 4 (1960), pp. 28–35. Reprinted in [6], pp. 151–158.

[4] Iwama, K., "Complementary approaches to CNF Boolean equations," pp. 223–236 in *Discrete Algorithms and Complexity*, edited by D. Johnson, Academic Press, New York, 1987.

[5] Iwama, K., "CNF satisfiability test by counting and polynomial average time," *SIAM Journal of Computing*, vol. 18 (1989), pp. 385–391.

[6] Siekmann, J. and G. Wrightson, editors, *The Automation of Reasoning*, Volume I, Springer-Verlag, New York, 1983.

Department of Mathematics
Queens College
The City University of New York
Flushing, New York 11367

Department of Mathematics
Graduate Center
The City University of New York
New York, New York 10036