GSAT and Dynamic Backtracking

Matthew L. Ginsberg
CIRL
1269 University of Oregon
Eugene, OR 97403

David A. McAllester
MIT AI Laboratory
545 Technology Square
Cambridge, MA 02139

Abstract

There has been substantial recent interest in two new families of search techniques. One family consists of nonsystematic methods such as GSAT; the other contains systematic approaches that use a polynomial amount of justification information to prune the search space. This paper introduces a new technique that combines these two approaches. The algorithm allows substantial freedom of movement in the search space but enough information is retained to ensure the systematicity of the resulting analysis. Bounds are given for the size of the justification database and conditions are presented that guarantee that this database will be polynomial in the size of the problem in question.

1 INTRODUCTION

The past few years have seen rapid progress in the development of algorithms for solving constraint-satisfaction problems, or CSPs. CSPs arise naturally in subfields of AI from planning to vision, and examples include propositional theorem proving, map coloring and scheduling problems. The problems are difficult because they involve search; there is never a guarantee that (for example) a successful coloring of a portion of a large map can be extended to a coloring of the map in its entirety.

The algorithms developed recently have been of two types. *Systematic* algorithms determine whether a solution exists by searching the entire space. *Local* algorithms use hill-climbing techniques to find a solution quickly but are *nonsystematic* in that they search the entire space in only a probabilistic sense.

The empirical effectiveness of these nonsystematic algorithms appears to be a result of their ability to follow local gradients in the search space. Traditional systematic procedures explore the space in a fixed order that is independent of local gradients; the fixed order makes following local gradients impossible but is needed to ensure that no node is examined twice and that the search remains systematic.

Dynamic backtracking [Ginsberg,1993] attempts to overcome this problem by retaining specific information about those portions of the search space that have been eliminated and then following local gradients in the remainder. Unlike previous algorithms that recorded such elimination information, such as dependency-directed backtracking [Stallman and Sussman,1977], dynamic backtracking is selective about the information it caches so that only a polynomial amount of memory is required. These earlier techniques cached a new result with every backtrack, using an amount of memory that was linear in the run time and thus exponential in the size of the problem being solved.

Unfortunately, neither dynamic nor dependency-directed backtracking (or any other known similar method) is truly effective at local maneuvering within the search space, since the basic underlying methodology remains simple chronological backtracking. New techniques are included to make the search more efficient, but an exponential number of nodes in the search space must still be examined before early choices can be retracted. No existing search technique is able to both move freely within the search space and keep track of what has been searched and what hasn't.

The second class of algorithms developed recently presume that freedom of movement is of greater importance than systematicity. Algorithms in this class achieve their freedom of movement by abandoning the conventional description of the search space as a tree of partial solutions, instead thinking of it as a space of total assignments of values to variables. Motion is permitted between any two assignments that differ on a single value, and a hill-climbing procedure is employed to try to minimize the number of constraints violated by the overall assignment. The best-known algorithms in this class are min-conflicts [Minton *et al.*,1990] and GSAT [Selman *et al.*,1992].

Min-conflicts has been applied to the scheduling domain specifically and used to schedule tasks on the Hubble space telescope. GSAT is restricted to Boolean satisfiability problems (where every variable is assigned simply true or false), and has led to remarkable progress in the solution of randomly generated problems of this type; its performance is reported [Selman and Kautz,1993, Selman *et al.*,1992, Selman *et al.*,1993] as surpassing that of other techniques such as simulated annealing [Kirkpatrick *et al.*,1982] and systematic techniques based on the Davis-Putnam procedure [Davis and Putnam,1960].

GSAT is not a panacea, however; there are many problems on which it performs fairly poorly. If a problem has no solution, for example, GSAT will never be able to report this with confidence. Even if a solution does exist, there appear to be at least two possible difficulties that GSAT may encounter.

First, the GSAT search space may contain so many local minima that it is not clear how GSAT can move so as to reduce the number of constraints violated by a given assignment. As an example, consider the CSP of generating crossword puzzles by filling words from a fixed dictionary into an empty frame [Ginsberg *et al.*,1990]. The constraints indicate that there must be no conflict in each of the squares; thus two words that begin on the same square must also begin with the same letter. In this domain, getting "close" is not necessarily any indication that the problem is nearly solved, since correcting a conflict at a single square may involve modifying much of the current solution. Konolige has recently reported that GSAT specifically has difficulty solving problems of this sort [Konolige,1994].

Second, GSAT does no forward propagation. In the crossword domain once again, selecting one word may well force the selection of a variety of subsequent words. In a Boolean satisfiability problem, assigning one variable the value true may cause an immediate cascade of values to be assigned to other variables via a technique known as *unit resolution.* It seems plausible that forward propagation will be more common on realistic problems than on randomly generated ones; the most difficult random problems appear to be tangles of closely related individual variables while naturally occurring problems tend to be tangles of sequences of related variables. Furthermore, it appears that GSAT's performance degrades (relative to systematic approaches) as these sequences of variables arise [Crawford and Baker,1994].

Our aim in this paper is to describe a new search procedure that appears to combine the benefits of both of the earlier approaches; in some very loose sense, it can be thought of as a systematic version of GSAT.

The next three sections summarize the original dynamic backtracking algorithm [Ginsberg,1993], presenting it from the perspective of local search. The termination proof is omitted here but can be found in earlier papers [Ginsberg,1993, McAllester,1993]. Section 5 present a modification of dynamic backtracking called *partial-order dynamic backtracking,* or PDB. This algorithm builds on work of McAllester's [McAllester,1993]. Partial-order dynamic backtracking provides greater flexibility in the allowed set of search directions while preserving systematicity and polynomial worst case space usage. Section 6 presents a new variant of dynamic backtracking that is still more flexible in the allowed set of search directions. While this final procedure is still systematic, it can use exponential space in the worst case. Section 7 presents some empirical results comparing PDB with other well known algorithms on a class of "local" randomly generated 3-SAT problems. Concluding remarks are contained in Section 8, and proofs appear in the appendix.

2 CONSTRAINTS AND NOGOODS

We begin with a slightly nonstandard definition of a CSP.

Definition 2.1 *By a* constraint satisfaction problem (I, V, κ) *we will mean a finite set* I *of variables; for each* $x \in I$, *there is a finite set* V_x *of possible values for the variable* x. κ *is a set of constraints each of the form* $\neg[(x_1 = v_1) \wedge \cdots \wedge (x_k = v_k)]$ *where each* x_j *is a variable in* I *and each* v_j *is an element of* V_{x_j}. *A* solution *to the* CSP *is an assignment* P *of values to variables that satisfies every constraint. For each variable* x *we require that* $P(x) \in V_x$ *and for each constraint* $\neg[(x_1 = v_1) \wedge \cdots \wedge (x_k = v_k)]$ *we require that* $P(x_i) \neq v_i$ *for some* x_i.

By the size *of a constraint-satisfaction problem* (I, V, κ), *we will mean the product of the domain sizes of the various variables,* $\prod_x |V_x|$.

The technical convenience of the above definition of a constraint will be clear shortly. For the moment, we merely note that the above description is clearly equivalent to the conventional one; rather than represent the constraints in terms of allowed value combinations for various variables, we write axioms that disallow specific value combinations one at a time. The size of a CSP is the number of possible assignments of values to variables.

Systematic algorithms attempting to find a solution to a CSP typically work with partial solutions that are then discovered to be inextensible or to violate the given constraints; when this happens, a backtrack occurs and the partial solution under consideration is modified. Such a procedure will, of course, need to record information that guarantees that the same partial solution not be considered again as the search proceeds. This information might be recorded in the structure of the search itself; depth-first search with chronological backtracking is an example. More sophisticated methods maintain a database of some form indicating explicitly which choices have been eliminated and which have not. In this paper, we will use a database consisting of a set of *nogoods* [de Kleer,1986].

Definition 2.2 *A* nogood *is an expression of the form*

$$(x_1 = v_1) \wedge \cdots \wedge (x_k = v_k) \rightarrow x \neq v \qquad (1)$$

A nogood can be used to represent a constraint as an implication; (1) is logically equivalent to the constraint

$$\neg[(x_1 = v_1) \wedge \cdots \wedge (x_k = v_k) \wedge (x = v)]$$

There are clearly many different ways of representing a given constraint as a nogood.

One special nogood is the *empty* nogood, which is tautologically false. We will denote the empty nogood by $\perp$; if $\perp$ can be derived from the given set of constraints, it follows that no solution exists for the problem being attempted.

The typical way in which new nogoods are obtained is by resolving together old ones. As an example, suppose we have derived the following:

$$\begin{array}{rcl} (x = a) \wedge (y = b) & \rightarrow & u \neq v_1 \\ (x = a) \wedge (z = c) & \rightarrow & u \neq v_2 \\ (y = b) & \rightarrow & u \neq v_3 \end{array}$$

where v_1, v_2 and v_3 are the only values in the domain of u. It follows that we can combine these nogoods to conclude that there is no solution with

$$(x = a) \wedge (y = b) \wedge (z = c) \qquad (2)$$

Moving z to the conclusion of (2) gives us

$$(x = a) \wedge (y = b) \rightarrow z \neq c$$

In general, suppose we have a collection of nogoods of the form

$$x_{i1} = v_{i1} \wedge \cdots \wedge x_{in_i} = v_{in_i} \rightarrow x \neq v_i$$

as i varies, where the same variable appears in the conclusions of all the nogoods. Suppose further that the antecedents all agree as to the value of the x_i's, so that any time x_i appears in the antecedent of one of the nogoods, it is in a term $x_i = v_i$ for a fixed v_i. If the nogoods collectively eliminate all of the possible values for x, we can conclude that $\bigwedge_j (x_j = v_j)$ is inconsistent; moving one specific x_k to the conclusion gives us

$$\bigwedge_{j \neq k} (x_j = v_j) \rightarrow x_k \neq v_k \qquad (3)$$

As before, note the freedom in our choice of variable appearing in the conclusion of the nogood. Since the next step in our search algorithm will presumably satisfy (3) by changing the value for x_k, the selection of consequent variable corresponds to the choice of variable to "flip" in the terms used by GSAT or other hill-climbing algorithms.

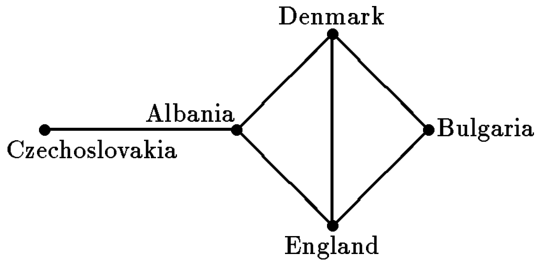

Figure 1: A small map-coloring problem

As we have remarked, dynamic backtracking accumulates information in a set of nogoods. To see how this is done, consider the map coloring problem in Figure 1, repeated from [Ginsberg,1993]. The map consists of five countries: Albania, Bulgaria, Czechoslovakia, Denmark and England. We assume – wrongly – that the countries border each other as shown in the figure, where countries are denoted by nodes and border one another if and only if there is an arc connecting them.

In coloring the map, we can use the three colors red, green and blue. We will typically abbreviate the colors and country names to single letters in the obvious way. The following table gives a trace of how a conventional dependency-directed backtracking scheme might attack this problem; each row shows a state of the procedure in the middle of a backtrack step, after a new nogood has been identified but before colors are erased to reflect the new conclusion. The coloring that is about to be removed appears in boldface. The "drop" column will be discussed shortly.

A	B	C	D	E	add		drop
r	g	**r**			$A = r \rightarrow C \neq r$	1	
r	g	b	**r**		$A = r \rightarrow D \neq r$	2	
r	g	b	**g**		$B = g \rightarrow D \neq g$	3	
r	g	b	b	**r**	$A = r \rightarrow E \neq r$	4	
r	g	b	b	**g**	$B = g \rightarrow E \neq g$	5	
r	g	b	b	**b**	$D = b \rightarrow E \neq b$	6	
r	g	b	**b**		$(A = r) \wedge (B = g) \rightarrow D \neq b$	7	6
r	**g**	b			$A = r \rightarrow B \neq g$	8	3, 5, 7

We begin by coloring Albania red and Bulgaria green, and then try to color Czechoslovakia red as well. Since this violates the constraint that Albania and Czechoslovakia be different colors, nogood (1) in the above table is produced.

We change Czechoslovakia's color to blue and then turn to Denmark. Since Denmark cannot be colored red or green, nogoods (2) and (3) appear; the only remaining color for Denmark is blue.

Unfortunately, having colored Denmark blue, we cannot color England. The three nogoods generated are (4), (5) and (6), and we can resolve these together because the three conclusions eliminate all of the possible colors for England. The result is that there is no solution with $(A = r) \wedge (B = g) \wedge (D = b)$, which we rewrite as (7) above. This can in turn be resolved with (2) and (3) to get (8), correctly indicating that the color of red for Albania is inconsistent with the choice of green for Bulgaria. The analysis can continue at this point to gradually determine that Bulgaria has to be red, Denmark can be green or blue, and England must then be the color not chosen for Denmark.

As we mentioned in the introduction, the problem with this approach is that the set Γ of nogoods grows monotonically, with a new nogood being added at every step. The number of nogoods stored therefore grows linearly with the run time and thus (presumably) exponentially with the size of the problem. A related problem is that it may become increasingly difficult to extend the partial solution P without violating one of the nogoods in Γ.

Dynamic backtracking deals with this by discarding nogoods when they become "irrelevant" in the sense that their antecedents no longer match the partial solution in question. In the example above, nogoods can be eliminated as indicated in the final column of the trace. When we derive (7), we remove (6) because Denmark is no longer colored blue. When we derive (8), we remove all of the nogoods with $B = g$ in their antecedents. Thus the only information we retain is that Albania's red color precludes red for Czechoslovakia, Denmark and England (1, 2 and 4) and also green for Bulgaria (8).

3 DYNAMIC BACKTRACKING

Dynamic backtracking uses the set of nogoods to both record information about the portion of the search space that has been eliminated and to record the current partial assignment being considered by the procedure. The current partial assignment is encoded in the antecedents of the current nogood set. More formally:

Definition 3.1 *An* acceptable next assignment *for a nogood set* Γ *is an assignment* P *satisfying every nogood in* Γ *and every antecedent of every such nogood. We will call a set of nogoods* Γ acceptable *if no two nogoods in* Γ *have the same conclusion and either* $\perp \in \Gamma$ *or there exists an acceptable next assignment for* Γ.

If Γ is acceptable, the antecedents of the nogoods in Γ induce a partial assignment of values to variables; any acceptable next assignment must be an extension of this partial assignment. In the above table, for example, nogoods (1) through (6) encode the partial assignment given by $A = r$, $B = g$, and $D = b$. Nogoods (1) though (7) fail to encode a partial assignment because the seventh nogood is inconsistent with the partial assignment encoded in nogoods (1) through (6). This is why the sixth nogood is removed when the seventh nogood is added.

Procedure 3.2 (Dynamic backtracking) To solve a CSP:

$P :=$ any complete assignment of values to variables
$\Gamma := \emptyset$
until either P is a solution or $\perp \in \Gamma$:
    $\gamma :=$ any constraint violated by P
    $\Gamma := \mathtt{simp}(\Gamma \cup \gamma)$
    $P :=$ any acceptable next assignment for Γ

To simplify the discussion we assume a fixed total order on the variables. Versions of dynamic backtracking with dynamic rearrangement of the variable order can be found elsewhere [Ginsberg,1993, McAllester,1993]. Whenever a new nogood is added, the fixed variable ordering is used to select the variable that appears in the conclusion of the nogood – the latest variable always appears in the conclusion. The subroutine `simp` closes the set of nogoods under the resolution inference rule discussed in the previous section and removes all nogoods which have an antecedent $x = v$ such that $x \neq v$ appears in the conclusion of some other nogood. Without giving a detailed analysis, we note that simplification ensures that Γ remains acceptable. To prove termination we introduce the following notation:

Definition 3.3 *For any acceptable* Γ *and variable* x, *we define the* live domain *of* x *to be those values* v *such that* $x \neq v$ *does not appear in the conclusion of any nogood in* Γ. *We will denote the size of the live domain of* x *by* $|x|_\Gamma$, *and will denote by* $m(\Gamma)$ *the tuple* $\langle |x_1|_\Gamma, \ldots, |x_n|_\Gamma \rangle$ *where* $x_1, \ldots, x_n$ *are the variables in the* CSP *in their specified order.*

Given an acceptable Γ, *we define the* size *of* Γ *to be*

$$\mathtt{size}(\Gamma) = \prod_x |V_x| - \sum_x \left[(|V_x| - |x|_\Gamma) \prod_{x_i > x} |V_{x_i}| \right]$$

Informally, the size of Γ is the size of the remaining search space given the live domains for the variables and assuming that all information about x_i will be lost when we change the value for any variable $x_j < x_i$.

The following result is obvious:

Lemma 3.4 *Suppose that* Γ *and* Γ' *are such that* $m(\Gamma)$ *is lexicographically less than* $m(\Gamma')$. *Then* $\mathtt{size}(\Gamma) < \mathtt{size}(\Gamma')$. ■

The termination proof (which we do not repeat here) is based on the observation that every simplification lexicographically reduces $m(\Gamma)$. Assuming that $\Gamma = \emptyset$ initially, since

$$\texttt{size}(\emptyset) = \prod_x |V_x|$$

it follows that the running time of dynamic backtracking is bounded by the size of the problem being solved.

Proposition 3.5 *Any acceptable set of nogoods can be stored in $o(n^2 v)$ space where n is the number of variables and v is the maximum domain size of any single variable.*

It is worth considering the behavior of Procedure 3.2 when applied to a CSP that is the union of two disjoint CSPs that do not share variables or constraints. If each of the two subproblems is unsatisfiable and the variable ordering interleaves the variables of the two subproblems, a classical backtracking search will take time proportional to the product of the times required to search each assignment space separately.[1] In contrast, Procedure 3.2 works on the two problems independently, and the time taken to solve the union of problems is therefore the sum of the times needed for the individual subproblems. It follows that Procedure 3.2 is fundamentally different from classical backtracking or backjumping procedures; Procedure 3.2 is in fact what has been called a *polynomial space aggressive backtracking procedure* [McAllester,1993].

4 DYNAMIC BACKTRACKING AS LOCAL SEARCH

Before proceeding, let us highlight the obvious similarities between Procedure 3.2 and Selman's description of GSAT [Selman *et al.*,1992]:

Procedure 4.1 **(GSAT)** To solve a CSP:

for $i := 1$ to MAX-TRIES
    $P :=$ a randomly generated truth assignment
    for $j := 1$ to MAX-FLIPS
        if P is a solution, **then** return it
        else flip any variable in P that results in
                the greatest decrease in the number
                of unsatisfied clauses
        end if
    end for
end for
return failure

The inner loop of the above procedure makes a local move in the search space in a direction consistent with the goal of satisfying a maximum number of clauses; we will say that GSAT follows the local gradient of a "maxsat" objective function. But local search can get stuck in local minima; the outer loop provides a partial escape by giving the procedure several independent chances to find a solution.

Like GSAT, dynamic backtracking examines a sequence of total assignments. Initially, dynamic backtracking has considerable freedom in selecting the next assignment; in many cases, it can update the total assignment in a manner identical to GSAT. The nogood set ultimately both constrains the allowed directions of motion and forces the procedure to search systematically. Dynamic backtracking cannot get stuck in local minima.

Both systematicity and the ability to follow local gradients are desirable. The observations of the previous paragraphs, however, indicate that these two properties are in conflict – systematic enumeration of the search space appears incompatible with gradient descent. To better understand the interaction of systematicity and local gradients, we need to examine more closely the structure of the nogoods used in dynamic backtracking.

We have already discussed the fact that a single constraint can be represented as a nogood in a variety of ways. For example, the constraint $\neg(A = r \wedge B = g)$ can be represented either as $A = r \rightarrow B \neq g$ or as $B = g \rightarrow A \neq r$. Although these nogoods capture the same information, they behave differently in the dynamic backtracking procedure because they encode different partial truth assignments and represent different choices of variable ordering. In particular, the set of acceptable next assignments for $A = r \rightarrow B \neq g$ is quite different from the set of acceptable next assignments for $B = g \rightarrow A \neq r$. In the former case an acceptable assignment must satisfy $A = r$; in the latter case, $B = g$ must hold. Intuitively, the former nogood corresponds to changing the value of B while the latter nogood corresponds to changing that of A. The manner in which we represent the constraint $\neg(A = r \wedge B = g)$ influences the direction in which the search is allowed to proceed. In Procedure 3.2, the choice of representation is forced by the need to respect the fixed variable ordering and to change the latest variable in the constraint.[2] Similar restrictions exist in the original presentation of dynamic backtracking itself [Ginsberg,1993].

[1]This observation remains true even if backjumping techniques are used.

[2]Note, however, that there is still considerable freedom in the choice of the constraint itself. A total assignment usually violates many different constraints.

5 PARTIAL-ORDER DYNAMIC BACKTRACKING

Partial-order dynamic backtracking [McAllester,1993] replaces the fixed variable order with a *partial* order that is dynamically modified during the search. When a new nogood is added, this partial ordering need not fix a unique representation – there can be considerable choice in the selection of the variable to appear in the conclusion of the nogood. This leads to freedom in the selection of the variable whose value is to be changed, thereby allowing greater flexibility in the directions that the procedure can take while traversing the search space. The locally optimal gradient followed by GSAT can be adhered to more often. The partial order on variables is represented by a set of ordering constraints called *safety conditions.*

Definition 5.1 *A* safety condition *is an assertion of the form $x < y$ where x and y are variables. Given a set S of safety conditions, we will denote by $\leq_S$ the transitive closure of $<$, saying that S is* acyclic *if $\leq_S$ is antisymmetric. We will write $x <_S y$ to mean that $x \leq_S y$ and $y \not\leq_S x$.*

In other words, $x \leq y$ if there is some (possibly empty) sequence of safety conditions

$$x < z_1 < \ldots < z_n < y$$

The requirement of antisymmetry means simply that there are no two distinct x and y for which $x \leq y$ and $y \leq x$; in other words, $\leq_S$ has no "loops" and is a partial order on the variables. In this section, we restrict our attention to acyclic sets of safety conditions.

Definition 5.2 *For a nogood γ, we will denote by S_γ the set of all safety conditions $x < y$ such that x is in the antecedent of γ and y is the variable in its conclusion.*

Informally, we require variables in the antecedent of nogoods to precede the variables in their conclusions, since the antecedent variables have been used to constrain the live domains of the conclusions.

The state of the partial order dynamic backtracking procedure is represented by a pair $\langle \Gamma, S \rangle$ consisting of a set of nogoods and a set of safety conditions. In many cases, we will be interested in only the ordering information about variables that can precede a fixed variable x. To discard the rest of the ordering information, we discard all of the safety conditions involving any variable y that follows x, and then record only that y does indeed follow x. Somewhat more formally:

Definition 5.3 *For any set S of safety conditions and variable x, we define the* weakening of S at x, *to be denoted $W(S, x)$, to be the set of safety conditions given by removing from S all safety conditions of the form $z < y$ where $x <_S y$ and then adding the safety condition $x < y$ for all such y.*

The set $W(S, x)$ is a weakening of S in the sense that every total ordering consistent with S is also consistent with $W(S, x)$. However $W(S, x)$ usually admits more total orderings than S does; for example, if S specifies a total order then $W(S, x)$ allows any order which agrees with S up to and including the variable x. In general, we have the following:

Lemma 5.4 *For any set S of safety conditions, variable x, and total order $<$ consistent with the safety conditions in $W(S, x)$, there exists a total order consistent with S that agrees with $<$ through x.*

We now state the PDB procedure.

Procedure 5.5 To solve a CSP:

$P :=$ any complete assignment of values to variables
$\Gamma := \emptyset$
$S := \emptyset$
until either P is a solution or $\perp \in \Gamma$:
    $\gamma :=$ a constraint violated by P
    $\langle \Gamma, S \rangle := \texttt{simp}(\Gamma, S, \gamma)$
    $P :=$ any acceptable next assignment for Γ

Procedure 5.6 To compute $\texttt{simp}(\Gamma, S, \gamma)$:

select the conclusion x of γ so that $S \cup S_\gamma$ is acyclic
$\Gamma := \Gamma \cup \{\gamma\}$
$S := W(S \cup S_\gamma, x)$
remove from Γ each nogood with x in its antecedent
if the conclusions of nogoods in Γ rule out all
    possible values for x **then**
  $\rho :=$ the result of resolving all nogoods in Γ with x
    in their conclusion
  $\langle \Gamma, S \rangle := \texttt{simp}(\Gamma, S, \rho)$
end if
return $\langle \Gamma, S \rangle$

The above simplification procedure maintains the invariant that Γ be acceptable and S be acyclic; in addition, the time needed for a single call to simp appears to grow significantly sublinearly with the size of the problem in question (see Section 7).

Theorem 5.7 *Procedure 5.5 terminates. The number of calls to* simp *is bounded by the size of the problem being solved.*

As an example, suppose that we return to our map-coloring problem. We begin by coloring all of the countries red except Bulgaria, which is green. The following table shows the total assignment that existed at the moment each new nogood was generated.

A	B	C	D	E	add		drop
r	g	r	r	r	$C = r \rightarrow A \neq r$	1	
b	g	r	r	r	$D = r \rightarrow E \neq r$	2	
b	g	r	r	g	$B = g \rightarrow E \neq g$	3	
b	g	r	r	b	$A = b \rightarrow E \neq b$	4	
					$(A = b) \wedge (B = g) \rightarrow D \neq r$	5	2
					$D < E$	6	
b	g	r	g	r	$B = g \rightarrow D \neq g$	7	
b	g	r	b	r	$A = b \rightarrow D \neq b$	8	
					$A = b \rightarrow B \neq g$	9	3, 5, 7
					$B < E$	10	6
					$B < D$	11	

The initial coloring violates a variety of constraints; suppose that we choose to work on one with Albania in its conclusion because Albania is involved in three violated constraints. We choose $C = r \rightarrow A \neq r$ specifically, and add it as (1) above.

We now modify Albania to be blue. The only constraint violated is that Denmark and England be different colors, so we add (2) to Γ. This suggests that we change the color for England; we try green, but this conflicts with Bulgaria. If we write the new nogood as $E = g \rightarrow B \neq g$, we will change Bulgaria to blue and be done. In the table above, however, we have made the less optimal choice (3), changing the coloring for England again.

We are now forced to color England blue. This conflicts with Albania, and we continue to leave England in the conclusion of the nogood as we add (4). This nogood resolves with (2) and (3) to produce (5), where we have once again made the worst choice and put D in the conclusion. We add this nogood to Γ and remove nogood (2), which is the only nogood with D in its antecedent. In (6) we add a safety condition indicating that D must continue to precede E. (This safety condition has been present since nogood (2) was discovered, but we have not indicated it explicitly until the original nogood was dropped from the database.)

We next change Denmark to green; England is forced to be red once again. But now Bulgaria and Denmark are both green; we have to write this new nogood (7) with Denmark in the conclusion because of the ordering implied by nogood (5) above. Changing Denmark to blue conflicts with Albania (8), which we have to write as $A = b \rightarrow D \neq b$. This new nogood resolves with (5) and (7) to produce (9).

We drop (3), (5) and (7) because they involve $B = g$, and introduce the two safety conditions (10) and (11). Since E follows B, we drop the safety condition $E < D$. At this point, we are finally forced to change the color for Bulgaria and the search continues.

It is important to note that the added flexibility of PDB over dynamic backtracking arises from the flexibility in the first step of the simplification procedure where the conclusion of the new nogood is selected. This selection corresponds to a selection of a variable whose value is to be changed.

As with the procedure in the previous section, when given a CSP that is a union of disjoint CSPs the above procedure will treat the two subproblems independently. The total running time remains the sum of the times required for the subproblems.

6 ARBITRARY MOVEMENT

Partial-order dynamic backtracking still does not provide total freedom in the choice of direction through the search space. When a new nogood is discovered, the existing partial order constrains how we are to interpret that nogood – roughly speaking, we are forced to change the value of late variables before changing the values of their predecessors. The use of a partial order makes this constraint looser than previously, but it is still present. In this section, we allow cycles in the nogoods and safety conditions, thereby permitting arbitrary choice in the selection of the variable appearing in the conclusion of a new nogood.

The basic idea is the following: Suppose that we have introduced a loop into the variable ordering, perhaps by including the pair of nogoods $x \rightarrow \neg y$ and $y \rightarrow x$. Rather than rewrite one of these nogoods so that the same variable appears in the conclusion of both, we will view the (x, y) combination as a single variable that takes a value in the product set $V_x \times V_y$.

If x and y are variables that have been "combined" in this way, we can rewrite a nogood with (for example) x in its antecedent and y in its conclusion so that both x and y are in the conclusion. As an example, we can rewrite

$$x = v_x \wedge z = v_z \rightarrow y \neq v_y \qquad (4)$$

as

$$z = v_z \rightarrow (x, y) \neq (v_x, v_y) \qquad (5)$$

which is logically equivalent. We can view this as eliminating a particular value for the pair of variables (x, y).

Definition 6.1 *Let S be a set of safety conditions (possibly not acyclic). We will write $x \equiv_S y$ if $x \leq_S y$ and $y \leq_S x$. The equivalence class of x under $\equiv$ will be denoted $\langle x \rangle_S$. If γ is a nogood whose conclusion involves the variable x, we will denote by γ_S the result of moving to the conclusion of γ all terms involving members of $\langle x \rangle_S$. If Γ is a set of nogoods, we will denote by Γ_S is the set of nogoods of the form γ_S for $\gamma \in \Gamma$.*

It is not difficult to show that for any set S of safety conditions, the relation $\equiv_S$ is an equivalence relation. As an example of rewriting a nogood in the presence of ordering cycles, suppose that γ is the nogood (4) and let S be such that $\langle y\rangle_S = \{x, y\}$; now γ_S is given by (5).

Placing more than one literal in the conclusions of nogoods forces us to reconsider the notion of an acceptable next assignment:

Definition 6.2 *A* cyclically acceptable next assignment *for a nogood set Γ under a set S of safety conditions is a total assignment P of values to variables satisfying every nogood in Γ_S and every antecedent of every such nogood.*

We now define a third dynamic backtracking procedure. Note that $W(S, x)$ remains well defined even if S is not acyclic, since $W(S, x)$ drops ordering constraints only on variables y such that $x <_S y$.

Procedure 6.3 To solve a CSP:

$P :=$ any complete assignment of values to variables
$\Gamma := \emptyset$
$S := \emptyset$
until either P is a solution or $\perp \in \Gamma$:
  $\gamma :=$ a constraint violated by P
  $\langle \Gamma, S\rangle :=$ `simp`(Γ, S, γ)
  $P :=$ any cyclically acceptable next assignment
    for Γ under S

Procedure 6.4 To compute `simp`(Γ, S, γ):

select a conclusion x for γ (now unconstrained)
$\Gamma := \Gamma \cup \{\gamma\}$
$S := W(S \cup S_\gamma, x)$
remove from Γ each nogood α with an element of $\langle x\rangle_S$
    in the antecedent of α_S
if the conclusions of nogoods in Γ_S rule out all
    possible values for the variables in $\langle x\rangle_S$ **then**
  $\rho :=$ the result of resolving all nogoods in Γ_S whose
    conclusions involve variables in $\langle x\rangle_S$
  $\langle \Gamma, S\rangle :=$ `simp`(Γ, S, ρ)
  end if
return $\langle \Gamma, S\rangle$

If the conclusion is selected so that S remains acyclic, the above procedure is identical to the one in the previous section.

Proposition 6.5 *Suppose that we are working on a problem with n variables, that the size of the largest domain of any variable is v, and that we have constructed Γ and S using repeated applications of* `simp`. *If the largest equivalence class $\langle x\rangle_S$ contains d elements, the space required to store Γ is $o(n^2 v^d)$.*

If we have an equivalence class of d variables each of which has v possible values then the number of possible values of the "combined variable" is v^d. The above procedure can now generate a distinct nogood to eliminate each of the v^d possible values, and the space requirements of the procedure can therefore grow exponentially in the size of the equivalence classes. The time required to find a cyclically allowed next assignment can also grow exponentially in the size of the equivalence classes. We can address these difficulties by selecting in advance a bound for the largest allowed size of any equivalence class. In any event, termination is still guaranteed:

Theorem 6.6 *Procedure 6.3 terminates. The number of calls to* `simp` *is bounded by the size of the problem being solved.*

Selecting a variable to place in the conclusion of a new nogood corresponds to choosing the variable whose value is to be changed on the next iteration and is analogous to selecting the variable to flip in GSAT. Since the choice of conclusion is unconstrained in the above procedure, the procedure has tremendous flexibility in the way it traverses the search space. Like the procedures in the previous sections, Procedure 6.3 continues to solve combinations of independent subproblems in time bounded by the sum of the times needed to solve the subproblems individually.

Here are these ideas in use on a Boolean CSP with the constraints $a \to b$, $b \to c$ and $c \to \neg b$. As before, we present a trace and then explain it:

a	b	c	add to Γ		remove from Γ
t	f	f	$a \to b$	1	
t	t	f	$b \to c$	2	
t	t	t	$c \to \neg b$	3	
			$\neg a$	4	1
			$a < b$	5	

The first three nogoods are simply the three constraints appearing in the problem. Although the orderings of the second and third nogoods conflict, we choose to write them in the given form in any case.

Since this puts b and c into an equivalence class, we do not drop nogood (2) at this point. Instead, we interpret nogood (1) as requiring that the value taken by (b, c) be either (t, t) or (t, f); (2) disallows (t, f) and (3) disallows (t, t). It follows that the three nogoods can be resolved together to obtain the new nogood given simply by $\neg a$. We add this as (4) above, dropping nogood (1) because its antecedent is falsified.

7 EXPERIMENTAL RESULTS

In this section, we present preliminary results regarding the implemented effectiveness of the procedure we have described. The implementation is based on the somewhat restricted Procedure 5.5 as opposed to the more general Procedure 6.3. We compared a search engine based on this procedure with two others, TABLEAU

[Crawford and Auton,1993] and WSAT, or "walk-sat" [Selman *et al.*,1993]. TABLEAU is an efficient implementation of the Davis-Putnam algorithm and is systematic; WSAT is a modification to GSAT and is not. We used WSAT instead of GSAT because WSAT is more effective on a fairly wide range of problem distributions [Selman *et al.*,1993].

The experimental data was not collected using the random 3-SAT problems that have been the target of much recent investigation, since there is growing evidence that these problems are not representative of the difficulties encountered in practice [Crawford and Baker,1994]. Instead, we generated our problems so that the clauses they contain involve groups of locally connected variables as opposed to variables selected at random.

Somewhat more specifically, we filled an $n \times n$ square grid with variables, and then required that the three variables appearing in any single clause be neighbors in this grid. LISP code generating these examples appears in the appendix. We believe that the qualitative properties of the results reported here hold for a wide class of distributions where variables are given spatial locations and clauses are required to be local.

The experiments were performed at the crossover point where approximately half of the instances generated could be expected to be satisfiable, since this appears to be where the most difficult problems lie [Crawford and Auton,1993]. Note that not all instances at the crossover point are hard; as an example, the local variable interactions in these problems can lead to short resolution proofs that no solution exists in unsatisfiable cases. This is in sharp contrast with random 3-SAT problems (where no short proofs appear to exist in general, and it can even be shown that proof lengths are growing exponentially on average [Chvátal and Szemerédi,1988]). Realistic problems may often have short proof paths: A particular scheduling problem may be unsatisfiable simply because there is no way to schedule a specific resource as opposed to because of global issues involving the problem in its entirety. Satisfiability problems arising in VLSI circuit design can also be expected to have locality properties similar to those we have described.

The problems involved 25, 100, 225, 400 and 625 variables. For each size, we generated 100 satisfiable and 100 unsatisfiable instances and then executed the three procedures to measure their performance. (WSAT was not tested on the unsatisfiable instances.) For WSAT, we measured the number of times specific variable values were flipped. For PDB, we measured the number of top-level calls to Procedure 5.6. For TABLEAU, we measured the number of choice nodes expanded. WSAT and PDB were limited to 100,000 flips; TABLEAU was limited to a running time of 150 seconds.

The results for the satisfiable problems were as follows. For TABLEAU, we give the node count for successful runs only; we also indicate parenthetically what fraction of the problems were solved given the computational resource limitations. (WSAT and PDB successfully solved all instances.)

Variables	PDB	WSAT	TABLEAU
25	35	89	9 (1.0)
100	210	877	255 (1.0)
225	434	1626	504 (.98)
400	731	2737	856 (.70)
625	816	3121	502 (.68)

For the unsatisfiable instances, the results were:

Variables	PDB	TABLEAU
25	122	8 (1.0)
100	509	1779 (1.0)
225	988	5682 (.38)
400	1090	558 (.11)
625	1204	114 (.06)

The times required for PDB and WSAT appear to be growing comparably, although only PDB is able to solve the unsatisfiable instances. The eventual *decrease* in the average time needed by TABLEAU is because it is only managing to solve the easiest instances in each class. This causes TABLEAU to become almost completely ineffective in the unsatisfiable case and only partially effective in the satisfiable case. Even where it does succeed on large problems, TABLEAU's run time is greater than that of the other two methods.

Finally, we collected data on the time needed for each top-level call to `simp` in partial-order dynamic backtracking. As a function of the number of variables in the problem, this was:

Number of variables	PDB (msec)	WSAT (msec)
25	3.9	0.5
100	5.3	0.3
225	6.7	0.6
400	7.0	0.7
625	8.4	1.4

All times were measured on a Sparc 10/40 running unoptimized Allegro Common Lisp. An efficient C implementation could expect to improve either method by approximately an order of magnitude. As mentioned in Section 5, the time per flip is growing sublinearly with the number of variables in question.

8 CONCLUSION AND FUTURE WORK

Our aim in this paper has been to make a primarily theoretical contribution, describing a new class of constraint-satisfaction algorithms that appear to combine many of the advantages of previous systematic and nonsystematic approaches. Since our focus has been on a description of the algorithms, there is obviously much that remains to be done.

First, of course, the procedures must be tested on a variety of problems, both synthetic and naturally occurring; the results reported in Section 7 only scratch the surface. It is especially important that realistic problems be included in any experimental evaluation of these ideas, since these problems are likely to have performance profiles substantially different from those of randomly generated problems [Crawford and Baker,1994]. The experiments of the previous section need to be extended to include unit resolution, and we need to determine the frequency with which exponential space is needed in practice by the full procedure 6.3.

Finally, we have left completely untouched the question of how the flexibility of Procedure 6.3 is to be exploited. Given a group of violated constraints, which should we pick to add to Γ? Which variable should be in the conclusion of the constraint? These choices correspond to choice of backtrack strategy in a more conventional setting, and it will be important to understand them in this setting as well.

A PROOFS

Proposition 3.5 *Any acceptable set of nogoods can be stored in $o(n^2v)$ space where n is the number of variables and v is the maximum domain size of any single variable.*

Proof. This can be done by first storing the partial assignment encoded in Γ using $o(n)$ space. The antecedent of each nogood can now be represented as a bit vector specifying the set of variables appearing in the antecedent, allowing the nogood itself to be stored in $o(n)$ space. Since no two nogoods share the same conclusion there are at most nv nogoods. ■

Lemma 5.4 *For any set S of safety conditions, variable x and total order $<$ consistent with the safety conditions in $W(S,x)$, there is a total order consistent with S that agrees with $<$ through x.*

Proof. Suppose that the ordering $<$ is given by

$$x_1 < \cdots < x_k = x < y_1 < \cdots < y_m \qquad (6)$$

Now let $<'$ be any ordering consistent with S, and suppose that the ordering given by $<'$ on the y_i in (6) is

$$z_1 <' \cdots <' z_m$$

We claim that the ordering given by

$$x_1, \ldots, x_k = x, z_1, \ldots, z_m \qquad (7)$$

is consistent with all of S. We will show this by showing that (7) is consistent with any specific safety condition $u < v$ in S.

If both u and v are x_i's, then the safety condition $u < v$ will remain in $W(S,x)$ and is therefore satisfied by (7). If both u and v are z_i's, they are ordered as $u < v$ by $<'$ which is known to satisfy the safety conditions in S. If u is an x_i and v is a z_j, $u < v$ clearly follows from (7).

The remaining case is where $u = z_i$ and $v = x_j$ for some specific z_i and x_j. The safety condition $z_i < x_j$ cannot appear in $W(S,x)$, since it is violated by $<$ in (6). It must therefore be the case that $x_j >_S x$. But now $W(S,x)$ will include the safety condition $x_j > x$, in conflict with the ordering given by (6). This contradiction completes the proof. ■

Theorem 5.7 *Procedure 5.5 terminates. The number of calls to* `simp` *is bounded by the size of the problem being solved.*

Proof. In fact, we will not prove the theorem using Procedure 5.6 as stated. Instead, consider the following simplification procedure:

Procedure A.1 To compute $\mathtt{simp}'(\Gamma, S, \gamma)$:

select the conclusion x of γ so that $S \cup S_{\{\gamma\}}$ is acyclic
$\Gamma := \Gamma \cup \{\gamma\}$
$S := W(S \cup S_\gamma, x)$
remove from Γ any nogood with conclusion y such
that $y >_S x$
if the conclusions of nogoods in Γ rule out all
possible values for x **then**
$\rho :=$ the result of resolving all nogoods in Γ with x
in their conclusion
$\langle\Gamma, S\rangle := \mathtt{simp}(\Gamma, S, \rho)$
end if
return $\langle\Gamma, S\rangle$

The difference between this procedure and Procedure 5.6 is that where Procedure 5.6 removed only nogoods with x in their antecedents, Procedure A.1 removes all nogoods with conclusion following x in the partial order $<_S$.

We now have the following:

Lemma A.2 *Suppose that Γ and S are chosen so that $S \supseteq S_\gamma$ for each $\gamma \in \Gamma$. Now if γ is a nogood that violates some acceptable next assignment for Γ and $\langle\Gamma', S'\rangle = \mathtt{simp}'(\Gamma, S, \gamma)$, $S' \supseteq S_\gamma$ for each $\gamma \in \Gamma'$.*

Proof. It is clear that the lemma would hold if we were to take $S := S \cup S_\gamma$, so we must only show that the weakening at x cannot drop the safety condition associated with some nogood in Γ'. But if the weakening drops the safety condition $z < y$, it must be that $y >_{S \cup S_\gamma} x$. Since x is the variable in the conclusion of γ, this implies that we must have $y >_S x$, in which case the underlying nogood with y in its conclusion will have been deleted as well. ■

It follows from the lemma that if simp removes a nogood γ, simp$'$ will drop it as well, since if y is the variable in the conclusion of γ, we clearly have of $y >_{S_\gamma} x$ (since simp drops only nogoods with x in their antecedents) so that $y >_S x$ by virtue of the lemma and simp$'$ drops the nogood as well.

We therefore see that the difference between the two procedures is only in the set of nogoods maintained; Procedure 5.5 as stated retains a superset of the nogoods retained by a version based on simp$'$. The set S of safety conditions is the same in both cases, and the nogood set is acceptable in both cases. It thus suffices to prove the theorem for simp$'$, since the larger set of nogoods computed using simp will simply result in fewer acceptable next assignments for the procedure to consider.

To see that the procedure using simp$'$ terminates, we begin with the following definition:

Definition A.3 *Given a set of safety conditions S and a fixed variable ordering $x_1 < x_2 < \cdots < x_n$ that respects $<_S$, let $m(\Gamma, S, <)$ be the tuple $\langle |x_1|_\Gamma, \ldots, |x_n|_\Gamma \rangle$. We will denote by* $\mathtt{size}(\Gamma, S, <)$ *the size of the remaining search space as given in Definition 3.3, and will denote by* $\mathtt{size}(\Gamma, S)$ *the maximum size as $<$ is allowed to vary.*

Proposition A.4 *Suppose that Γ is acceptable, S is acyclic, and $S \supseteq S_\gamma$ for each $\gamma \in \Gamma$. Now if γ is a nogood that violates some acceptable next assignment for Γ and $\langle \Gamma', S' \rangle = \mathtt{simp}'(\Gamma, S, \gamma)$, then $\mathtt{size}(\Gamma', S') < \mathtt{size}(\Gamma, S)$.*

Proof. Let x be the variable in the conclusion of γ. The first nontrivial step of the procedure simp$'$ is $\Gamma := \Gamma \cup \{\gamma\}$. This reduces $|x|_\Gamma$. The next step is $S := W(S \cup S_\gamma, x)$. This introduces new orderings. Let $<$ be any total ordering consistent with $W(S \cup S_\gamma, x)$. There must now exist a total ordering $<'$ which is consistent with $S \cup S_\gamma$ such that $<$ and $<'$ agree through x. Since $|x|_\Gamma$ has been reduced, the tuple associated with $<$ must be lexicographically smaller then the tuple associated with $<'$ at the time the procedure was called. This implies that all tuples allowed after $S := W(S, x) \cup S_\Gamma$ are lexicographically smaller than some tuple allowed at the beginning of the simplification. Applying Lemma 3.4, we can conclude that the size of the $\langle \Gamma, S \rangle$ pair has been reduced.

The next step removes from Γ all nogoods with conclusion $y >_S x$. Although this increases the size of the live domain for y, the fact that $y >_S x$ allows us to repeat the lexicographic argument of the preceding paragraph. Finally, if the simplification performs a resolution and executes a recursive call, then that recursion must continue to decrease the size of $\langle \Gamma, S \rangle$. ∎

It follows that a modification of Procedure 5.5 using simp$'$ will in fact terminate in a number of steps bounded by the original value of $\mathtt{size}(\Gamma, S)$, which is the size of the problem being solved. Procedure 5.5 itself will terminate no less quickly. ∎

Proposition 6.5 *Suppose that we are working on a problem with n variables, that the size of the largest domain of any variable is v, and that we have constructed Γ and S using repeated applications of* simp. *If the largest equivalence class $\langle x \rangle_S$ contains d elements, the space required to store Γ is $o(n^2 v^d)$.*

Proof. We know that the nogood set will be acyclic if we group together variables that are equivalent under $\leq_\Gamma$. Since this results in at most d variables being grouped together at any point, the maximum domain size in the reduced problem is v^d and the maximum number of nogoods stored is thus bounded by nv^d. As previously, the amount of space needed to store each nogood is $o(n)$. ∎

Theorem 6.6 *Procedure 6.3 terminates. The number of calls to* simp *is bounded by the size of the problem being solved.*

Proof. The proof is essentially unchanged from that of Theorem 5.7; we provide only a sketch here. The only novel features of the proof involve showing that the lexicographic size falls as either variables are merged into an equivalence class or an equivalence class is broken so that the variables it contains are once again handled separately. In order to do this, we extend Definition A.3 to handle equivalence classes as follows:

Definition A.5 *Given a set of safety conditions S and a fixed variable ordering $x_1 < x_2 < \cdots < x_n$ that respects $<_S$, let $||x_i||$ be given by*

$$||x_i|| = \begin{cases} 1, & \text{if } x_i \equiv x_{i+1}; \\ \prod_{y \in \langle x_i \rangle_S} |y|_\Gamma, & \text{otherwise.} \end{cases} \quad (8)$$

Now denote by $\hat{m}(\Gamma, S, <)$ the tuple $\langle ||x_1||, \ldots, ||x_n|| \rangle$. We will denote by $\hat{m}(\Gamma, S)$ that tuple which is lexicographically maximal as $<$ is allowed to vary.

This definition ensures that the lexicographic value decreases whenever we combine variables, since the remaining choices for the combined variable aren't counted until the latest possible point. It remains to show that the removal of nogoods or safety conditions does not split an equivalence class prematurely.

This, however, is clear. If removing a safety condition $y < z$ causes two other variables y_1 and y_2 to become not equivalent, it must be the case that $y_1 \equiv y_2 \equiv z$ before the safety condition was removed. But note that when the safety condition is removed, we must have made progress on a variable $x <_S z$. There is thus no lexicographic harm in splitting z's equivalence class. ∎

Experimental code Here is the code used to generate instances of the class of problems on which our ideas were tested. The two arguments to the procedure are the size s of the variable grid and the number c of clauses to be "centered" on any single variable.

For each variable x on the grid we generated either $\lfloor c \rfloor$ or $\lfloor c \rfloor + 1$ clauses at random subject to the constraint that the variables in each clause form a right triangle with horizontal and vertical sides of length 1 and where x is the vertex opposite the hypotenuse. There are four such triangles for a given x. There are eight assignments of values to variable for each triangle giving 32 possible clauses. Our Common Lisp code for generating these 3-SAT problems is given below. Variables at the edge of the grid usually generate fewer than c clauses so the boundary of the grid is relatively unconstrained.

```
(defun make-problem (s c &aux result xx yy)
  (dotimes (x s)
    (dotimes (y s)
      (dotimes (i (+ (floor c)
                     (if (> (random 1.0)
                            (rem c 1.0))
                         0 1)))
        (setq xx (+ x -1 (* 2 (random 2)))
              yy (+ y -1 (* 2 (random 2))))
        (when (and (< -1 xx) (< xx s)
                   (< -1 yy) (< yy s))
          (push (new-clause x y xx yy s)
                result)))))
  result))

(defun new-clause (x y xx yy s)
  (mapcar
   #'(lambda (a b &aux (v (+ 1 (* s a) b)))
       (if (zerop (random 2)) v (- v)))
   (list x xx x) (list y y yy)))
```

References

[Chvátal and Szemerédi,1988] V. Chvátal and E. Szemerédi. Many hard examples for resolution. *JACM*, 35:759–768, 1988.

[Crawford and Auton,1993] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 21–27, 1993.

[Crawford and Baker,1994] James M. Crawford and Andrew B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994. Submitted.

[Davis and Putnam,1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.

[de Kleer,1986] Johan de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[Ginsberg *et al.*,1990] Matthew L. Ginsberg, Michael Frank, Michael P. Halpin, and Mark C. Torrance. Search lessons learned from crossword puzzles. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 210–215, 1990.

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

[Kirkpatrick *et al.*,1982] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1982.

[Konolige,1994] Kurt Konolige. Easy to be hard: Difficult problems for greedy algorithms. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, Germany, 1994.

[McAllester,1993]
David A. McAllester. Partial order backtracking. `ftp.ai.mit.edu:/pub/dam/dynamic.ps`, 1993.

[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.

[Selman and Kautz,1993] Bart Selman and Henry Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 290–295, 1993.

[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.

[Selman *et al.*,1993] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *Proceedings 1993 DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

[Stallman and Sussman,1977] R. M. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196, 1977.