Combining Local Search and Backtracking Techniques for Constraint Satisfaction*

Jian Zhang† and **Hantao Zhang**
Department of Computer Science
The University of Iowa
Iowa City, Iowa 52242
{jizhang, hzhang}@cs.uiowa.edu

Abstract

Backtracking techniques are well-known traditional methods for solving many constraint satisfaction problems (CSPs), including the satisfiability (SAT) problem in the propositional logic. In recent years, it has been reported that local search techniques are very effective in solving some large-scale instances of the SAT problem. In this research, we combine the backtracking and local search techniques into a single method for solving SAT and CSPs. When setting a parameter of the method to either of its two extreme values, we obtain the ordinary backtracking procedure or the local search procedure. For some problems, if the parameter takes values in the middle of the two extremes, the new method is much more effective than either backtracking or local search. We tested the method with classical problems like the n-Queens and random SAT instances, as well as some difficult problems from finite mathematics. In particular, using the new method, we solved four open problems in design theory.

Introduction

Constraint satisfaction problems (CSPs) are very important in computer science and AI. Many practical problems (e.g., machine vision, planning and graph coloring) can be viewed as special cases of CSPs. In general, an instance of CSP consists of a set of variables, a finite domain of values for each variable, and a collection of constraints. A solution to a CSP is a full set of assignments (in which every variable is assigned a value) such that no constraint is violated.

The satisfiability problem (SAT) in propositional logic is a well-known special case of CSP where the variables are propositions whose values are boolean, and the constraints are often represented by a formula in conjunctive normal form (CNF) or a set of clauses. In this paper, we also study the satisfiability of first-order formulas in finite domains. These problems are also known as *model finding* or *model generation*, because a solution is called a model in logic.

Backtracking techniques are often used to solve finite domain CSPs, including the SAT problem. In this approach, one starts with an empty interpretation and repeatedly assigns a value to a new variable as long as the assignment does not violate the constraints; if a conflict occurs, it backtracks and assigns a different value to the variable. In this way, it explores the search space systematically and exhaustively. The performances of backtracking algorithms can be improved in a number of ways such as forward checking and lookahead (Kumar 1992).

Despite all the improvements on the backtracking techniques, many practical instances of CSP are still too hard for this approach. It has been known that local search methods have a great chance of success for a class of CSPs (Gu 1991; Minton et al. 1992). In contrast to the backtracking method, a local search starts with a random full interpretation and tests if the constraints are satisfied under this interpretation. If they are, the method stops with success; otherwise, the value of some variable is changed according to some criteria and then the process is repeated. There are also many reports that study local search techniques for the SAT problem (Selman, Levesque, & Mitchell 1992; Gu 1993; Gent & Walsh 1993).

Local search is very attractive when the search space of the problem is too large for a backtracking based procedure and the solution set to the problem is very dense (the n-queen problems and the randomly generated SAT problems are such problems). Although the method is incomplete, i.e., there is no guarantee that any solution will be found, it can indeed solve some large scale problems which are beyond the reach of complete search methods. However, this method is less successful for some structured problems whose

*This work is supported in part by the National Science Foundation under Grants CCR-9504205 and CCR-9357851.

† Permanent Address: Institute of Software, Academia Sinica, P.O.Box 8718, Beijing 100080, P.R.China

solution set is sparse. For instance, several backtracking based programs have been reported to solve open quasigroup problems (Slaney, Fujita, & Stickel 1995; Zhang & Stickel 1994; McCune 1994). However, the local search method has difficulty in solving these problems.

A primary motivation of our research is to solve open questions in mathematics. Of particular interest to us are the quasigroup problems whose solutions are eagerly sought by mathematicians (Bennett & Zhu 1992). Many of these problems are quite difficult and require very long time for complete search programs to solve. We find that it is often beneficial to combine the local search and backtracking search in the following way. We start the search procedure with a **partial** interpretation which can be obtained by hill climbing, and then use backtracking. If the search fails, another partial interpretation is chosen and the process repeats. The next partial interpretation is often obtained by modifying the previous one.

There are two extreme cases of the method. In the first case, the initial interpretation is empty, and the procedure is identical to backtracking search. In the second case, the initial interpretation is a set of assignments for each variable, and the procedure is similar to a local search. In general, this method is incomplete. That is, if it stops without finding a solution, we cannot say that the constraints are unsatisfiable. But unlike the greedy search and hill-climbing procedures, our method also relies on constraint propagation and backtracking.

In this paper, we describe the basic ideas of the new method and study it empirically. We shall report some experimental results on several different satisfiability problems. These results were obtained on a SGI workstation IRIX 5.3. We used the following programs: GSAT (Selman, Levesque, & Mitchell 1992), WSAT (Selman & Kautz 1993), SATO (Zhang & Stickel 1994) and SEM (Zhang & Zhang 1995). All of them are written in C. GSAT and WSAT are based on local search and random walk, while the other two programs use backtracking. GSAT, WSAT and SATO are for propositional logic; SEM is for first-order logic. SATO implements the Davis-Putnam algorithm (Davis & Putnam 1960), which consists mainly of unit propagation and case-splitting. SEM uses more sophisticated inference rules and a kind of forward checking. The new method was implemented by modifying these two programs. With the new approach, we solved four open cases of the quasigroup problems. To our knowledge, neither the backtracking based programs nor the local search based programs have been able to solve these problems.

Search for Finite Models: Basic Concepts and Quasigroup Examples

The model finding (or model generation) problem in first-order logic can be briefly described as follows. Given a set of sentences (or more simply, clauses), find a suitable interpretation of the function and predicate symbols in a given finite domain, such that all of the sentences hold. An example of model generation is to find quasigroups satisfying certain properties.

A *quasigroup* is an algebra (S, f), where S is a finite set and f is a binary operation whose multiplication table is a Latin square, i.e., each row and each column is a permutation of the elements in S. Quasigroups can be axiomatized by the following two clauses:

$$f(x,y) \neq f(x,z) \vee y = z,$$
$$f(x,z) \neq f(y,z) \vee x = y$$

where the variables x, y, z are assumed to be universally quantified over S. A quasigroup is *idempotent* if $f(x,x) = x$ for all x. Mathematicians are interested in quasigroups satisfying certain constraints. These constraints can be expressed by first order formulas, some of which are given below. For more about them, see (Bennett & Zhu 1992; Fujita, Slaney, and Bennett 1993).

Name	Constraint
QG2	$f(f(y,x),y) = f(f(z,x),z) \Rightarrow y = z$
QG3	$f(f(x,y), f(y,x)) = x$
QG4	$f(f(y,x), f(x,y)) = x$
QG50	$f(f(f(y,x), f(x, f(y,x))), y) = x$
QG6	$f(f(x,y),y) = f(x, f(x,y))$

Let us assume that a domain of size n is the set $D_n = \{ 0, 1, \ldots, n-1 \}$. The finite model finding problem can be formulated as a CSP. The variables of the CSP are the *cell variables* (i.e., ground terms like $f(0,0)$, $f(0,1)$, etc.) in the multiplication tables of the functions. The domain of each variable is D_n. As for the constraints, we substitute each variable in the clauses by the elements of D_n in all possible ways, and obtain a set of *ground* instances of the clauses (i.e. clauses that do not contain variables). The goal is to find a set of assignments (e.g., $f(0,1) = 2$) such that the given set of ground clauses hold. Note that the finite model generation problem in first-order logic can also be transformed into a propositional satisfiability problem (Kim & Zhang 1994; McCune 1994).

In the following, we shall not distinguish between SAT, CSP and finite model generation. They will be simply described as finding suitable values for a set of variables such that a given set of clauses are satisfied. This kind of problems are usually solved by a backtracking procedure, which begins with an empty set of

```
proc IC1($V$: variables; $C$: clauses;
        $MaxTries$, $MaxChg$: integer)
begin
  for $t := 1$ to $MaxTries$ do
    $I$ := random_interpretation($V$);
    for $k := 1$ to $MaxChg$ do
      if satisfiable($I$, $C$)
         then return success;
      $I$ := local_change($I$, $C$);
    end for
  end for
  return failure;
end proc
```

Figure 1: A greedy local search procedure

assignments and proceeds to a full set. At each step, it tries to find a value for a cell variable whose value is not yet known. After assigning some value to the cell variable, it propagates this assignment by logical reasoning. Such a procedure is complete but is computationally expensive. Usually only models of small size can be found within a reasonable amount of time. In many applications, we are only interested in the existence of an arbitrary model. In such cases, completeness may be sacrificed for efficiency. In the following we shall study incomplete search methods which may find large models more easily.

Local Search Procedures for Model Finding

Recently several authors studied the local search method for propositional satisfiability testing (Gu 1993; Selman, Levesque, & Mitchell 1992). The method is also known as hill climbing or iterative improvement or greedy search. It uses a "cost function" (or score function) to guide the search. The cost function is usually defined to be the number of clauses which are false under a given full interpretation of the variables. The search procedure starts with a random full interpretation, and repeatedly flips the truth value of some variable so that the value of the cost function decreases. When the cost becomes zero, the interpretation is a model of the clauses. It can also occur that the procedure is caught in a local minimum, in which case the cost is not zero, but it does not decrease when you change the truth value of any variable. Then you have to try another interpretation, and repeat the whole process.

The above idea can be readily used in first-order logic. The procedure in Figure 1 generalizes the GSAT procedure (Selman, Levesque, & Mitchell 1992).

The procedure `random_interpretation`(V) returns a random interpretation which assigns a value to every variable in V. The procedure `satisfiable`(I, C) tests if the clauses are satisfied under this interpretation.

The procedure `local_change`(I, C) computes the value of the cost function (i.e., the number of false clauses in C under the current interpretation) when one variable changes its value while the rest variables keep their old values. After this is done for every variable, it returns the interpretation whose associated cost is minimal.

Each try in `IC1` begins with a random set of assignments, and repeatedly improves it by changing the values of individual variables (i.e., by making local changes). The maximum number of changes in each try is given by the parameter *MaxChg*. A local change is allowed even when the number of satisfied clauses remains the same. That is, there can be "sideway moves".

We have experimented with the above procedure on various problems. In our experiments, not all the variables are involved in `random_interpretation`. Certain constraints enable us to fix the values of some variables. For example, when finding idempotent quasigroups (IQGs), we know that each cell variable $f(i,i)$ should take the value i.

The procedure `IC1` was implemented in SEM. It can solve the n-Queen and the IQG problems quite quickly; but it has difficulty in solving some other problems such as finding small finite groups and rings. We also translated some quasigroup problems into propositional clauses and used GSAT and WSAT to solve them. In general, WSAT is much more effective than GSAT, and can find IQGs very quickly. But it is not so easy for WSAT to find IQGs satisfying additional constraints. For example, the QG6.13 problem (i.e., finding a 13-element quasigroup satisfying QG6) is known to have many solutions (Slaney, Fujita, & Stickel 1995). It can be solved by SATO in a few minutes, but WSAT failed to find a solution after running for many hours (with different combinations of parameters used). We also tested WSAT on some smaller problems which have more solutions than other quasigroup problems[1]. They are very easy for SATO and SEM. On each problem, we let WSAT run 10 times. The following table gives the longest and shortest execution times.

[1]In our experiments, we did not use any special clause for eliminating isomorphism (Fujita, Slaney, and Bennett 1993). This is better for a local search procedure.

```
proc IC2($V$: variables; $C$: clauses;
        $MaxTries$, $MaxBr$, $MaxD$,
        $MaxFlips$, $MaxChg$: integer)
begin
  for $t := 1$ to $MaxTries$ do
    $I$ := init($V$, $MaxD$, $MaxFlips$);
    if $I$ = failure then goto the next try;
    for $k := 1$ to $MaxChg$ do
      if satisfiable2($I$, $C$, $MaxBr$)
         then return success;
      $I$ := local_change2($I$, $C$);
    end for
  end for
  return failure;
end proc
```

Figure 2: A two-phase incomplete search procedure

Problem	shortest	longest
QG2.7	696 sec.	3518 sec.
QG3.8	1 sec.	23 sec.
QG4.9	4 sec.	267 sec.

A New Incomplete Search Strategy

We think that, when finding finite models of first-order theories, logical reasoning (or constraint propagation) is very important. For example, if we have the symmetry axiom $f(x,y) = f(y,x)$, then the values of the cell variables $f(1,2)$ and $f(2,1)$ should remain the same during the search process. They should not be changed independently.

In this section, we describe a different incomplete search strategy for finding large finite models, which does not eliminate constraint propagation and backtracking. In our approach, the whole search process is divided into two phases. In phase one, we obtain a "good" partial solution, and then, in phase two, we try to extend it to a complete one by backtracking search. If this fails, we get another partial solution and try again. When the number of solutions is not too small, i.e., the constraints are not too hard to satisfy, it is quite likely that a complete solution is found after a few tries.

Our approach can be described as the general procedure IC2 in Figure 2. It looks quite similar to IC1 but acts very differently. In the following, we discuss the subroutines used in IC2.

init: The procedure init(V, $MaxD$, $MaxFlips$) tries to find an initial partial interpretation, i.e., a set of assignments to $MaxD$ variables in V, where $MaxD$ is a given non-negative integer. Various techniques such as local search and constraint propagation can be used in this procedure. Below we briefly describe several options for its implementation.

1. Randomly choose $MaxD$ variables and assign random values to them. When $MaxD = |V|$, this procedure is identical to random_interpretation(V) in IC1.
2. Randomly choose $MaxD$ variables and apply a greedy local search on them (rather than on the whole set of variables) to find a partial interpretation under which no clause in C is false. If no such interpretation can be found, the value *failure* is returned. In the greedy search, the "cost function" is the number of falsified clauses, and the maximum number of "flips" is given by $MaxFlips$.
3. Repeat option 2 several times to obtain a number of partial interpretations. Only one of the best partial interpretations will be recorded and returned, according to some criteria (e.g., number of constraints satisfied, number of assignments after the constraint propagation).
4. Assign an available value to one variable at a time, followed by constraint propagation, until a set of $MaxD$ variables are chosen. If a dead end occurs before $MaxD$ variables are assigned, the value *failure* is returned.

satisfiable2: This procedure decides the satisfiability of C, given a partial interpretation I. It uses backtracking and constraint propagation techniques to extend I into a full solution. The procedure returns *success* when such an interpretation is found. If $MaxD = 0$ (i.e., the input I is an empty interpretation), $MaxTries = MaxChg = 1$, and $MaxBr = \infty$, then IC2 is identical to a complete search procedure.

When option 4 is used for the implementation of init, the partial interpretation generated by init corresponds to a branch (from the root to an internal node) in the complete backtracking-based search tree. The procedure satisfiable2 then explores the subtree below this node. In the upper part of the tree above this node, we do not explore every branch. Instead, only some selected branches are considered.

Sometimes it may happen that the procedure satisfiable2 spends too much time to explore the search space. This may occur when the value of $MaxD$ is too small, or when one happens to choose a bad initial partial model. Indeed, one shortcoming of backtracking procedures is that they often get stuck in a large unsatisfiable region of the search space. One way to circumvent the problem is to impose a limit on the amount of time used in the second phase or on the size of the backtracking search tree. The parameter *MaxBr* to `satisfiable2` serves this purpose. *MaxBr* is the maximum number of branches that `satisfiable2` can explore in the backtracking search tree. If the procedure `satisfiable2` has explored *MaxBr* branches without finding a solution, it will terminate and IC2 continues to the next try. Of course, we may choose to ignore the parameter *MaxBr* by giving it a very big value.

local_change2: This procedure is similar to `local_change` in IC1. The first goal is to find a partial interpretation that minimizes the number of falsified clauses; the second goal is to find a partial interpretation that maximizes the number of true clauses. (Some clauses have no truth values.) When hill-climbing is also used in `init`, the second goal is the only goal of this procedure.

Experimental Results

We modified the program SEM (Zhang & Zhang 1995) so that it can do the incomplete search as described previously. Table 1 summarizes some experimental results on the following problems: Queens, idempotent quasigroups (IQG), groups, noncommutative groups (NCG) and rings[2]. The reader may refer to standard textbooks on abstract algebra for the axioms of these problems. In Table 1, *size* means the size of the model, *tries* and *time* refer to the number of tries and the elapsed time (in seconds) for finding the first model, respectively. A star ('*') indicates that a model was not found within 100 tries.

We can see that, in many cases, the new method can find large models quite easily.[3] In general, it takes much longer time for SEM, a very efficient backtracking model generator, to find one model of such large sizes. The last column of Table 1 gives the running times of the original program to find smaller models.

We also modified the program SATO and tested it with some random 3SAT problems. The running times on randomly generated formulas range from within a second to many hours. For a typical hard 3SAT formula that has 400 variables and 1700 clauses, using the exhaustive search, SATO finds the first solution in about 14 minutes. Using the fourth option ($MaxD$ = 5 and $MaxBr$ = 100000) to obtain a partial model,

[2]Among the four options for implementing `init`, the last one was used for these experimental results. The value of *MaxChg* is set to 1, and the procedure `satisfiable2` has a time limit of 1 minute.

[3]However, the new program does not work well on problems that have only a few solutions, like finding groups of order 17 or 19. In each case, there is essentially only one solution, i.e., the cyclic group.

problem	size	MaxD	tries	time	T0
Queen	8	5	3	1	.2
	12	5	1	2	2
	16	10	1	15	13
IQG	12	12	1	4	> 100
	20	45	1	.2	
	21	50	1	.3	
	22	50	1	.5	
Group	15	12	7	4	68
	16	12	12	10	37
	17	12	*		> 200
	18	12	9	10	
	19	15	*		
	20	15	5	3	
NCG	16	12	5	2	15
	18	12	12	18	8
	20	12	53	170	199
	24	12	5	50	
Ring	15	15	8	26	> 250
	18	15	4	31	
	20	15	8	74	

Table 1: Incomplete Search in SEM

SATO finds the first model in about 1 minute. Using the second option ($MaxD$ = 10) to obtain a partial model, the execution times range from several seconds to 2 minutes. The programs GSAT and WSAT are quite efficient on random 3SAT problems. For this particular formula, if we choose $MaxFlips$ = 4000, GSAT usually finds a solution in 3 or 4 seconds after about 10 tries. WSAT almost always finds a solution in the first try, within 1 second. However, there are also some randomly generated satisfiable formulas on which SATO performs better than GSAT.

Open Problems Solved

Some questions in mathematics need much computer time to solve, because the sizes of the models are so large. But it is not unusual that there are enough solutions for an incomplete search method to succeed. Indeed, we adopted such a strategy and solved several open questions in combinatorics. In (Fujita, Slaney, and Bennett 1993; Slaney, Fujita, and Stickel 1995), some open problems regarding the existence of quasigroups were reported to be solved. We continued to attack the remaining open problems and were able to solve some of them.

What we have been looking for are Latin squares with holes, or *holey* quasigroups (Bennett & Zhu 1992). Syntactically, such a quasigroup has a type, e.g., (h^n, k^1). This means, its size is $hn + k$, and it has

n holes of size h and one hole of size k. A hole is simply an empty subsquare, i.e., the cells in that area has no specific values. All holes are assumed to be disjoint.

Using our new incomplete model generation programs, we were able to find several previously unknown holey quasigroups, including QG2($1^{27}, 4^1$), QG4($2^5, 3^1$), QG4($1^{13}, 2^1$), and QG50($1^{21}, 5^1$). To the best of our knowledge, neither the backtracking based programs nor the local search based programs have been able to reproduce these results. There are still many open quasigroup problems. For instance, the existences of QG2($1^9, 1^1$), QG4($1^{17}, 2^1$), QG4($1^{21}, 2^1$), QG4($1^{25}, 2^1$), and QG50($1^{14}, 1^1$) are still unknown.

Concluding Remarks

Backtracking and local search are two different methods for constraint satisfaction and satisfiability testing. Each of them has some merits and shortcomings. The former explores the search space exhaustively and systematically. It is complete but cannot solve some large-scale problems in an acceptable amount of time. The latter works very well on some difficult random problems, but does not use constraint propagation[4].

In this paper, we propose a new approach which combines the benefits of the aforementioned two methods. To solve a given problem, we make several tries. Each try starts with a partial solution, and uses backtracking search to extend it to a complete solution. That is, each try jumps to an internal node of the search tree and explores exhaustively the search space below that node. Constraint propagation is involved in generating the initial partial solutions. And iterative improvement can be used in both phases of the search. We also put some limit on the backtracking search to avoid spending too much time in unsatisfiable regions. We believe that our approach is suitable for those problems whose search space is larger than a pure backtracking method can handle and whose solution set is not dense enough for a local search method to succeed.

We have implemented the new incomplete search strategy and conducted some preliminary experiments. The initial results are very encouraging. In particular, we are able to solve open problems that can not be solved by other programs. We plan to conduct more experiments in the future in order to have a better understanding of the new procedure.

[4]In the GSAT program, one may also use unit propagation in generating the initial assignment. But in general constraint propagation plays only a minor role.

References

Bennett, F.E., and Zhu, L. 1992. Conjugate-Orthogonal Latin Squares and Related Structures. in: J. Dinitz & D. Stinson (eds.), *Contemporary Design Theory: A Collection of Surveys*, John Wiley & Sons, 41–96.

Davis, M., and Putnam, H. 1960. A Computing Procedure for Quantification Theory. *J. of the ACM* 7(3): 201–215.

Fujita, M., Slaney, J., and Bennett, F.E. 1993. Automatic Generation of Some Results in Finite Algebra. *Proc. IJCAI-93*, 52–57.

Gent, I.P., and Walsh, T. 1993. Towards an Understanding of Hill-climbing Procedures for SAT. *Proc. AAAI-93*, 28–33.

Gu, J. 1991. 3,000,000 Queens in Less Than One Minute. *ACM SIGART Bulletin* 2(2): 22–24.

Gu, J. 1993. Local Search for Satisfiability (SAT) Problem. *IEEE Trans. on Systems, Man, and Cybernetics* 23(4): 1108–1129.

Kim, S., and Zhang, H. 1994. ModGen: Theorem Proving by Model Generation. *AAAI-94*, 162–167.

Kumar, V. 1992. Algorithms for constraint satisfaction problems: A survey. *AI Magazine* 13(1): 32–44.

McCune, W. 1994. A Davis-Putnam Program and Its Application to Finite First-order Model Search: Quasigroup Existence Problems. Technical Report ANL/MCS-TM-194, Argonne National Laboratory.

Minton, S., Johnston, M., Philips, A., and Laird, P. 1992. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58(1-3): 161–205.

Selman, B., Levesque, H., and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems, *Proc. AAAI-92*, 440–446.

Selman, B., and Kautz, H.A. 1993. Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. *Proc. IJCAI-93*, 290–295.

Slaney, J., Fujita, M., and Stickel, M. 1995. Automated Reasoning and Exhaustive Search: Quasigroup Existence Problems. *Computers and Mathematics with Applications* 29(2): 115–132.

Zhang, H., and Stickel, M. 1994. Implementing the Davis-Putnam Algorithm by Tries. Technical Report 94-12, Dept. of Computer Science, University of Iowa.

Zhang, J., and Zhang, H. 1995. SEM: a System for Enumerating Models. *Proc. IJCAI-95*, 298–303.