Stochastic Local Search for Incremental SAT

Malek Mouhoub
Department of Computer Science, University of Regina
3737 Wascana Parkway, Regina SK, Canada, S4S 0A2
phone : +1 (306) 585 4700 fax : +1 (306) 585 4745
email : mouhoubm@cs.uregina.ca

ABSTRACT
The boolean satisfiability problem (SAT) is stated as follows: given a boolean formula in CNF, find a truth assignment that satisfies its clauses. In this paper, we present a general framework based on stochastic local search and the structure of the CNF formula for solving incremental SAT problems. Given a satisfiable boolean formula in CNF, incremental SAT consists of checking whether the satisfiability is preserved when new clauses are added to the current clause set and if not, look for a new assignment that satisfies the old clauses and the new ones. The results of the experimentation we have conducted demonstrate the efficiency of our method to deal with large randomly generated incremental SAT problems.

KEY WORDS
Stochastic Local Search, GSAT, Dynamic Satisfiability.

1 Introduction

A boolean variable is a variable that can have one or two values: true or false. If x is a boolean variable, $\neg x$ is the negation of x. A literal is a boolean variable or its negation. A clause is a sequence of literals separated by the logical *or* operator ($\vee$). A logical expression in conjunctive normal form (CNF) is a sequence of clauses separated by the logical *and* operator ($\wedge$). For example, the following is a logical expression in CNF: $(x_1 \vee x_3) \wedge (\overline{x_1} \vee x_2) \wedge \overline{x_3}$.

The *CNF-Satisfiability Decision Problem* (called also SAT problem) is to determine, for a given logical expression in CNF, whether there is some truth assignment (set of assignment of true and false to the boolean variables) that makes the expression true. For example, the answer is "yes" for the above CNF expression since the truth assignment $\{x_1 = true\ ,\ x_2 = true\ ,\ x_3 = false\}$ makes the expression true.

SAT problems are fundamental to the theory of NP-completeness. Indeed, using the concept of "polynomial-time reducibility" all NP-complete problems can be reduced in polynomial time to any NP-complete problem including uniform mapping such as SAT [1]. This means that any new technique for SAT problems will lead to general approaches for many hard combinatorial problems.

One important issue when dealing with SAT problems is to be able to maintain the satisfiability of a propositional formula anytime a conjunction of new clauses is added. That is to check whether a solution to a SAT problem continues to be a solution anytime a set of new clauses is added and if not, whether the solution can be modified efficiently to satisfy the old formula and the new clauses.

In this paper we discuss the applicability of stochastic local search (SLS) algorithms for solving SAT problems in an incremental way. This choice is motivated by the fact that the underlying local search paradigms are well suited for recovering solutions after local changes (addition of constraints) of the problem occur. We use the SLS method within a general framework we propose for solving incremental SAT problems. This general framework takes advantage of the structure of the SAT formula in order to reduce the effort needed at each time to check the satisfiability of the formula anytime new clauses are added. Note that related work on solving SAT problems in an incremental way has already been reported in the literature [2, 3, 4, 5, 6]. These methods rely on stochastic local search [2, 3] or branch and bound [4, 5, 6] and handle the addition of one clause at a time. As we will see in the next Section, our method handles the addition of more than one clause at a time and depends on the structure of the CNF formula. Experimental comparison of our method with one of the well known methods proposed in the literature [4] has been carried out on randomly generated incremental SAT instances. The results favour our method in all the cases.

In the next section, we define the incremental SAT problem and present a general framework based on SLS for solving it. Section 3 is dedicated to the empirical experimentation evaluating our method. Concluding remarks and possible perspectives are finally presented in Section 4.

[1] We refer the reader to the paper published by Cook[1] proving that if CNF-Satisfiability is in P, then P = NP.

```
Procedure GSAT
begin
  for i ← 1 until MAX-TRIES do
  begin
    T ← a randomly generated truth assignment
    for j ← 1 until MAX-FLIPS do
      if T satisfies the formula then return T
      else make a flip
  end
  return ("no satisfying assignment found")
end
```

Figure 1: GSAT Procedure

2 Stochastic Local Search Based Framework for Solving incremental SAT

Before we define the incremental SAT problem and present the general method to solve it, let us introduce one of the well known SLS algorithms for solving SAT.

2.1 GSAT for SAT problems

One of the well known SLS algorithms for solving SAT problems is the GSAT procedure [7, 8] presented in figure 1. GSAT is a greedy based algorithm that starts with a random assignment of values to boolean variables. It then iterates by selecting at each step a variable, flips its value from false to true or true to false and records the decrease in the number of unsatisfied clauses. The algorithm stops and returns a solution if the number of unsatisfied clauses is equal to zero. After a given number of iterations, the algorithm updates the current solution to the new solution that has the largest decrease in unsatisfied clauses and starts flipping again until a solution satisfying all the clauses is found or a given number of tries is reached.

2.2 GSAT for Incremental SAT

We define dynamic SAT problem as a sequence of static SAT problems SAT_0, ..., SAT_i, SAT_{i+1}, ..., SAT_n each resulting from a change in the preceding one imposed by the "outside world". This change can either be a restriction (adding a new set of clauses) or a relaxation (removing a set of clauses because these later clauses are no longer interesting or because the current SAT has no solution). In this paper we focus only on restrictions.

Let us assume we have the following situation: $SAT_{i+1} = SAT_i \wedge NC$, where SAT_i is the current satisfiable SAT formula, NC is a new set of clauses to be added to SAT_i and SAT_{i+1} is the new formula obtained after adding NC. The goal here is to

check the consistency of SAT_{i+1} when adding the new set of clauses denoted by NC. This is done by performing the following steps.

1. If $x \wedge \overline{x}$ is contained in NC, return "NC is inconsistent". NC cannot be added to SAT_i.
2. Simplify NC by removing any clause containing a disjunction of the form x $\vee$ $\overline{x}$.
3. Let $NC = NC_1 \wedge NC_2$ where NC_1 is the set of clauses, each containing at least one variable that appears in SAT_i and NC_2 the set of clauses that do not contain any variable that appears in SAT_i or NC_1. SAT_i = $S_1 \wedge S_2$ where S_1 is the set of clauses, each containing at least one variable that appears in NC and S_2 the set of clauses that do not contain any variable that appears in NC or S_1.
4. Assign the truth assignment of SAT_i to NC_1. If NC_1 is satisfiable goto 7.
5. Using SLS flip the variables of NC_1 that do not appear in S_1. If NC_1 is satisfied goto 7. The search method starts from an initial configuration which satisfies SAT_i.
6. Using SLS look for a truth assignment that satisfies both S_1 and NC_1. If no such assignment is found return "NC cannot be added" as it will violate the consistency of SAT_i.
7. Using SLS look for a truth assignment for NC_2. If no such assignment is found return "NC cannot be added" as it will violate the satisfiability of SAT_i.

In step 7 GSAT is used as shown in figure 1. When used in step 5, the GSAT algorithm starts from the initial configuration (corresponding to the truth assignment found for the formula before adding the new clauses) instead of a random configuration. Also, only the variables belonging to NC_1 and which do not appear in S_1 can be chosen for the flip. In step 6, GSAT starts from the best configuration found in step 5. Also, all the configurations explored in step 5 are avoided in step 6.

3 Experimentation

In this section we present the experimental tests we have conducted in order to evaluate the performance of our method for solving incremental SAT problems. All tests are performed on a 2 GHz Pentium 4 computer under Linux. In each test, our incremental method (that we call incremental GSAT) is compared (in terms of running time in seconds) to the well known algorithm based on branch and bound for solving incremental SAT [4] (that we call incremental BB). All procedures are coded in C language.

3.1 Generation of Incremental SAT Instances

Since we did not find libraries providing dynamic SAT problems, we have generated incremental SAT instances from random SAT ones taken from the well known SATLIB library [2]. Each dynamic SAT instance is obtained from a SAT one in a series of stages. At each stage, a random number of clauses is taken from the SAT instance and added to the dynamic SAT one until there are no more clauses to take. In the following we consider st the total number of stages and N the total number of clauses of the SAT instance. The number of clauses $(N_1 \dots N_{st})$ taken at each stage are generated as follows. N_1 and N_2 are randomly chosen form $[1, N/5 - 1]$. N_3 and N_4 will then be generated from $[1, (N - N_1 - N_2)/4 - 1]$, and N_5 and N_6 from $[1, (N - N_1 - N_2 - N_3 - N_4)/3 - 1]$. $N_7, N_8, \dots, N_{st}$ will be generated in the same manner. This will ensure that the average number of clauses in each stage is almost equal to N/st. In the following tests, the value of st is fixed to 10.

Using the method above, we have generated incremental SAT instances from the following three types of SAT problems.

"Flat" Graph Coloring Problem. For a given graph, the "flat" graph coloring problem tries to color the nodes of the graph such that any pair of connected nodes have different colors. Here we focus on decision variant which consists of deciding whether for a particular number of colors, a coloring of the given graph exists. SAT instances corresponding to the "flat" coloring problem are generated as shown in [9].

Incremental "Morphed" Graph Coloring Problem. A particular class of graph coloring problems consists of morphing regular ring lattices with random graphs. $A(typeB)$ p-morph of two graphs $A = (V, E_1)$ and $B = (V, E_2)$ is a graph $C = (V, E)$ where E contains all the edges common to A and B, a fraction p of the edges from $E_1 - E_2$ (the remaining edges of A), and a fraction $1 - p$ of the edges from $E_2 - E_1$. More details regarding the generation of SAT instances for morphed graphs can be found in [10].

Random-3-SAT Instances with Controlled Backbone Size Problems. The backbone of a satisfiable SAT instance is the set of entailed literals. A literal l is entailed by a satisfiable SAT instance S if and only if S AND (NOT l) is unsatisfiable. The backbone size is the number of entailed literals. Random 3-SAT Instances with Controlled Backbone Size are generated as shown in [11].

In order to evaluate our method on hard incremental SAT problems, we have generated other incremental 3-SAT instances (from uniform random 3-SAT problems generated as shown in [12]) as follows. Since for 3-SAT problems the phase transition (corresponding to the hardest problems to solve) corresponds to a ratio $\frac{\#clauses}{\#variables} = 4.2$ we first split the random SAT instance into sets of clauses such that the addition of each set to the incremental SAT formula will always lead to a ratio $\frac{\#clauses}{\#variables}$ close to 4.2 . This way the formula to solve at each stage will be close to the phase transition.

Table 1: Comparative tests on randomly generated Incremental “Flat” Graph Coloring Problems.

testset	#variables	#clauses	Incremental GSAT	Incremental BB
flat30-60	90	300	0.00490	0.004
flat50-115	150	545	0.0153	0.09
flat75-180	225	840	0.113	0.85
flat100-239	300	1117	0.731	1.11
flat125-301	375	1403	3.829	13.2
flat150-360	450	1680	4.354	67.05

Table 2: Comparative tests on randomly generated Incremental “Morphed” Graph Coloring Problems.

testset	#variables	#clauses	Incremental GSAT	Incremental BB
sw100-8-lp0-c5	500	3100	0.012	0.010
sw100-8-lp1-c5	500	3100	0.021	0.27
sw100-8-lp2-c5	500	3100	0.0268	0.8
sw100-8-lp3-c5	500	3100	0.258	2.39
sw100-8-lp4-c5	500	3100	0.38	11.4
sw100-8-lp5-c5	500	3100	1.659	24.7
sw100-8-lp6-c5	500	3100	4.776	67.6
sw100-8-lp7-c5	500	3100	5.869	127
sw100-8-lp8-c5	500	3100	6.75	233
sw100-8-lp9-c5	500	3100	7.35	173

3.2 Test Results

The results of the tests performed on Incremental “Flat” Graph Coloring Problems, Incremental “Morphed” Graph Coloring Problems, and Incremental Random-3-SAT Instances with Controlled Backbone Size Problems are respectively reported in tables 1, 2 and 4. As we can see our incremental method is much faster than incremental BB for all problems which demonstrates the efficiency of our method over Hookers’ algorithm.

Table 3 presents comparative tests of our incremental method and incremental BB, for solving incremental hard uniform 3-SAT instances. The results obtained by our method are very appealing comparing to those of incremental BB. This is mainly due to steps 3 to 7 of our resolution method. Indeed, anytime a new set of clauses is added to a given formula in CNF, the goal of these steps is to maintain the satisfiability of the new formula by checking the satisfiability of the new set of clauses and only the clauses related to them from the initial set.

Table 3: Comparative tests on randomly generated Incremental Hard Uniform 3-SAT Problems.

testset	#variables	#clauses	Incremental GSAT	Incremental BB
uf50-218	50	218	0.00174	0.00464
uf75-325	75	325	0.00285	0.180
uf100-430	100	430	0.00476	0.273
uf125-538	125	538	0.00660	0.652
uf150-645	150	645	0.00705	4.021
uf175-753	175	753	0.0154	5.302
uf200-860	200	860	0.0191	2.901
uf225-960	225	960	0.0213	5.348
uf250-1065	250	1065	0.0211	14.194

4 Conclusion

In this paper we have presented a new method based on stochastic local search for maintaining the satisfiability of SAT problems in an incremental way. Our work is of interest to a large variety of combinatorial problems that need to be processed in an evolutive environment. This can be the case of applications such as reactive scheduling and planning, dynamic combinatorial optimization, dynamic constraint satisfaction and machine learning in an dynamic environment.

One perspective of our work is to deal with retraction of clauses in an efficient way. Assume that during the search, a given clause (or a set of clauses) is removed. Would it be worthwhile to reconsider any decision made because of these clause(s) or would it be more costly than just continuing on with search. Another idea we will investigate in order to improve the performance of our general procedure for solving incremental SAT consists of processing steps 4 until 6 and step 7 of our procedure in parallel. If any of these two parallel phases fails then the main procedure will stop and returns NC inconsistent. Finally our method can easily be extended to handle incremental MAX-SAT problems. The maximum satisfiability problem (MAX-SAT) consists of finding a truth assignment that satisfies the maximum possible number of clauses in a given SAT formula. The incremental MAX-SAT problem focuses on maintaining the maximum satisfiability of a propositional formula anytime a conjunction of new clauses is added. More precisely, this consists of checking whether the minimum number of violated clauses is preserved after new clauses are added otherwise we have to look for a new minimum. This can be achieved by performing the same method we have proposed in Section 2.2 with a slight modification of the GSAT algorithm in order to handle the maximum satisfiablity (instead of a complete satisfiability).

Table 4: Comparative tests on randomly generated Incremental 3-SAT Instances with Controlled Backbone.

testset	#variables	#clauses	Incremental GSAT	Incremental BB
CBS_k3_n100_m403_b10	100	403	0.0461	0.05
CBS_k3_n100_m403_b30	100	403	0.048	0.06
CBS_k3_n100_m403_b50	100	403	0.0694	0.9
CBS_k3_n100_m403_b70	100	403	0.181	1.2
CBS_k3_n100_m403_b90	100	403	0.706	1.8
CBS_k3_n100_m411_b10	100	411	0.0660	2.3
CBS_k3_n100_m411_b30	100	411	0.107	2.9
CBS_k3_n100_m411_b50	100	411	0.0717	3.2
CBS_k3_n100_m411_b70	100	411	0.148	4.6
CBS_k3_n100_m411_b90	100	411	0.663	5.8
CBS_k3_n100_m418_b10	100	418	0.0771	0.7
CBS_k3_n100_m418_b30	100	418	0.105	0.8
CBS_k3_n100_m418_b50	100	418	0.198	0.9
CBS_k3_n100_m418_b70	100	418	0.108	1.1
CBS_k3_n100_m418_b90	100	418	0.121	1.7
CBS_k3_n100_m423_b10	100	423	0.104	2.2
CBS_k3_n100_m423_b30	100	423	0.158	2.44
CBS_k3_n100_m423_b50	100	423	0.157	3.12
CBS_k3_n100_m423_b70	100	423	0.133	1.8
CBS_k3_n100_m423_b90	100	423	0.214	1.5
CBS_k3_n100_m429_b10	100	429	0.0628	3.4
CBS_k3_n100_m429_b30	100	429	0.0644	0.7
CBS_k3_n100_m429_b50	100	429	0.317	4.8
CBS_k3_n100_m429_b70	100	429	0.1800	3.6
CBS_k3_n100_m429_b90	100	429	0.188	3.5
CBS_k3_n100_m435_b10	100	435	0.0572	0.9
CBS_k3_n100_m435_b30	100	435	0.0947	1.1
CBS_k3_n100_m435_b50	100	435	0.049	0.8
CBS_k3_n100_m435_b70	100	435	0.150	1.45
CBS_k3_n100_m435_b90	100	435	0.193	3.78
CBS_k3_n100_m441_b10	100	441	0.1170	1.12
CBS_k3_n100_m441_b30	100	441	0.049	0.8
CBS_k3_n100_m441_b50	100	441	0.056	2.6
CBS_k3_n100_m441_b70	100	441	0.130	5.4
CBS_k3_n100_m441_b90	100	441	0.206	7.8
CBS_k3_n100_m449_b10	100	449	0.0323	11.2
CBS_k3_n100_m449_b30	100	449	0.0102	0.6
CBS_k3_n100_m449_b50	100	449	0.0422	0.5
CBS_k3_n100_m449_b70	100	449	0.037	0.6
CBS_k3_n100_m449_b90	100	449	0.347	2.12

References

[1] S. A. Cook. The complexity of theorem proving procedures. In *3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[2] H.H. Hoos and K O'Neil. Stochastic Local Search Methods for Dynamic SAT - an Initial Investigation. In *AAAI-2000 Workshop 'Leveraging Probability and Uncertainty in Computation*, pages 22–26, 2000.

[3] J. Gutierrez and A.D. Mali. Local search for incremental satisfiability. In *International Conference on Artificial Intelligence*, pages 986–991, 2002.

[4] J.N. Hooker. Solving the incremental satisfiability problem. *Journal of Logic Programming*, 15:177–186, 1993.

[5] H. Bennaceur, I. Gouachi, and Plateau. An incremental branch-and-bound method for satisfiability problem. *INFORMS Journal on Computing*, 10:301–308, 1998.

[6] J. Wittemore, J. Kim, and K.A. Sakallah. Satire: A new incremental satisfiability engine. In *DAC 2001*, pages 542–545, 2001.

[7] B. Selman and H. A. Kautz. An empirical study of greedy local search for satisfiability testing. In *AAAI'93*, pages 46–51, 1993.

[8] B. Selman, H. A. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In *AAAI'94*, pages 337–343. MIT Press, 1994.

[9] T. Hogg. Refining the phase transition in combinatorial search. *Artificial Inteligence*, 81:127–154, 1996.

[10] I. Gent, H. H. Hoos, P. Prosser, and T. Walsh. Combining structure and randomness. In *AAAI-99*, pages 654–660, 1999.

[11] J. Singer, I. Gent, and A. Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, pages 235–270, 2000.

[12] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *IJCAI-91*, pages 331–337, 1991.