

# Local Search for Incremental Satisfiability

Jos'e Gutierrez and Amol Dattatraya Mali,

Dept. of Electrical Engg. & Computer Science

University of Wisconsin, Milwaukee, WI 53211

Phone: 1-414-229-6762, Fax: 1-414-229-2769

jose@uwm.edu, mali@miller.cs.uwm.edu

Content Areas: Propositional Satisfiability, Local Search

## Abstract

(Appears in the proceedings of International Conference on Artificial Intelligence (IC-AI), Las Vegas, Volume 3, June 2002, pp. 986-992.)

Incremental satisfiability (ISAT) is the problem of testing if a propositional sentence ( $KB \wedge \alpha$ ) is satisfiable, given that the sentence  $KB$  is satisfiable, along with the truth assignment for the variables from  $KB$ . This problem is of practical relevance since it is often required to test whether a solution continues to be a solution when new constraints are added to the existing constraints and if not, whether the solution can be modified to satisfy both old and new constraints. Incremental satisfiability problem has been hitherto solved only by systematic search methods. Local search methods have significantly contributed to the enhancement of SAT solving procedures. We show how local search can be used for incremental satisfiability solving. Our experimental results show that many large instances of incremental satisfiability can be solved in a significantly short time. Our local search procedure integrates the ability of local search to explore search space with flexibility, with the recycling of given satisfying assignment to  $KB$ . Our evaluation shows that model of  $KB$  serves as a better starting assignment for local search for solving ( $KB \wedge \alpha$ ) than a randomly generated assignment to variables from ( $KB \wedge \alpha$ ). We discuss some other notions in propositional reasoning like DISTANCE-SAT, SAT solving by partitioning and supermodels and show how ISAT serves as a unifying framework for studying these problems.

## 1 Introduction

SAT (boolean satisfiability) is a highly widely studied combinatorial problem, where boolean variables are connected with logical operators like AND ( $\wedge$ ), OR ( $\vee$ ), implication ( $\Rightarrow$ ).  $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee x_4)$  is an instance of the SAT problem where  $x_1, x_2, x_3$  and  $x_4$  are the variables and  $(x_1 \vee \neg x_2)$  and  $(x_2 \vee x_3 \vee x_4)$  are two clauses.

All variables have the same domain  $\{true, false\}$ . The assignment  $x_1 = true, x_2 = true$  solves this SAT instance, irrespective of the values assigned to  $x_3$  and  $x_4$ . SAT is as expressive as a general constraint satisfaction problem.

Several tasks like logic circuit design and diagnosis, scheduling and planning have been converted into SAT and solved faster with the aid of faster SAT solvers [Crawford & Selman 1996; Crawford & Baker 1994; Walser et al 1998; Boyan & Moore 1998]. The role of satisfiability in electronic design automation has been highlighted in [Marques-Silva & Sakallah 2000]. One of the important challenge papers in the IJCAI-1997 conference [Selman et al 1997] is on propositional reasoning and many challenges in this paper invite the artificial intelligence community to solve larger SAT problems, especially using local search methods. The importance of SAT is also clear from several workshops held on it [Du et al 1997], along with the special issue SAT 2000, of the Journal of Automated Reasoning on the state of art in SAT solving at the beginning of year 2000.

Incremental satisfiability (ISAT) is the problem of checking if the sentence ( $KB \wedge \alpha$ ) is satisfiable (has a model), given that the propositional knowledge base  $KB$  is satisfiable, along with a truth assignment (model) which makes  $KB$  true, where  $\alpha$  is a sentence in propositional logic [Hooker 1993],[Kim et al 1999]. For example, let  $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_4) \wedge \neg x_3$  be a  $KB$  (has 5 variables and 4 clauses) to which  $\alpha$ , which let us say is  $(\neg x_4 \wedge (x_5 \vee x_6))$  is added. Let  $x_1 = true, x_2 = true, x_3 = false, x_4 = false$  and  $x_5 = true$  be the given model of  $KB$  as a part of the specification of the ISAT. With  $x_6 = true$ , ( $KB \wedge \alpha$ ) can be satisfied, without changing the given model of  $KB$ . If  $\alpha$  is  $x_4$ , then changing the model of  $KB$  with  $x_4 = true$  and  $x_5 = false$  makes ( $KB \wedge \alpha$ ) true, showing that the ISAT has a solution. Just as SAT is a prototypical constraint satisfaction problem (CSP), ISAT could be considered as a prototypical dynamic CSP where constraints and/or variables are added and/or replaced. Despite this, none of the papers from [Du et al 1997], an edited book on satisfiability testing, address ISAT.

There are several problems in practice where we are interested in testing if a solution to a CSP continues to be

a solution if new constraints are added and if not, how can the existing solution be changed to solve the new constraints along with the old ones. For example, plans generated for certain situations may fail if environments change. But these plans may not become useless, they could be modified to work in new situation. Being able to efficiently solve the incremental satisfiability problem also means making progress in solving such problems.

Given that the local search methods have been very successful in solving SAT problems, it is worth investigating if they can solve ISAT instances efficiently. We develop a local search procedure to solve ISAT and test it on instances generated by us as well as DIMACS benchmark problems. The solving times obtained are very encouraging. While solving problems like ISAT, it is often important to recycle solution (model of  $KB$ ), generally to reduce non-computational costs that arise due to addition of constraints like those in  $\alpha$ . For example, the cpu time and memory needed to find a plan represent computational cost and the amount of time or money in dollars needed to execute the plan represents non-computational cost. It is clear that a small increase in computational cost may be acceptable if the non-computational cost is going to be decreased significantly. Since local search methods are myopic and take arbitrary leaps in the search space, they do not look compatible with the idea of recycling. By arbitrary leaps in search space, we mean arbitrary changes to partial solutions. We show how local search methods can be integrated with the recycling of solutions. We also show how ISAT serves as a unifying framework for studying problems in propositional reasoning like DISTANCE-SAT, finding supermodels and solving SAT by decomposition.

The paper is organized as follows. In section 2, we explain key ideas in current SAT and ISAT solving procedures. In section 3, we report the local search procedure for solving ISAT. In section 4, we present empirical evaluation. In section 5, we discuss related work and future extensions to the ISAT solving procedure. Conclusions are reported in section 6.

## 2 Background

In this section, we first briefly survey various SAT solving procedures. Then we explain the ISAT solving procedure from [Hooker 1993].

Conventional approaches to SAT solving are based on assigning different values (true or false) to variables in a SAT instance and simplifying the instance based on the assigned values, until either a solution is found or lack of solution is detected. For example, if  $x_1 \wedge (\neg x_1 \vee x_2)$  is a given SAT instance, if  $x_1$  is assigned true value, the second clause simplifies to  $x_2$ .  $(x_1 \vee x_2) \wedge \neg x_1 \wedge \neg x_2$  is an instance with no solution. A truth assignment that does not satisfy all clauses is called a partial solution. In general, if  $F$  is the given SAT instance, the conventional SAT solving procedures choose a variable  $v$  and create two branches, one containing  $(F \wedge v)$  and the other containing  $(F \wedge \neg v)$  and try to satisfy these two

new instances. They repeat this procedure by choosing different variables and branching on them, generally creating a search tree of exponential size, until a solution is found or lack of solution is detected. The procedures in [Gallo & Urbani 1989; Park & Gelder 96] do this kind of systematic (global) search. Though recently developed systematic SAT solving algorithms [Bayardo & Schrag 1997],[Li & Anbulagan 1997],[LI 2000] have been more efficient, many problems are still solved faster than these by local search.

Local search has been used to solve SAT instances [Gu 1993]. GSAT [Selman et al 1992] is a local search driven solver which conducts hill climbing search (gradient descent). It stores only the current partial solution in its memory and improves it (thus doing iterative improvement). GSAT randomly assigns true or false values to variables and computes the number of clauses satisfied by this assignment. It then computes the change in the number of clauses satisfied by flipping the values assigned, by considering one flip at a time. It finally makes that flip which gives the maximum increase in the number of clauses satisfied (thus doing hill climbing). Though incomplete, it has solved many SAT problems orders of magnitude faster than the conventional procedures. Local search has been used to efficiently solve production planning problems [Walser et al 1998]. It has become possible to solve many SAT instances containing around  $10^4$  variables and  $10^6$  clauses in real time. Some problems with as large as  $10^5$  variables have also been efficiently solved and this is a major progress, as compared to 100 variable problems that could be solved in 1990 [Kautz 2000].

An approach based on global search to solve the ISAT problem has been proposed [Hooker 1993] where logical inference procedures are used to save work in deciding if  $(KB \wedge \alpha)$  can be solved, by exploiting the information that  $KB$  can be solved. If truth assignment that made  $KB$  true makes  $\alpha$  true, their procedure terminates, else it detects parts of the truth assignment of  $KB$  that make  $\alpha$  false and changes those parts and iterates.

## 3 Procedure for Solving ISAT

In this section, we give a high level explanation of the procedure, followed by detailed description.

The key idea in our local search procedure is to try to solve an ISAT problem by keeping the model of  $KB$  intact and using this model and assignments to variables that occur in  $\alpha$  only to satisfy clauses in  $\alpha$ . We change the model of  $KB$  only when  $(KB \wedge \alpha)$  cannot be solved after a certain number of iterations. The reason behind this is not just to save time but also recycle previous effort. In many cases, if  $(KB \wedge \alpha)$  is satisfiable, keeping the model of  $KB$  intact or changing it by a small amount (changing truth assignments of a small number of variables) will also mean reusing previously generated solution.

A clause from an ISAT problem can be classified into one of the 3 following categories - (i) containing only

those variables that occur only in  $KB$ , (ii) containing only those variables that occur only in  $\alpha$ , (iii) containing some variables that occur (a) only in  $KB$  or (b) only in  $\alpha$  and some more that do not fall in any of the two categories (a) and (b). In the ISAT solving procedure, we use this classification to satisfy  $(KB \wedge \alpha)$ , recycling solution of  $KB$  as much as possible. Maximum recycling of solution is not guaranteed. We initially assign values to variables that occur in  $\alpha$  only (step 5). Then we assign values to variables that occur in both  $KB$  and  $\alpha$  (thus taking freedom to change values of some variables from the model of  $KB$ ) in step 6. Then we consider all variables in  $(KB \wedge \alpha)$  and assign truth values to them (in step 7), thus taking freedom to change values of more variables from model of  $KB$  than in step 6. In different steps, we focus on different groups of variables to solve an ISAT problem.

Below we describe the local search procedure in detail.

0. Remove clauses containing pure literals.

1. If  $\alpha$  contains  $(x \wedge \neg x)$ ,  $x$  being a variable, return “unsatisfiable”. Remove clauses of the form  $(x \vee \neg x)$  since they are always satisfied.

2. If no variables are common between  $KB$  and  $\alpha$ , solve  $\alpha$  independently using hill climbing as in GSAT. If  $\alpha$  is satisfiable, return that  $(KB \wedge \alpha)$  is satisfiable, else return that no solution was found.

3. If some variables appear in both  $KB$  and  $\alpha$ , such that each clause in  $\alpha$  contains at least one such common variable, check if the truth assignment of  $KB$  also satisfies  $\alpha$ . If it does, return that  $(KB \wedge \alpha)$  is satisfiable.

4. Remove the set of clauses  $C$  from  $KB$  that are made true by truth assignments to variables that do not occur in  $\alpha$  or any other clause in  $KB$  outside  $C$  at all. For example, if  $KB$  is  $(x \vee y) \wedge (\neg y \vee z) \wedge (z \vee r \vee s) \wedge (p \vee q) \wedge (\neg q \vee t)$  and  $\alpha$  is  $(\neg x \vee y) \wedge (\neg z \vee \neg r) \wedge (w \vee m)$ , then the clauses  $(p \vee q) \wedge (\neg q \vee t)$  are removed from  $KB$  by this step.

5. Let  $\alpha'$  be the set of clauses that (i) contain no variables from  $KB$  or (ii) do contain variables from  $KB$  and also some variables that occur in  $\alpha$  only, such that these clauses are not satisfied by the truth assignment to variables in  $KB$ . Let  $A$  be the set of variables from  $\alpha'$  such that these variables occur in  $\alpha$  only.

(The motivation for having this step is to try to satisfy  $(KB \wedge \alpha)$  without disturbing the model of  $KB$ . If this fails, we change the model of  $KB$  in step 6).

Randomly assign truth values to variables in  $A$ .

For  $k_1$  iterations, where  $k_1$  is user chosen, do { Flip the value of a variable from  $A$  such that this flip gives the largest increase in the number of satisfied clauses from

$\alpha'$ . Store the truth assignment that lead to the largest increase in satisfied clauses and replace it by the one that gives a larger increase. If there is a local minimum, make a random truth assignment. If  $\alpha'$  is found to be satisfiable, check if the models of  $KB$  and  $\alpha'$  together make  $(KB \wedge \alpha)$  true. If that is the case, return that  $(KB \wedge \alpha)$  is satisfiable }.

(If  $\alpha$  contains a clause all variables of which appear in  $KB$  such that model of  $KB$  fails to satisfy this clause, then step 5 will not satisfy  $\alpha$  and thus also not satisfy  $(KB \wedge \alpha)$ . However this step is still included since it gives a truth assignment that satisfies several clauses from  $\alpha$ , so that we can start with this truth assignment and make some changes to model of  $KB$  in the following steps, to satisfy  $(KB \wedge \alpha)$ ).

6. Start with the truth assignment for  $\alpha'$  in step 5 that gave the largest increase in the number of satisfied clauses from  $\alpha'$  (in of course,  $k_1$  iterations).

(Execute this step 6 only if  $\alpha$  contains an unsatisfied clause which contains a variable that also appears in  $KB$ .) Let the set of variables that occur in both  $KB$  and  $\alpha$  and also in unsatisfied clauses of  $\alpha$  be  $B$ .

For  $k_2$  user chosen iterations, do { Flip the value of a variable from  $B$  such that this flip gives the largest increase from the current state (truth assignment) in the number of clauses from  $(KB \wedge \alpha)$  that are satisfied, without making any clause from  $KB$  false. (So we are flipping a common variable to maximize the number of satisfied clauses in  $\alpha$ , without making  $KB$  false. So we can change the model of  $KB$  here as long as no clause from it is made false.) Store the truth assignment that gives the largest increase and replace it by one that is better (so that this can be used as the starting point for the next step (7), in case solution is not found in this step).

If there is a local minimum, change the truth assignment randomly. If  $\alpha$  is satisfied, return that  $(KB \wedge \alpha)$  is satisfiable. }

7. Start with the truth assignment from step 6.

For  $k_3$  user chosen iterations, do { Flip the value of a variable that occurs in  $KB$  only, that leads to the largest increase in the number of satisfied clauses from  $(KB \wedge \alpha)$  (so more clauses from  $\alpha$  may be satisfied than in previous steps, making some clauses from  $KB$  false). If there is a local minimum, randomly change the truth assignment to variables from  $KB$ . If  $(KB \wedge \alpha)$  is satisfiable, return the truth assignment. }

(The part below solves  $(KB \wedge \alpha)$  like GSAT.)

For  $k_4$  user chosen iterations, do { Flip the value of a variable in  $(KB \wedge \alpha)$ , that leads to the largest increase in

the number of satisfied clauses from  $(KB \wedge \alpha)$ . If there is a local minimum, randomly change the truth assignment to variables from  $(KB \wedge \alpha)$ . If  $(KB \wedge \alpha)$  is satisfiable, return the truth assignment. }

8. If a satisfying truth assignment is obtained, incrementally change the assignment, considering one violation (deviation from model of  $KB$ ) at a time. If the change leaves  $(KB \wedge \alpha)$  satisfied, carry out the change to the truth assignment found.

(This step increases amount of recycling of the given model of  $KB$ . If the value of a variable in the model of  $KB$  is changed by any step before step 8, it is considered to be a violation or deviation from the model of  $KB$ . Step 8 removes these violations by dealing with one such variable at a time.)

9. Return “No satisfying assignment found”.

## 4 Empirical Evaluation

We generated several ISAT instances to evaluate the local search procedure. The results are shown in Fig. 1 and 2. The second and third columns in Fig. 1 and 2 contain the number of variables and clauses in  $(KB \wedge \alpha)$  respectively. The global search code [Hooker 1993] could not be made available for a comparison of its performance with our local search-based procedure (the author does not have a copy of the code anymore). The experiments were ran on dual Intel pentium II 400 MHz Sun OS 5.7 machine. The reported times are cpu seconds. We used some benchmark problems from the library of such problems at <http://aida.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html>. Though several SAT challenge problems are available, no such ISAT problems are available. Thus we created ISAT problems in the following manner. After taking a satisfiable SAT instance  $S$ , we asked a user to provide a percentage value  $x$ . This may be percentage of common number of variables between  $KB$  and  $\alpha$  or the percentage of clauses in the satisfiable instance that must occur in  $KB$ . In the experiments in Fig. 1, first  $x$  percent clauses from a satisfiable SAT instance  $S$  were put into  $KB$  and remaining  $(1 - x)$  percent clauses from the SAT instance were put into  $\alpha$ . This style of creating ISAT instances has the advantage that one can feed  $S$  to a complete SAT solver and make sure that  $S$  is satisfiable. Then it is guaranteed that irrespective of the value of  $x$  chosen, the resulting ISAT has a solution. The fourth column in Fig. 1 contains the percentage values used to split satisfiable SAT instances into  $KB$  and  $\alpha$ , so that first  $x$  percent clauses from  $S$  appear in  $KB$  and the remaining clauses from  $S$  appear in  $\alpha$ . In Fig. 2, the percentage denotes the percentage of variables common between  $KB$  and  $\alpha$ . This is found by using the ratio of the number of common variables and the total number of variables in  $(KB \wedge \alpha)$ .

The problems aim1, aim2, aim3, aim4, aim5, aim6, aim7 and aim8 are respectively aim-100-1.6-yes1-1, aim-100-2.0-yes1-1, aim-100-3.4-yes1-1, aim-100-6.0-yes1-1, aim-200-1.6-yes1-1, aim-200-2.0-yes1-1, aim-200-3.4-yes1-1, and, aim-200-6.0-yes1-1 from the DIMACS collection. The problems flat-1, flat2 and flat-3 are respectively the problems flat30-1, flat50-1 and flat75-1 from the DIMACS collection.

The first 3 problems in Fig. 1 are random 3-SAT problems. 3-SAT has been of greater interest because of the phase transition phenomenon in which randomly generated 3-SAT instances for which the the ratio of the number of clauses and variables is around 4.19 are hard to solve [Crawford & Auton 1993], [Crawford & Selman 1996]. For the first 2 problems, this ratio is 4.36 and for the third problem, this is 4.33, showing that these problems are such hard problems. The fourth problem is a combinatorially hard problem of graph coloring from graph theory. The last two problems in Fig. 1 were generated by us. In these two problems, length varied from clause to clause. These problems were verified to have solution by solving them with the systematic SAT solver satz [Li & Anbulagan 1997].

The highly efficient local search-based SAT solver walksat could not solve any of the problems from Fig. 2 within 4 hours, the minimum time cutoff we used. In some cases, walksat was terminated after 6, 8, 24 and 48 hours. walksat starts from a randomly generated initial assignment to variables from  $(KB \wedge \alpha)$ . Our results show that starting from an assignment that satisfies  $KB$  significantly helps local search in solving  $(KB \wedge \alpha)$ . Step 8 plays an important role in repairing solutions found, to increase the recycling of the given model of  $KB$ .

## 5 Discussion

We showed how local search can be used to solve ISAT and provided its empirical evaluation. In this section, we discuss connections of ISAT with some other problems in propositional reasoning, followed by a future extension to our approach.

[Bailleaux & Marquis 1999] define DISTANCE-SAT and provide algorithms for solving it. This is the problem of finding if there is model of a formula  $F$  that disagrees with an expected configuration on at the most  $d$  variables. This is related to ISAT in the following manner.  $F$  in DISTANCE-SAT can be considered as  $(KB \wedge \alpha)$  in ISAT. Part or whole of the model of  $KB$  can be considered to be an expected configuration, if it is desired that solution of  $KB$  should be completely or partly recycled. If one is interested in 100 % recycling of model of  $KB$ , this model could be ANDed with  $\alpha$  and the resulting instance could be passed to a SAT solver. If the instance is satisfiable, 100 % recycling is possible. For example, if  $x = true, y = true, z = false, d = false$  is model of  $KB$ ,  $(x \wedge y \wedge \neg z \wedge \neg d \wedge \alpha)$  could be passed to a SAT solver. If this instance is satisfiable, model of  $F$  that disagrees with model of  $KB$  on zero variables exists. Solution of  $(x \wedge y \wedge z \wedge d \wedge \alpha)$  disagrees with model of  $KB$  on 2 vari-

Problem	# Variables	# Clauses	%	Time (secs)
Random 3-SAT	50	218	50	48
Random 3-SAT	50	218	20	73.29
Random 3-SAT	75	325	50	2369.89
Graph Coloring	600	2237	35	1.9
Random	5000	250000	50	7.10
Random	10000	500000	2	5.23

Figure 1: Performance of the local search procedure on ISAT instances generated using clause-based split.

Problem	# Variables	# Clauses	%	Time (secs)
aim1	100	160	40	0
aim1	100	160	50	2156
aim2	100	200	30	0
aim3	100	340	40	3
aim4	100	600	10	14
aim4	100	600	30	0
aim5	200	320	40	0
aim6	200	400	30	0
aim7	200	680	50	1316
aim8	200	1200	30	1126
aim8	200	1200	40	1098
aim8	200	1200	80	0
flat1	90	300	5	0
flat2	150	545	20	1
flat2	150	545	30	2
flat3	225	840	5	0

Figure 2: Performance of the local search procedure on ISAT instances generated using common variables-based split.

ables  $(z, d)$ . If  $(x \wedge y \wedge z \wedge d \wedge \alpha)$  is satisfiable, model of  $F$  that disagrees with expected configuration on 2 variables exists. In general, negating the truth assignments of  $d$  variables from model of  $KB$  and showing that the conjunction of these unit clauses and  $\alpha$  is satisfiable, is like showing that a model of  $F$  that disagrees with expected configuration (model of  $KB$ ) on at the most  $d$  variables exists.

The main difference between DISTANCE-SAT and ISAT is that in DISTANCE-SAT, no new clauses are added to the problem. DISTANCE-SAT can be considered as a special case of ISAT where (i)  $\alpha$  is either a tautology or a clause from  $KB$  itself and (ii)  $KB$  is not solved and its model is not available and (iii) what a user thinks should be part of a model of  $KB$  is available (this is expected configuration). We do want to mention that the algorithm for solving DISTANCE-SAT [Baillaux & Marquis 1999] is systematic.

Changing solutions with a bound on the change (solution recycling), to satisfy newly introduced constraints, without violating old constraints, has been addressed in [Ginsberg et al 1998], in the form of robust solutions also known as “supermodels”. Supermodels are solutions that can satisfy more new constraints with fewer changes. In the context of ISAT, models of  $KB$  that do not need lot of change, to satisfy a large number of instances of form  $(KB \wedge \alpha)$ , can be considered as super-

models.

Solving a SAT instance by decomposition can be considered as solving multiple ISAT instances where there are multiple  $KB$ s and  $\alpha$ s and the number of  $KB$ s and  $\alpha$ s changes online. The main difference between ISAT and solving SAT by decomposition is that model of  $KB$  is given in ISAT and this model is found online, in solving SAT by decomposition. For example, if a set of satisfiable clauses  $S$  is split into  $S_1, S_2, S_3$  and  $S_4$ , solutions of  $S_1, S_2$  could be viewed as solutions of  $KB_1$  and  $KB_2$  that can be extended or revised to respectively solve  $(KB_1 \wedge S_3)$  and  $(KB_2 \wedge S_4)$  and so on. Here  $KB_1$  and  $KB_2$  are same as  $S_1$  and  $S_2$  respectively. Let us denote  $(KB_1 \wedge S_3)$  by  $KB'$ . Let us denote  $(KB_2 \wedge S_4)$  by  $\alpha'$ . Then  $S$  can be shown to be satisfiable by using an ISAT solver with solution of  $KB'$  and  $(KB' \wedge \alpha')$  as its inputs. Thus ISAT serves as a unifying framework for studying problems of DISTANCE-SAT, finding supermodels and solving SAT by decomposition.

In dynamic environments, constraints may be added to or removed from an existing CSP. A solution of CSP remains a solution when constraints are removed. Thus solution will have to be changed only when constraints are added. ISAT can thus be studied as a prototypical dynamic CSP.

There are some useful directions in which our local search procedure can be extended to more smoothly integrate hill climbing and solution recycling. Instead of

going through various well separated phases like changing the truth of variables in  $\alpha$  only, then the common variables, then the variables that appear in  $KB$  only, and then any variable from  $(KB \wedge \alpha)$ , one can modify GSAT so that it uses a different evaluation function which incorporates both the idea of maximizing the increase in the number of satisfied clauses and the idea of making as few changes to the given model of  $KB$  as possible. Thus instead of flipping the value of a variable such that this flip gives more increase in the number of satisfied clauses than any other flip, one can carry out that flip for which the ratio of increase in the number of satisfied clauses and the number of changes in the model of  $KB$  is maximum. Functions other than this ratio could also be defined.

## 6 Conclusion

ISAT can be studied as a prototypical dynamic CSP where constraints are added and/or replaced. ISAT also serves as a unifying framework for studying problems like DISTANCE-SAT, finding supermodels and solving SAT by decomposition. To exploit the capability of local search in solving SAT efficiently and to recycle solutions, we developed a local search procedure which starts with more restrictions on the leaps in search space and gradually removes these restrictions, incrementally getting more flexibility in changing truth assignments. We showed that the potential of local search in solving challenging SAT instances can be exploited in solving ISAT as well, with proper modifications to handle the model of  $KB$ . The procedure could efficiently solve a variety of ISAT instances, including several benchmark problems. Our results show that model of  $KB$  serves as a better starting assignment for local search for solving  $(KB \wedge \alpha)$  than a randomly generated assignment to variables from  $(KB \wedge \alpha)$ . Repairing solution found by local search to reduce its deviation from model of  $KB$  a posteriori can be effective in recycling the model. In practice, this can significantly reduce non-computational costs.

**Acknowledgement:** This work is supported in part by NSF grant IIS-0119630 to Amol Mali.

## References

[**Bailleux & Marquis 1999**] Olivier Bailleux and Pierre Marquis, DISTANCE-SAT: Complexity and algorithms, *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pp. 642-647, 1999.

[**Bayardo & Schrag 1997**] Roberto J. Bayardo Jr. and R. C. Schrag, Using CSP look-back techniques to solve real world SAT instances, *Proceedings of the National Conference on Artificial Intelligence*, pp. 203-208, 1997.

[**Boyan & Moore 1998**] Justin A. Boyan and Andrew W. Moore, Learning evaluation functions for global optimization and boolean satisfiability, *Proceedings of*

*National Conference on Artificial Intelligence (AAAI)*, 1998, pp. 3-10.

[**Crawford & Auton 1993**] James M. Crawford and Larry D. Auton, Experimental results on the crossover point in satisfiability problems, *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pp. 21-27, 1993.

[**Crawford & Baker 1994**] James M. Crawford and Andrew B. Baker, Experimental results on the application of satisfiability algorithms to scheduling problems, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1994, pp. 1092-1097.

[**Crawford & Selman 1996**] James Crawford and Bart Selman, New methods for solving large constraint and reasoning problems, *Tutorial presented at the National Conference on Artificial Intelligence (AAAI)*, 1996.

[**Du et al 1997**] Dingzhu Du, Jun Gu and Panos M. Pardalos, Editors, *DIMACS Volume 35: Satisfiability problem - Theory and applications*, American Mathematical Society, 1997.

[**Frank 1997**] Jeremy Frank, Learning short-term weights for GSAT, *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 1997, pp. 384-389.

[**Gallo & Urbani 1989**] Giorgio Gallo and Giampaolo Urbani, Algorithms for testing the satisfiability of propositional formulae, *Journal of Logic Programming*, 1989:7, pp. 45-61.

[**Ginsberg et al 98**] Matthew L. Ginsberg, Andrew J. Parkes and Amitabha Roy, Supermodels and robustness, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1998, pp. 334-339.

[**Gu 1993**] J. Gu, Local search for satisfiability (SAT) problem, *IEEE Transactions on systems, man and cybernetics*, 23(4): 1108-1129, July 1993.

[**Gu et al 1997**] Jun Gu, Paul W. Purdom, John Franco and Benjamin W. Wah, Algorithms for the satisfiability (SAT) problem: A survey, *Appears in [Du et al 1997]*, pp. 19-151.

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

[**Kautz 2000**] Henry Kautz, Satisfiability and State Transition Systems: An AI Perspective, **Invited talk at Conference on Automated Deduction (CADE)**, 2000.

[**Kim et al 1999**] Joonyoung Kim, Jesse Whittemore,

Joao Marques-Silva and Karem A. Sakallah, Incremental boolean satisfiability and its application to delay fault testing, IEEE/ACM international workshop on logic synthesis (IWLS), June 1999.

[**Li & Anbulagan 1997**] Chu Min Li and Anbulagan, Heuristics based on unit propagation for satisfiability problems, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1997.

[**LI 2000**] Chu Min LI, Equivalency reasoning to solve a class of hard SAT problems, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2000.

[**Marques-Silva & Sakallah 2000**] Joao P. Marques-Silva and Karem A. Sakallah, Boolean satisfiability in electronic design automation, Proceedings of the IEEE/ACM Design Automation Conference (DAC), June 2000.

[**Park & Gelder 1996**] Tai Joon Park and Allen Van Gelder, Partitioning methods for satisfiability testing on large formulas, *Proceedings of Conference on Automated Deduction (CADE)*, 1996.

[**Selman et al 1992**] Bart Selman, David Mitchell and Hector Levesque, A new approach to solving hard satisfiability problems, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1992.

[**Selman et al 1997**] Bart Selman, Henry Kautz and David McAllester, Ten challenges in propositional reasoning and search, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.

[**Walser et al 1998**] Joachim P. Walser, Ramesh Iyer and Narayan Venkatasubramanian, An integer local search method with application to capacitated production planning, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1998, pp. 373-379.