Applying Strong Components Detection to Max3SAT

DAVID PINTO, ALBINO PETLACALCO, ANDRÉS VÁZQUEZ
Faculty of Computer Science
Benemérita Universidad Autónoma de Puebla,
Edificio 135, Ciudad Universitaria, San Manuel, CP 72570
PUEBLA, PUE. MEXICO.
{dpinto, petlacalco, andrex}@cs.buap.mx

Abstract: - The problem of maximum satisfiability is an NP-Complete problem, which means that a deterministic algorithm for solving it in polinomial time does not exist unless P=NP. New methods that find solutions that approximate optimal solutions are requested. We have developed a heuristic based on strong components detection, to approximate Max3SAT, that improves the time execution by eliminating leaves nodes. We have obtained good results even when the size of the problem increases.

Key-Words: - Strong Components, MaxSAT, NP-Problem.

1 Introduction

The problem of maximum satisfiability [1] can be used for many kinds of problems. In the particular case of verification of consistency [5] in databases, the information stored is usually expressed by implications, which can then be transformed into formulas. In this way, if the database is inconsistent, we want to know the minimum number of clauses to be eliminated in order to recover the consistency. The wide range of applications of MaxSAT and the fact that this is a central problem of complexity theory, makes it an important problem to study.

Given a formula F in conjuntive normal form (CNF), the satisfiability problem consists of finding an assignment that satisfies the formula or indicates if such an assignment does not exist. When we cannot find any assignment that satisfies the formula F, we want to find an assignment such as maximizing the number of clauses in F; this variant of the satisfiability problem is known as *the maximum satisfiability problem* (MaxSAT). Max3SAT is a particular instance of MaxSAT, in that it is a problem based on MaxSAT but only with three literals in each clause of the formula. It has been shown that Max3SAT is equivalent to MaxSAT [6].

Several algorithms have been proposed to solve SAT and MaxSAT [2], [3]. In this paper we present a new heuristic to approximate Max3SAT. The method that we have developed is based on the detection of strong components [7]. Given an instance F of 3SAT in 3CNF, we obtain an associated graph $G(F)$ of it, and a strong component without inconsistency, i.e., there is not an arc from x to $\sim x$ in G(F) for each node in the component; in order to do this, we used additional polinomial reductions.

We have tested the heuristic on large instances, with 100, 300, 500 variables and 200-500, 300-1200, 1000-2000 clauses respectively and we have obtained good solutions. These sizes have been considered because, according to Selman, in these ratios we can find the most difficult instances of SAT [4].

This paper has been structured as follows: In Section 2 we explain the concepts used in the heuristic. In Section 3 we present how the heuristic works. In Section 4 we present the results we obtained. Finally, in Section 5 we present our conclusions.

2 Background

In order to implement a heuristic for Max3SAT, we need some basic principles which help us reach our goals. A brief explanation is provided regarding the reductions that are used, strong components detection and an important theorem to guarantee satisfiability.

These reductions are used to transform a CNF formula to a graph. Since we used a basic logical theorem to reach the transformation (related with the implication logical operator), it is very important to introduce a section that allows us understand how to translate a 3CNF formula to a 2CNF one. In the next subsection we introduce this subject. The subsection subsequent explains the logical theorem mentioned below. A brief explanation of strong components detection is given at subsection 2.3, and in subsection 2.4 we link all of the concepts to provide a sense of meaning to strong components with satisfiability of the formula.

2.1 3CNF to 2CNF Reduction

Let F be a formula in CNF. Consider the next polinomial reduction from 3CNF to 2CNF [8]:

Let $\{x_1, \ldots, x_n\}$ be a set of variables and F have the form $f_i \wedge f_2 \wedge \ldots \wedge f_m$ where $f_i = x_{j1} \vee x_{j2} \vee x_{j3}$, with $i = 1 , \ldots , m$ and $j = 1 , \ldots, n$.

In addition, every f_i in F is replaced with g_i as follows (Note that the variable t_i is added):

$$g_i = x_{j1} \wedge x_{j2} \wedge x_{j3} \wedge t_i \wedge (\sim x_{j1} \vee \sim x_{j2}) \wedge (\sim x_{j2} \vee \sim x_{j3}) \wedge (\sim x_{j3} \vee \sim x_{j1}) \wedge (x_{j1} \vee \sim t_i) \wedge (x_{j2} \vee \sim t_i) \wedge (x_{j3} \vee \sim t_i)$$

It is clear from the formulation below that the formula F is insatisfiable and that the maximum number of clauses that can be satisfied are $7m$.

Although Papadimitriou proposed this reduction, we have observed that the added artificial variable can be eliminated from the transformation from 2CNF. We can then graph the algorithm by increasing its speed but without decreasing the quality of the results.

The resolutions of 2CNF and 3CNF are equivalent, however, the Max2SAT problem stills lies in NP [9].

2.2 2CNF to Graph Reduction

Another reduction [8] that we have to consider is the following one:

Given a formula F, we construct a graph $G(F)$ where there is an arc from node x to node y iff there is a clause $(\sim x \vee y)$ or $(y \vee \sim x)$ in F. In addition if there is an arc from x to y, there is an arc from $\sim x$ to $\sim y$ too.

The reduction below is useful because of its logical implications regarding its use in obtaining a graph associated with F after having made the reduction from 3CNF to 2CNF.

2.3 Strong Components Detection

Based on the reduction below, we used the strong components detection to find a path in the graph $G(F)$. This path does not allow inconsistencies, i.e., we do not add the node x in the strong component if earlier we have added the node ~x. We used the algorithms proposed by Nuutila [7] to perform it. Nuutila proposed several algorithms that avoid redundant computations in the graph.

The main idea of the algorithm of strong components detection in a graph $G(F)$ is to define the variables successor and root for each node in G. For each node x in G, we perform the transitive closure of x if x has not been visited yet. At the beginning if the variable root of x is x itself, a function $min(x\ ;\ y)$ is then defined to determine which node has been entered first into the method. Then we assign the node that was entered first to the variable root. Furthermore for each arc in x we perform the transitive closure, and the set of successor nodes of x is stored in the variable successor of the node root of x. In this way we avoid data replication, thereby increasing the speed of the algorithm.

2.4 Satisfiability in a Graph

We also used the theorem shown at [8], known as ”Papadimitriou’s reduction”, to guarantee the satisfiability of F, given the graph that represents it.

Theorem: F is insatisfiable if a node x in $G(F)$ exists such that we can find a path from x to $\sim x$ and from $\sim x$ to x.

We also can build the inverse relation: first we consider the node x and assign to it the value *true* if we have not yet assigned any value in previous steps. Then we consider the nodes that can be raised from x, and assign the *true* value too. In addition, we assign the value *false* to the negations of those nodes. We repeat the step below until all the nodes have some *true* value. Eventually each node in G will have a *true* value and there will be not path from x to $\sim x$ in $G(F)$. The succession of nodes that have each the value *true* will satisfy the formula F.

3 The Algorithm

With the definitions and reductions described below, we developed a heuristic that takes a formula F in 3CNF as input and transforms it into a 2CNF. Following, a transformation from 2CNF to a graph $G(F)$ is applied. We then detect the strong components in $G(F)$ and determine the component that maximizes the number of clauses satisfied in F.

The process below generates $2m + 2n$ nodes for a problem with n variables and m clauses according to the reductions that we have mentioned, this because for each clause c_i the reduction adds a new variable t_i. In order to improve the performance of the algorithm and reduce the representation of the formula, we have introduced some changes. We do not add the nodes t_i and $\sim t_i$ in G. The nodes t_i and $\sim t_i$ are leaves nodes and they do not have any arcs to other nodes. This process reduces the size of the representation of the problem from $2m + 2n$ to $2n$.

We implemented both algorithms, by both adding and writing the artificial variable. We named RA the first and RB the second and the results obtained are shown in the next section.

4 Results

We used the Kullmann's package [10] to generate 3SAT instances. According to Selman's empirical analysis [4] we generated the following groups of instances: for 100 variables, instances with 200, 300, 400 and 500 clauses, for 300 variables instances with 600, 900, 1200 clauses and for 500 variables instances with 1000, 1500, 2000 clauses. The results that we report in this paper are the average result of these instances.

According to the formulation of the heuristic we explained in the section above, we tested both heuristics, *RA* and *RB*. We both included and omitted the variable t_i using the instances mentioned above and obtained the following results:

In Table 1 we show the time average in minutes of the heuristics with 100 variables. The difference in time is remarkable; *RB* obtains results of the same instances in seconds while *RA* takes minutes.

TABLE 1
RUNNING TIME AVERAGE - 100 VARIABLES

Clauses	RA	RB
200	1.63	0.01
300	4.69	0.02
400	9.46	0.03
500	22.31	0.03

In Tables 2, 4 and 6 we show that the results for both algorithms *RA* and *RB* are the same, giving evidence that the improvement of the algorithm seems to be correct. Future work should verify this hypothesis.

TABLE 2
SATISFIED CLAUSES PERCENT - 100 VARIABLES

Clauses	RA	RB
200	99.1	99.1
300	98.27	98.27
400	97.63	97.63
500	96.72	96.72

In Figure 1 we show a graphical view of the time differences between both algorithms with instances of 100 variables.

In Table 3 we show the time average in minutes of the heuristics with 300 variables. The difference in time, as with 100 variables, is notable; *RB* also obtains results of the same instances in seconds while *RA* takes minutes.

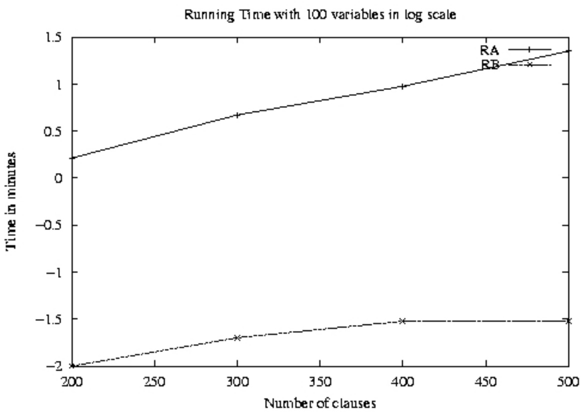

Fig. 1

In Figure 2 we show a graphical view of the percentage of satisfied clauses of both algorithms with instances of 100 variables.

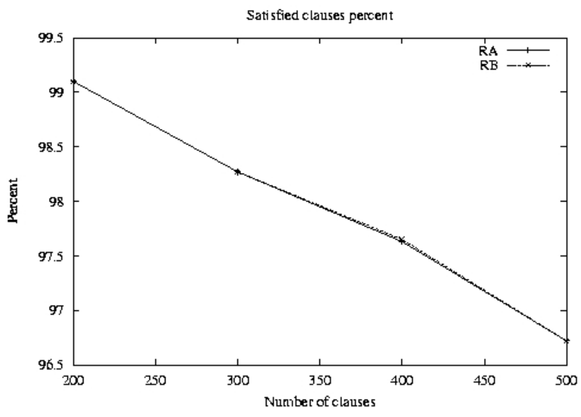

Fig. 2 - 100 variables

In Figure 3 we show a graphical view of the time differences between both algorithms with instances of 300 variables.

TABLE 3
RUNNING TIME AVERAGE - 300 VARIABLES

Clauses	RA	RB
600	48.4	0.54
900	98.37	1.76
1200	476.84	2.74

TABLE 4
SATISFIED CLAUSES PERCENT- 300 VARIABLES

Clauses	RA	RB
600	98.95	98.95
900	98.37	98.37
1200	97.67	97.67

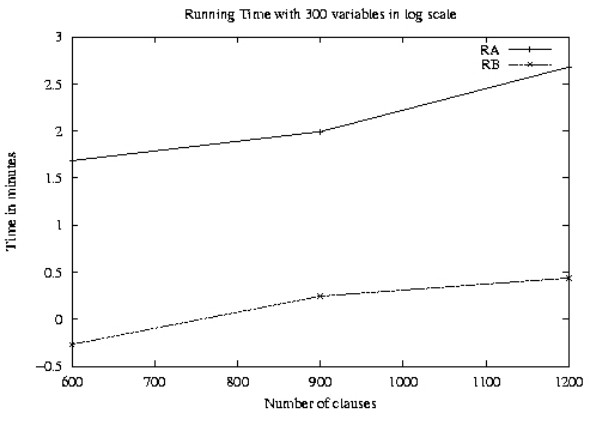

Fig. 3

In Figure 4 we show a graphical view of the percentage of satisfied clauses of both algorithms with instances of 300 variables.

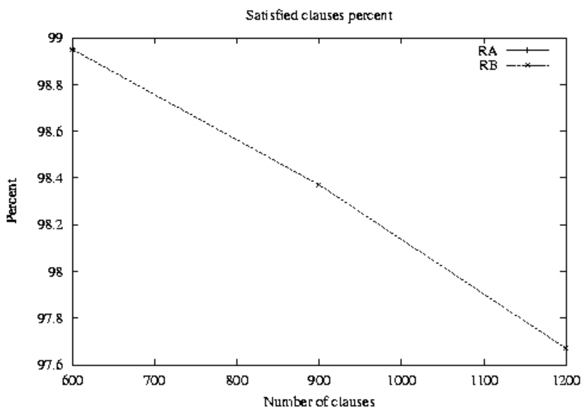

Fig. 4 - 300 variables

In Figure 5 we show a graphical view of the time differences between both algorithms with instances of 500 variables.

TABLE 5
RUNNING TIME AVERAGE - 500 VARIABLES

Clauses	RA	RB
1000	51.05	3.49
1500	143.44	6.42
2000	373.89	8.49

TABLE 6
SATISFIED CLAUSES PERCENT - 500 VARIABLES

Clauses	RA	RB
1000	98.94	98.94
1500	98.33	98.33
2000	97.72	97.72

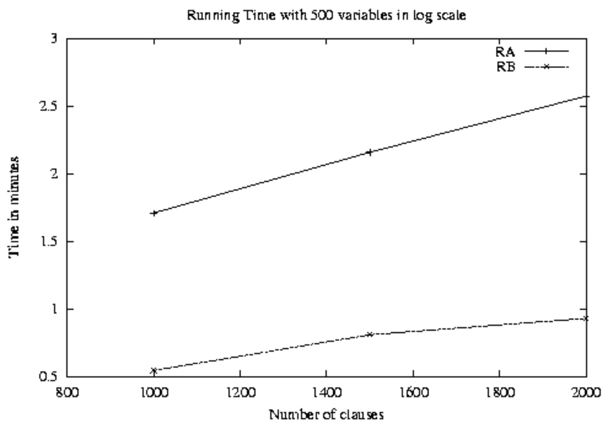

Fig. 5

In Figure 6 we show a graphical view of the satisfied clauses percent of both algorithms with instances of 500 variables.

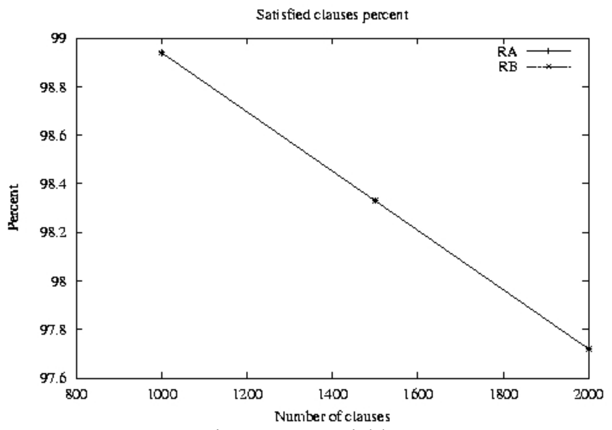

Fig. 6 - 500 variables

In Table 5 we show the time average in minutes of the heuristics with 500 variables. The difference in time, as with 100 and 300 variables, is notable; *RB* also obtains results of the same instances in seconds while *RA* takes minutes.

5 Conclusions

We have developed a heuristic based on strong components detection to Max3SAT that improves the time execution by eliminating leaves nodes, and the results obtained are good even when the size of the problem increases. The heuristic inputs a formula *F* in CNF and transforms it into a graph *G*(*F*), then detects the strong components in *G*(*F*) to thereby determine the component that maximizes the number of clauses satisfied in *F*.

We have improved the performance of the algorithm reducing the representation of the formula when we translate it into a graph, thereby avoiding adding the artificial variable (generated by the reduction from 3CNF to 2CNF) as leaves nodes in the graph.

As can be seen in Section 4, the new algorithm obtains the same number of clauses satisfied, but the running time decreases.

Future work should mathematically verify that this hypothesis is true.

References:

[1] S. Cook, The P versus NP Problem, University of Toronto, 2000.

[2] S. Joy, J. Mitchell, B. Borchers, Solving MAX-SAT and Weighted MAX-SAT Problems using Branch-and-Cut, Mathematical Science, Rensselaer Polytechnic Institute, Troy, NY 2180, 1998.

[3] Selman Bart, Henry A. Kautz, Brahm Cohen, Local Search Strategies for Satisfatibility Testing, AT&T Bell Laboratories, 1993.

[4] David Mitchell, Bart Selman, Hector Levesque, Hard and Easy Distributions of SAT Problems, Dept. of Computer Science, Simon Fraser University, AT&T Bell Laboratories, Dept. of Computer Science, University of Toronto.

[5] Pierre Hansen, Brigitte Jaumard Algorithms for the Maximum Satisfatibility Problem, Computing 44, pag 279-303 Springer Verlag 1990.

[6] M. Garey and D. Johnson. Computers and Intractability. W.H. Freeman, 1979.

[7] Esko Nuutila, An experimental study on transitive clousure representation, Laboratory of Information Processing Science, Helsinki University of Technology, Finland, 1996.

[8] Christos H. Papadimitriou, Computacional Complexity Addison-WesleyPublishing Company, pag 183 - 187, 1994.

[9] Guillermo Morales Luna, Guillermo De Ita Luna, Approximation algorithms for MaxSAT, Sección de Computación Centro de Investigación y Estudios Avanzados del IPN, Universidad Autónoma de Puebla, México, 2000.

[10] Oliver Kullmann, First report on an adaptive density based branching rule for DLL-like SAT solvers, using a database for mixed random conjuntive normal forms created using the Advanced Encryption Standard (AES), Computer Science Department, University of Wales Swansea, Swansea, UK, 2002