A note on some variations of MaxSAT algorithms

Guillermo De Ita
Universidad Autónoma de Puebla
gdeita@fcfm.buap.mx

Guillermo Morales
Centro de Investigación y
Estudios Avanzados del IPN
gmorales@alpha.cs.cinvestav.mx

January, 1997

Abstract

In this paper some variations of Johnson's algorithm for MaxSAT are examined. We present strategies to define the non-determinancies of that algorithm. First, in order to select current variables, there are considered variable weights as well as the usual clause weights. After, a modification of the algorithm is introduced with the aim to decide the processing order of the variables just before the main iteration in Johnson's algorithm. We present some practical comparisons.

1 Introduction

The SAT problem, consisting on deciding whether a Boolean conjunctive form is satisfied, is the prototypical element in the NP-complete problems class. Its optimization version *MaxSAT*, which computes the maximum number of simultaneously satisfiable clauses in a Boolean conjunctive form, has been widely studied as a representative problem in the class of combinatorial optimization problems.

The MaxSAT problem has a wide variety of applications such as the review of the consistency in expert systems, knowledge bases, integrity constraints in databases where one wants to know the reason for which the system cannot answer some queries: a knowledge base can be represented as a set of clauses, i.e. as a logical theory that may include facts, integrity constraints, inference rules and default clousure assumption. An update may contradict some previous information, rendering the theory inconsistent and the problem is to revise the previous information in order to restore consistency [1].

However, the main interest in MaxSAT is primarily motivated by the important role that it plays in a theory of approximation that has emerged in the last few years, and that MaxSAT is a prototypical problem representative of the complexity class MaxNP consisting of the optimization versions of problems in NP obeying the so called *linear reduction*[5].

1.1 Preliminary notions

Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be a set of Boolean variables. Let $Lit(\mathbf{X})$ be the set of literals: $Lit(\mathbf{X}) = \mathbf{X} \cup \{\overline{X_1}, \ldots, \overline{X_n}\}$.

A *clause* is a sequence of literals and is interpreted as the disjunction of the literals in the sequence, and a *conjunctive form, CF,* is a sequence of clauses and is interpreted as the conjunction of its clauses.

An *assignment* is a set of variables: A variable assumes the truth value *true* if it belongs to the assignment, otherwise it assumes the value *false*. If X is false under the assignment it is said, by an abuse of language, that $\overline{X}$ *is in* the assignment and we write $\overline{X} \in A$.

A clause is true if at least one of its literals is true. A CF is true if each of its clauses is true.

Given a CF

$$F = \bigwedge_{i=1}^{m} C_i = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} l_{ij}$$

where F is the sequence of clauses $[C_1, \ldots, C_m]$, each clause C_i is the sequence of literals $[l_{i1}, \ldots, l_{ik_i}]$ and k_i is the length of C_i.

For any assignment $A \subset \mathbf{X}$, let $c(F, A)$ be the number of clauses satisfied by A, i.e.:

$$c(F, A) = \text{card}\{i \leq m | \exists j \in [1, k_i] : l_{ij} \in A\},$$

where $\text{card}(S)$ denotes the *cardinality* of set S.

Evidently, each CF F with m clauses is *satisfiable* iff there is an assignment A such that $c(F, A) = m$.

MaxSAT is the problem that, given a CF, obtains an assignment that maximizes the number of satisfied clauses:

Instance: A CF F.
Solution:
An assignment $A_0 \subset \mathbf{X}$ such that

$$c(F, A_0) = \text{Max}\{c(F, A) | A \subset \mathbf{X}\}.$$

1.2 Approximation in optimization problems

Let $P(\cdot, \cdot)$ be a prototypical *optimization problem*:

Instance: A real function $f : D \to R$.
Solution:
A point $x_0 \in D$ realizing the maximum of f over the domain D: $f(x_0) = \text{Max}\{f(x) | x \in D\}$.

The *size* of the problem $P(f, D)$ is, let us say, the number of bits necesary to write the pair (f, D).

Let $Sol(P(f, D))$ be the set of solutions of $P(f, D)$.

Let $A : (f, D) \mapsto A(f, D) \in D$ a procedure that proposes for any instance (f, D) a candidate for being an element of $Sol(P(f, D))$. We say that the algorithm A *approximates* the problem P with *ratio* $r \in R$ if for each instance (f, D) of P we have $|f(A(f, D))| \geq r \cdot |f(\mathbf{x})|$ for some solution $\mathbf{x} \in Sol(P(f, D))$, in other words, the maximized value proposed by the algorithm is within a ratio r to the actual maximize value.

A is of *polynomial time* if there is an exponent $m \geq 0$ such that $t_A(f, D) = O(Size(f, D)^m)$, where t_A is a measure function of the algorithm A running time.

Any polynomial time algorithm A approximating the optimization problem within a ratio r is said to be a *polynomial time approximation algorithm* (PTAA) of ratio r.

There arise several possibilities:

1. There is a sequence of algorithms $\{A_n\}_n$ such that each A_n is a PTAA of ratio r_n and the sequence $\{r_n\}_n$ is such that $r_n \underset{n \to +\infty}{\nearrow} 1$. In this case the collection of algorithms $\{A_n\}_n$ is said to be a *polynomial time approximation scheme* (PTAS).
2. Just for some ratio $r < 1$ there is a PTAA.
3. For no r there is a PTAA.

There is naturally the problem to characterize the optimization problems falling in each of the above categories.

For many optimization problems, algorithms with small worst-case ratios were quickly found. In some other cases, the problem to find a worst-case ratio r algorithm, for any constant $r \in [0, 1]$, was proved to be as hard as finding an optimal solution.

2 Algorithms

In this section we present a basic algorithm (due to Johnson [2]) for MaxSAT and some variations.

2.1 Johnson's Algorithm

Let $F = [C_1, \ldots, C_m]$ be a CF. The algorithm uses the notion of *informational weight* of each clause:

$$C \in F \;\Rightarrow\; w(C) = 2^{-|C|},$$

where $|C|$ denotes the clause C length.

For each variable, it measures how convenient is it to assign the value *true* or the value *false* to that variable. This convenience is related to the sum of the informational weights of the clauses satisfied by the value of the variable.

In order to sketch the algorithm, let us use the following symbols:

A	:	the sought assignment,
a	:	the cardinality of A,
Left	:	the set of clauses that remain to be valuated,
Var	:	the set of variables that remain to be assigned,
c	:	the current counter of satisfied clauses.

Let us proceed as follows:

1. Initially let us make $A = \emptyset$, $a = 0$, *Left* $= F$, *Var* $= \mathbf{X}$ and $c = 0$.
2. While there remain unevaluated clauses and unassigned variables:
   (a) Choose an unassigned variable X,
   (b) let VT be the set of clauses that become *true* when the *true* value is attached to X, and
   (c) let VF be the set of clauses that become *true* when the *false* value is attached to X.
   (d) Let ST be the weights sum of clauses in VT, and
   (e) let SF be the weights sum of clauses in VF.
   (f) According to whether $ST \geq SF$ put X or $\overline{X}$ into A and update accordingly the values of the variable symbols.

```
{  for each C ∈ F do w(C) := 2^{-|C|} ;
   A := ∅ ; a := 0 ;
   Left := F ; Var := X ; c := 0 ;
   while (Left ≠ ∅) ∧ (Var ≠ ∅) do
   {  (*) choose a X ∈ Var ;
      Var := Var − {X} ;
      VT := {C ∈ Left | X ∈ C} ;
      ST := Σ_{C∈VT} w(C) ;
      VF := {C ∈ Left | X̄ ∈ C} ;
      SF := Σ_{C∈VF} w(C) ;
      if ST ≥ SF then
      {  A := A ∪ {X} ; a++ ;
         c := c + card(VT) ; Left := Left − VT ;
         for each C ∈ VF do w(C) := 2 · w(C) }
      else
      {  A := A ∪ {X̄} ;
         c := c + card(VF) ; Left := Left − VF ;
         for each C ∈ VT do
         w(C) := 2 · w(C) } }
}
```

Figure 1: Johnson's Algorithm.

(g) If the clauses in VT have become *true*, the clauses in VF should be satisfied by their remaining literals, i.e. these clauses may be regarded as having lost the false literal $\overline{X}$. Thus, their informational weights must be duplicated.

(h) Otherwise the updating is symmetric.

In Figure 1 we present the above algorithm in a pseudocode. This algorithm may give rise in a natural way to a family of algorithms, [1], by fixing strategies to choose the current assigning variable at step (*) in the pseudocode.

Independently of the way to choose the current assigning variable, Johnson's algorithm has polynomial-time complexity with a worst case approximation bound of $(1 - \frac{1}{2^k})$ where k is the minimum number of literals in any clause.

Papadimitrou and Yannakakis [5] have showed that every problem in MaxNP is constant-approximable. Even more, there is a constant threshold $c < 1$ such that if MaxSAT could be approximated in polynomial time with ratio better than c then $P = NP$. In view of this, it is important to take a closer look at the approximability properties of MaxSAT with the eventual goal of determining the best possible ratio r that can be achieved in polynomial time. Yannakakis had showed that instances of MaxSAT can be transformed equivalently, regarding the approximation properties, to instances with at least two literals per clause, in which case the Johnson's algorithms give an approximation constant $\frac{3}{4}$.

2.2 Greedy algorithm

Given an instance of MaxSAT it is quite relevant the order in which the variables are instantiated when searching for an assignment that maximizes the number of satisfied clauses.

For instance, let us consider the CF $F = \{\{X_1\}, \{\overline{X_1}, X_2\}, \{\overline{X_1}, X_3\}, \{\overline{X_1}, X_4\}\}$ and let us proceed following the lexicographical ordering. The first variable to be selected is X_1 and it gives the computed weights sums $ST = 0.5$ and $SF = 0.75$. Since $ST < SF$ the *False* value is attained to X_1. But this assignment renders F to be false. Thus, the lexicographical ordering fails to find a maximum of the satisfied clauses in the (satisfiable) CF F.

A first strategy to choose a current assigning variable in Johnson's algorithm is to select, in each iteration of the algorithm, the variable that appears most with a defined sign and in shorter clauses.

For this purpose, let us define the informational weight for each variable as

$$w(X) := \mathrm{Abs}\left(\sum_{X \in C_i} \frac{1}{|C_i|} - \sum_{\overline{X} \in C_j} \frac{1}{|C_j|}\right),$$

with the evident intention to discriminate the variables that appear mostly with a given "sign", and between these, those variables that appear in shorter clauses. Thus, a variable has greater informational weight when it appears most with a sign than with its opposite and it is "more" essential to some clauses.

In order to make efficiently the selection of the current variable, we can proceed as follows:

1. Order in descendent form the set of variables according to its informational weight.
2. Select at each iteration the current variable according to this ordering.

In Figure 2 we present the pseudocode of this strategy. The variable symbols have the same meaning as in the pseudocode in Figure 1.

A second strategy to choose a current assigning variable is to select the variable that maximizes the distance $|ST - SF|$.

for each $X \in Var$ **do**

$$\{\ w(X) := \mathrm{Abs}\left(\sum_{X \in C_i} \frac{1}{|C_i|} - \sum_{\overline{X} \in C_j} \frac{1}{|C_j|}\right)\ \};$$

order Var according the values of weights $w(X)$

Figure 2: A greedy strategy to choose a current assigning variable.

{ **for each** $X_1 \in Var$ **do**

$\{ VT(X_1) := \{C \in Left | X_1 \in C\}$;

$$ST(X_1) := \sum_{C \in VT(X_1)} w(C)\ ;$$

$VF(X_1) := \{C \in Left | \overline{X_1} \in C\}$;

$$SF(X_1) := \sum_{C \in VF(X_1)} w(C)\ \};$$

let X be such that

$$|ST(X) - SF(X)| = \max_{X_1 \in Var} |ST(X_1) - SF(X_1)|$$

}

Figure 3: A greedy strategy to choose a current assigning variable.

In Figure 3 we present the pseudocode of this strategy.

The mere substitution of step (*) in Figure 1 by the pseudocode in Figure 3 is evidently very expensive since it introduces $O(m \cdot n)$ (redundant) calculations. Instead, it is possible to order the set of variables $\mathbf{X}$ according to the values of $|ST - SF|$, related firstly to the whole CF F, and later to modify the list of ordered variables as new variables are gotten instantiated. In fact, such modifications are realized just by redefining the clause weights. Whenever a clause is satisfied by an assignment, its weight becomes null. If a false literal appears in a clause then the clause weight is duplicated.

In Figure 4 at the end of the paper we present the pseudocode for this procedure.

Finally, let us remark that the clause informational weights are defined through the exponential function

$$k \mapsto 2^{-k}.$$

Similarly, for any decreasing function $f : N \to R^+$ such that $f(n) = o(1)$, i.e. $f(n) \underset{n \longrightarrow +\infty}{\searrow} 0$, the informational weight of any clause C can be defined as $w(C) = f(|C|)$. Let $h : k \mapsto \frac{f(k)}{f(k+1)}$. Thus the weights updating in Johnson's algorithm is realized by making

$$w(C) := h(|C|)w(C)\ ;\ |C| - -$$

where the operation $X \mapsto X - -$ is the usual *predecessor* function. The greedy variation gives rise to a class of algorithm by changing the function f. In particular, we may consider the (much slower) decreasing harmonic function $f : n \mapsto n^{-1}$.

m	Algorithm.1	Algorithm.2	Algorithm 3
100	3.2	2.6	2
220	10.8	10.2	8
350	28.	26.	24
600	33.	29.6	24
900	53.3	52.3	42
1200	77.	70.3	65

Table 1: Average numbers of unsatisfied clauses, considering five sample CF's.

3 Computational results

In order to show our examples let us refer to Johnson's algorithm (figure 1) as Algorithm 1, the first variant with the selection scheme in figure 2 is Algorithm 2 and the variation in figure 4 is Algorithm 3.

Fixed the total number of clauses, $m \in [100, 1200]$, we built randomly five sets of clauses, each defining a CF, with a fixed number of variables, $n = 50$, and at each set the average clause length was $k = 3$.

The average numbers of unsatisfied clauses, for any fixed m, are shown in table 1, and they are plotted in Figure 5. The average running times, for any fixed m, are shown in table 2, and they are plotted in Figure 6.

4 Conclusions

The implementations of Johnson's algorithm and their discussed variations showed particular behaviours. In some cases we got better performances according to the obtained maximal value whilst their runnings were slightly slower.

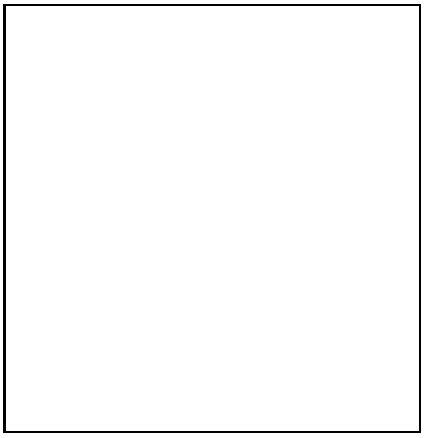

Figure 4: Graph corresponding to table 1.

m	Algorithm.1	Algorithm.2	Algorithm.3
100	0.53	0.53	1.7
220	1.52	1.52	3.6
350	3.	3.05	6.0
600	7.39	7.5	13.5
900	16.	16.0	25.5
1200	26.84	26.8	39.14

Table 2: Average numbers of running times (in seconds), considering five sample CF's.

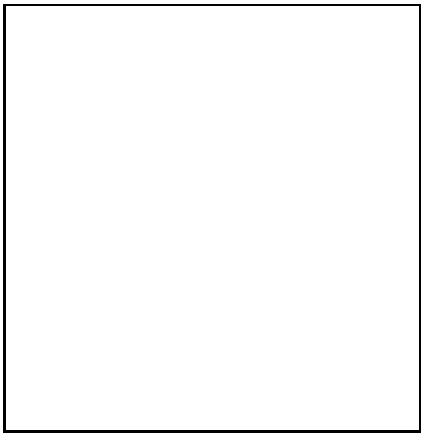

Figure 5: Graph corresponding to table 2.

Note that since the choice of the current assigning variable in the Johnson's original algorithm is left largely undetermined, there is some potential for improvement the average case behaviour. The proposed variations maintain the polynomial-time complexity, in fact it is increased by the product of a linear term and get approximation rates much higher than the general estimated approximation bound. Indeed, in our proofs this rate has been always higher than .9 and our third variation has showed the best results.

References

[1] P. Hansen and B. Jaumard, *"Algorithms for the Maximum Satisfiability Problem"*, Computing 44, 279-303, 1990.

[2] D. Johnson, *"Approximation algorithms for combinatorial problems"*, Journal of Computer and System Sciences 9, 256-278, 1974.

[3] D. Johnson, *"The NP-Completeness Column: An Ongoing Guide"*, Journal of Algorithms, Vol.13, 502-524,1992.

[4] M. Yannakakis, *"On the Approximation of Maximum Satisifiability"*, Journal of Algorithms, Vol. 17, pp. 475-502, 1994.

[5] Papadimitrou C., Yannakakis M.: *"Optimization, approximation and complexity classes"*, Jour. of computer and syst. sci. vol. 43, pp. 425-440, 1991.

{ **for each** $C \in F$ **do** $w(C) := 2^{-|C|}$;
$A := \emptyset$; $a := 0$;
Left $:= F$; *Var* $:= \mathbf{X}$; $c := 0$;
for each $X_1 \in$ *Var* **do**
$\{VT(X_1) := \{C \in$ *Left*$|X_1 \in C\}$;
$VF(X_1) := \{C \in$ *Left*$|\overline{X_1} \in C\}$ $\}$;
while (*Left* $\neq \emptyset$) $\wedge$ (*Var* $\neq \emptyset$) **do**
{ **for each** $X_1 \in$ *Var* **do**
$\{ST(X_1) := \sum_{C \in VT(X_1)} w(C)$;
$SF(X_1) := \sum_{C \in VF(X_1)} w(C)$ $\}$;
let X be such that
$|ST(X) - SF(X)| = \max_{X_1 \in Var} |ST(X_1) - SF(X_1)|$;
if $ST \geq SF$ **then**
{ $A := A \cup \{X\}$; $a{+}{+}$;
$c := c + \mathrm{card}(VT)$; *Left* $:=$ *Left* $- VT(X)$;
for each $C \in VT(X)$ **do** $w(C) := 0$;
for each $C \in VF(X)$ **do** $w(C) := 2 \cdot w(C)$ $\}$
else
{ $A := A \cup \{\overline{X}\}$;
$c := c + \mathrm{card}(VF)$; *Left* $:=$ *Left* $- VF(X)$;
for each $C \in VF(X)$ **do** $w(C) := 0$;
for each $C \in VT(X)$ **do** $w(C) := 2 \cdot w(C)$ $\}$ $\}$
}

Figure 6: A greedy variation of Johnson's Algorithm.