Approximation algorithms for MaxSAT

Guillermo Morales-Luna* Guillermo De Ita Luna†

April, 1995

Abstract

The main aim of NP-completeness theory is the analysis of intractability. Many of the first problems shown NP-hard were optimization problems. Due to the fact that finding optimal solutions to such problems takes exponential time, the alternative was investigated over algorithms that, working in polynomial time, could find solutions "near-optimal". The term *approximation algorithm* soon was used to denote any algorithm that takes this approach.

In this paper we show the basics of Approximation Theory for NP-completeness and sketch a collection of algorithms.

1 Introduction

The satisfiability problem is the first problem shown to be NP-complete. It is NP-complete even for the case where each clause consists of exactly three literals. The problem of maximum satisfiability is NP-complete even for the case where each clause has at most two literals. In this paper we make a review of the most famous methods to approximate solutions of MaxSAT. The approximation problem is important because it underlines the computational difference between intractability and "practical" computation.

In this section we introduce the preliminary notions of our presentation. In the second section we present the relation between Max SAT and other equivalent problems. Later we present the basic approximation algorithms due to Johnson, thereafter we concentrate on Max2SAT and we continue to algorithms for the general case of MaxSAT. Finally we make some general considerations from the Complexity Theory point of view.

*Sección de Computación, Centro de Investigación y Estudios Avanzados del IPN, México, D.F. MEXICO. gmorales@alpha.cs.cinvestav.mx

†Facultad de Ciencias Físico-Matemáticas, Universidad Autónoma de Puebla, Puebla, MEXICO. gdeita@fcfm.buap.mx

1.1 The problem MaxSAT

Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be a set of variables. The set of *literals* over $\mathbf{X}$ is $\mathit{Lit}(\mathbf{X}) = \{X, \overline{X} | X \in \mathbf{X}\}$. A *clause* is a list of literals, and is read as the disjunction of its literals. A *clause form* is a list of clauses, and is read as the conjunction of its clauses. A *weighted formula* is a pair $WF = (CF, \mathbf{w})$ where $CF = (C_i)_{i \leq m}$ is a clause form and $\mathbf{w} = (w_i)_{i \leq m} \in N^m$ is an integer vector. For each $i \leq m$, w_i is the *weight* of the clause $C_i = (l_{ij})_{j \leq k_i}$.

Let $\mathit{Two}^n = \{0,1\}^n$ be the set of *truth values*. An *assignment* $\boldsymbol{\epsilon} \in \mathit{Two}^n$ determines a *weight value* at the weighted formula WF as $wc(CF, \mathbf{w}; \boldsymbol{\epsilon}) = \sum_{C_i(\boldsymbol{\epsilon})=1} w_i$.

Maximum satisfiability (MaxSAT) is the following problem:

Instance: A weighted formula WF.
Solution: An assignment $\boldsymbol{\epsilon}_0 \in \mathit{Two}^n$ that maximizes the weight values:

$$wc(CF, \mathbf{w}, \boldsymbol{\epsilon}_0) = \mathrm{Max}\,\{wc(CF, \mathbf{w}, \boldsymbol{\epsilon}) | \boldsymbol{\epsilon} \in \mathit{Two}^n\}\,.$$

Max$k$SAT is the same problem MaxSAT corresponding to clauses with exactly k literals.

Two weighted forms $(CF_1, \mathbf{w}_1)$, $(CF_2, \mathbf{w}_2)$ are *equivalent* if for each assignment $\boldsymbol{\epsilon} \in \mathit{Two}^n$ the same weight value is obtained, $wc(CF_1, \mathbf{w}_1, \boldsymbol{\epsilon}) = wc(CF_2, \mathbf{w}_2, \boldsymbol{\epsilon})$.

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: $f(x_0) = \mathrm{Max}\{f(x) | x \in D\}$.

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

Let $\mathit{Sol}(P(f, D))$ 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 $\mathit{Sol}(P(f, D))$. 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 $\mathbf{x} \in \mathit{Sol}(P(f, D))$, 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 $t_A(f, D) = O(\mathit{Size}(f, D)^m)$, 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. Feige [F], proved that there is no polynomial time approximation algorithm for the Clique problem, for any constant ratio, unless P=NP.

The maximum satisfiability problem (MaxSAT) is the version of optimization of the basic satisfiability problem. Since the MaxSAT problem is NP-hard, even in the particular case that there are just two variables per clause, this is a natural problem to be treated by the approximation methods, based on heuristic algorithms. Indeed, the computing time required either to find an optimal solution, or to prove the optimality of such a solution when one is found, becomes prohibitive as the problem gets large.

This kind of results was reinforced by Arora et al.[A], namely, they showed that Max3SAT does not have a polynomial time approximation scheme (the solutions of this problem can not be approximated within any optimal constant factor r, arbitrarily close to 1) unless P=NP, and as a consequence neither does any other MaxSNP-hard problem (the MaxSNP complexity class consists of the optimization versions of NP-complete problems).

Thus, there is a constant threshold $r < 1$ such that if Max3SAT could be approximated in polynomial time with ratio better than c then P=NP. In this way, MaxSAT may be a prototypical problem to determine the best ratio r that can be achieved in polynomial time.

2 Relation of MaxSAT with other problems

2.1 MaxCUT

This a typical problem in graph theory. Let us recall that a *(edge)-weighted graph* is a graph $G = (V, E)$ together with a function $w : V^{(2)} \to N$ such that $\forall \{v_1, v_2\} \in V^{(2)} : \{v_1, v_2\} \notin E \Rightarrow w(\{v_1, v_2\}) = 0$.

If $\{U, U^c\}$ is a partition of the set of nodes V, then the induced *cut* is the set of edges that goes from one set to the other in the partition:

$$Cut(U) = \{e \in E | (e \cap U \neq \emptyset) \wedge (e \cap U^c \neq \emptyset)\}.$$

The *weight of the cut* is $wc(U) = \sum_{e \in Cut(U)} w(e)$.

MaxCUT is the problem consisting of the calculation of a maximum cut for a given weighted graph.

Proposition 2.1 *If Max2SAT has a PTAA with ratio r then MaxCUT has a PTAA with ratio $\frac{r}{3-2r}$.*

Let $G = (V, E)$ be an instance of MaxCUT, consisting of n nodes and k edges. Let us make

- Variables: $\mathbf{X} = V$.
- Clauses: $F = \{X \vee Y, \overline{X} \vee \overline{Y} | \exists e \in E : e = \{X, Y\}\}$.

For any assignment $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n) \in Two^n$, we may also write $\boldsymbol{\epsilon} = (\epsilon_X)_{X \in \mathbf{X}}$, at least one of the clauses $X \vee Y, \overline{X} \vee \overline{Y}$ is true. In fact, both are true if and only if $\epsilon_X \neq \epsilon_Y$.

For each partition $\{U, U^c\}$ of V consider the assignment $\boldsymbol{\epsilon}_U : X \mapsto \begin{cases} 1 & \text{if } X \in U \\ 0 & \text{if } X \notin U \end{cases}$

If the partition induces a cut of k_U edges then $\boldsymbol{\epsilon}_U$ should satisfy $k + k_U$ clauses.

Suppose that z_0 is the actual maximum number of simultaneously satisfiable clauses, and that there is a PTAA that calculates an assignment satisfying $z \geq r \cdot z_0$ clauses. The corresponding partitions define cuts with $z - k$ and $z_0 - k$ edges. Evidently,

$$\begin{array}{rcll} r \cdot (z_0 - k) & \leq & (z - k) + k(1 - r) & (\text{because } k(1 - r) \geq 0), \\ & \leq & (z - k) + 2(z - k)(1 - r) & (\text{because } \frac{k}{2} \leq (z - k)), \\ & = & (z - k) \cdot (3 - 2r) & \end{array}$$

i.e. $\frac{r}{3-2r} \cdot (z_0 - k) \leq (z - k)$.

2.2 Maximizing quadratic polynomials over the unit cube

The *unconstrained quadratic 0-1 programming* (UQ01P) problem has as instance a quadratic polynomial $Q(\mathbf{X}) = q_0 + \sum_{j=1}^{n} q_j X_j + \sum_{1 \le j_1 < j_2 \le n} q_{j_1 j_2} X_{j_1} X_{j_2}$, and looks toward a (global) maximum over the vertices of the unit cube $I^n = [0,1]^n$.

Any instance of Max2SAT is transformed into an equivalent instance of UQ01P by the following codification:

- *Literals*: If X^ϵ is a literal, where $X^\epsilon) = \begin{cases} X & \text{if } \epsilon = 1 \\ \overline{X} & \text{if } \epsilon = 0 \end{cases}$ being $X \in \mathbf{X}$, let $\Phi(X^\epsilon) = \begin{cases} X & \text{if } \epsilon = 1 \\ 1 - X & \text{if } \epsilon = 0 \end{cases}$

  Thus for any literal l and any assignment $\boldsymbol{\epsilon} \in \mathit{Two}^n$ we have:

$$\begin{array}{lcl} l(\boldsymbol{\epsilon}) = 1_{\text{Boolean}} & \Leftrightarrow & \Phi(l)(\boldsymbol{\epsilon}) = 1_{\text{real}}, \\ l(\boldsymbol{\epsilon}) = 0_{\text{Boolean}} & \Leftrightarrow & \Phi(l)(\boldsymbol{\epsilon}) = 0_{\text{real}}. \end{array}$$

- *Clauses*:

  - *Unit clauses*: For any unit clause $C = l$ let $\Phi(C) = \Phi(l)$.
  - *Length two clauses*: For any clause of the form $C = l_1 \vee l_2$ let $\Phi(C) = \Phi(l_1) + \Phi(l_2) - \Phi(l_1)\Phi(l_2)$.

  For any clause C with at most two literals and any assignment $\boldsymbol{\epsilon} \in \mathit{Two}^n$ we have:

$$\begin{array}{lcl} C(\boldsymbol{\epsilon}) = 1_{\text{Boolean}} & \Leftrightarrow & \Phi(C)(\boldsymbol{\epsilon}) = 1_{\text{real}}, \\ C(\boldsymbol{\epsilon}) = 0_{\text{Boolean}} & \Leftrightarrow & \Phi(C)(\boldsymbol{\epsilon}) = 0_{\text{real}}. \end{array}$$

- For a form $(\mathit{CF} = \{C_i\}_{i=1}^n, \mathbf{w} = \{w_i\}_{i=1}^n)$, let $\Phi(\mathit{CF}) = \sum_{i=1}^n w_i \Phi(C_i)$. Then, for any $\boldsymbol{\epsilon} \in \mathit{Two}^n$ we have: $wc(\mathit{CF}, \mathbf{w}, \boldsymbol{\epsilon}) = \Phi(\mathit{CF})(\boldsymbol{\epsilon})$.

Thus the calculation of an assignment maximizing the number of satisfied clauses is equivalent to maximize the corresponding quadratic polynomial in the vertices of the unit cube.

Reciprocally, any instance of UQ01P is transformed into an equivalent instance of Max2SAT by the following procedure:

Given a quadratic polynomial $Q(\mathbf{X}) = q_0 + \sum_{j=1}^{n} q_j X_j + \sum_{1 \le j_1 < j_2 \le n} q_{j_1 j_2} X_{j_1} X_{j_2}$, we may rewrite it in the form $Q(\mathbf{X}) = c - G(\mathbf{X}, \overline{\mathbf{X}})$ where $G(\mathbf{X}, \mathbf{Y})$ is a quadratic polynomial with just positive coefficients and c is a constant. Indeed,

- if $q_j > 0$ then substitute the monomial $q_j X_j$ by its equivalent form $q_j - q_j \cdot \overline{X_j}$, because $X_j = (1 - (1 - X_j)) = 1 - \overline{X_j}$,

- if $q_{j_1 j_2} > 0$ then substitute the monomial $q_{j_1 j_2} X_{j_1} X_{j_2}$ by its equivalent form $q_{j_1 j_2} - q_{j_1 j_2} \overline{X_{j_1}} - q_{j_1 j_2} X_{j_1} \overline{X_{j_2}}$, because $X_{j_1} X_{j_2} = X_{j_1} - X_{j_1} \overline{X_{j_2}} = 1 - \overline{X_{j_1}} - X_{j_1} \overline{X_{j_2}}$.

Now for each term dT in $G(\mathbf{X}, \overline{\mathbf{X}})$ we consider a weighted clause:

- if $dT = d \cdot X^{\epsilon}$ is a linear term in $G(\mathbf{X}, \overline{\mathbf{X}})$ put the unit clause $C = X^{-\epsilon}$ in CF with weight d, and
- if $dT = d \cdot X^{\epsilon} Y^{\delta}$ is a quadratic term in $G(\mathbf{X}, \overline{\mathbf{X}})$ put the clause $C = X^{-\epsilon} \vee Y^{-\delta}$ in CF with weight d,

in any case, for each $\boldsymbol{\epsilon} \in \mathit{Two}^n$,

$$\begin{array}{rcl} T(\boldsymbol{\epsilon}) = 0_{\text{real}} & \Leftrightarrow & C(\boldsymbol{\epsilon}) = 1_{\text{Boolean}}, \\ T(\boldsymbol{\epsilon}) = 1_{\text{real}} & \Leftrightarrow & C(\boldsymbol{\epsilon}) = 0_{\text{Boolean}}, \end{array}$$

thus $G(\mathbf{X}, \overline{\mathbf{X}})\big|_{\mathbf{X}=\boldsymbol{\epsilon}}$ is the weight sum corresponding to the clauses that are not satisfied by $\boldsymbol{\epsilon}$, and consequently $Q(\boldsymbol{\epsilon}) = (c - W) + wc(CF, \boldsymbol{\epsilon})$ where $W = \sum_{T \in \mathit{Terms}} d(T)$ is the sum of all weights.

A PTAA in UQ01P gives an estimation, in polynomial time, of a lower bound for the maximum of the objective quadratic polynomial function.

2.3 Integer programming

Let us consider the following *integer programming* (IP) problem:

$$\text{Maximize} \quad \sum_{i=1}^{m} w_i y_i \qquad \text{with respect to } \mathbf{x}, \mathbf{y},$$

$$\text{subject to:} \quad \left\{ \begin{array}{ll} y_i \leq \sum_{X_j \in C_i} x_j + \sum_{\overline{X_j} \in C_i} (1 - x_j) & i = 1, \ldots, m \\ x_j \in \{0, 1\} & j = 1, \ldots, n \\ 0 \leq y_i \leq 1 & i = 1, \ldots, m \end{array} \right.$$

If $(F = \{C_i\}_{i=1}^m, \mathbf{w} = (w_i)_{i=1}^m)$ is an instance of MaxSAT then we may transform solutions $(\boldsymbol{\epsilon} \in \mathit{Two}^n, wc_M)$ of MaxSAT into solutions of the corresponding integer programming problem by making

$$\begin{array}{rcl} \forall j = 1, \ldots, n & : & x_j = 1 \;\Leftrightarrow\; \epsilon_j = 1, \\ & & x_j = 0 \;\Leftrightarrow\; \epsilon_j = 0 \\ \forall i = 1, \ldots, m & : & y_i = 1 \;\Leftrightarrow\; C_i = 1, \\ & & y_i = 0 \;\Leftrightarrow\; C_i = 0, \end{array}$$

and both maxima will coincide, with common value wc_M.

The (IP) problem may be *relaxed* by changing each condition of the form $x_j \in \{0, 1\}$ by the condition $0 \leq x_j \leq 1$ over the real numbers. Call the relaxed problem the *linear programming* (LP) problem. Let Y_{IP} and Y_{LP} be the respective maxima of IP and LP. Evidently we must have $Y_{\text{IP}} \leq Y_{\text{LP}}$. In fact, if each clause in the MaxSAT instance has more than one literal then $x_j = \frac{1}{2}$, $\forall j$, and $y_i = 1$, $\forall i$, will give a solution of LP. Such solution does not define any solution to MaxSAT. In this case, relaxation is a vacuous procedure.

If there are unit clauses then we may solve the corresponding IP problem as follows:

1. Solve the relaxed linear programming problem LP (which can be solved in polynomial time because its coefficients are in the discrete set $\{-1, 0, 1\}$). Let $(\mathbf{x}^*, \mathbf{y}^*)$ be a solution.

2. Find an assignment by the Johnson's probabilistic algorithm, with probabilities $p_j = x_j^*$, $\forall j$.

It can be shown that this procedure gives an approximation ratio $r = \left(1 - \frac{1}{e}\right)$ (see[GW]).

3 Basic algorithms for MaxSAT

These algorithms proceed incrementally: Choose a literal to be instantiated (maximizing in some way the number of satisfied clauses), omit the satisfied clauses, omit the instantiated variable in the current clausal form and repeat until all variables are instantiated.

In the presentations of the basic algorithms we will use the same notation:

F	:	The given clausal form (algorithm's input),
$\boldsymbol{\epsilon}$	:	the sought assignment,
Sati	:	the set of clauses that have already been satisfied,
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,
z	:	the proposed maximum value (algorithm's output).

Also in our discussions, we will denote by q the minimum number of literals at each clause in the given clausal form F, by z_0 the actual solution of MaxSAT and by z its approximation calculated by the discussed algorithm.

In the algorithms, there are bounds for the approximation ratio expressed by relations of the form $z \leq g(q)z_0$, where g is a non-decreasing function $g : N \to [0, 1]$. Thus the approximations are worst as the clauses are shorter.

3.1 First algorithm

In figure 1 we present a first algorithm due to Johnson [J].

For this algorithm, we have $z \geq \frac{q}{q+1} z_0$.

```
{   ε := nil ; Sati := ∅ ;
    Left := F ; Var := X ; c := 0 ;
    while (Left ≠ ∅) ∧ (Var ≠ ∅) do
    {   let X ∈ Var be s.t. X^ε is the most frequent literal in Left ;
        Var := Var − {X} ;
        Sati_1 := {C ∈ Left|X^ε ∈ C} ; c_1 := card(Sati_1)
        ε := ε * {X ← ε} ;
        c := c + c_1 ; Left := Left − Sati_1 ;
        Sati := Sati ∪ Sati_1 };
    z := c
}
```

Figure 1: A first algorithm for MaxSAT (D. Johnson)

3.2 Second algorithm

The algorithm uses the notion of *informational weight* of each clause: $C \in F \Rightarrow w(C) = 2^{-\text{length}(C)}$.

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 figure 2 we present a second algorithm due also to Johnson [J].

For this algorithm, we have $z \geq \frac{2^q-1}{2^q} z_0 = \left(1 - \frac{1}{2^q}\right) z_0$.

There is a non-deterministic selection of the current assigned variable (step marked (*) in figure 2). Different strategies to select this variable gives rise to different versions of Johnson's algorithm.

In [DM] some of these strategies are shown and they are compared related to their approximation ratios and their running times.

3.3 A probabilistic algorithm

This algorithm is similar to the second Johnson algorithm, and has been presented in [Y].

Let $X_j : [0,1] \to \mathit{Two}$ be a random variable such that $\mathit{Prob}(X_j = 1) = p_j$, $j = 1, \ldots, n$. A random assignment is of the form $\boldsymbol{\tau} = \bigotimes_{j=1}^n (\tau_j \leftarrow X_j)$ (let's assume that the coordinate assignments are independent events). Thus, for each $\boldsymbol{\epsilon} \in \mathit{Two}^n$ we have $\mathit{Prob}(\boldsymbol{\tau} = \boldsymbol{\epsilon}) = \prod_{j=1}^n (p_j)_{\epsilon_j}$, where $(p_j)_{\epsilon_j} = \begin{cases} p_j & \text{if } \epsilon_j = 1, \\ 1 - p_j & \text{if } \epsilon_j = 0. \end{cases}$
The expected value of an assignment is $E(\boldsymbol{\tau}) = \sum_{\boldsymbol{\epsilon} \in \mathit{Two}^n} \mathit{Prob}(\boldsymbol{\tau} = \boldsymbol{\epsilon}) \cdot \boldsymbol{\epsilon}$. The

```
{   for each C ∈ F do w(C) := 2^(-length(C)) ;
    ε := nil ;
    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
        {   ε := ε ∗ (X ← 1) ;
            c := c + card(VT) ; Left := Left − VT ;
            for each C ∈ VF do w(C) := 2 · w(C) }
        else
        {   ε := ε ∗ (X ← 0) ;
            c := c + card(VF) ; Left := Left − VF ;
            for each C ∈ VT do w(C) := 2 · w(C) } }
}
```

Figure 2: A second algorithm

expected value of the weight function is

$$\tilde{W} = E(wc(F, \boldsymbol{\tau})) = \sum_{\boldsymbol{\epsilon} \in Two^n} wc(F, \boldsymbol{\epsilon}) Prob(\boldsymbol{\tau} = \boldsymbol{\epsilon}),$$

or equivalently

$$\tilde{W} = \sum_{i=1}^{m} w_i Prob(C_i = 1) = \sum_{i=1}^{m} w_i \left(1 - \prod_{X \in C_i} (1 - p_X) \prod_{\overline{X} \in C_i} p_X \right).$$

In figure 3 we sketch the pseudocode of a *greedy probabilistic* algorithm proceeding as follows

Input: A weighted clausal from F and a probability p_j for each variable X_j.
Output: An assignment $\boldsymbol{\tau}$ such that $wc(F, \boldsymbol{\tau}) \geq \tilde{W}$.

Proposition 3.1 *The weight value of the assignment computed by the algorithm in figure 3 is greater or equal than $\tilde{W}$.*

Indeed, at the beginning of the algorithm the expected weight value is exactly $\tilde{W}$.

Let us see that at each repetition of the algorithm's main cycle the expected values do not decrease.

Suppose that the current variable X_j has been assigned to 1 by the algorithm. Each clause $C \in VT(X_j)$ modifies its contribution to the expected value from $w(C)(1-u(C))$ to $w(C)$, and each clause $C \in VF(X_j)$ modifies its contribution to the expected value from $w(C)(1-u(C))$ to $w(C)(1-\frac{u(C)}{p_j})$. Thus the global updating of the expected values signifies a difference of

$$\sum_{C \in VT(X_j)} w(C)u(C) - \sum_{C \in VF(X_j)} w(C)u(C) \left(\frac{1}{p_j} - 1 \right)$$

which is positive because X_j has assumed the value 1.

If X_j assumes the value 0 the situation is symmetrical and it also gives an increase on the expected value.

In particular, for the probabilities $p_i = \frac{1}{2}, \forall i \leq m$, we have $\tilde{W} \geq \sum_{i=1}^{m} \left(1 - \frac{1}{2}\right) w_i = \frac{1}{2} \sum_{i=1}^{m} w_i$, thus we obtain an approximation ratio of $\frac{1}{2}$. But, if the shorter clause has q literals, then $\tilde{W} \geq \left(1 - \frac{1}{2^q}\right) \sum_{i=1}^{m} w_i$, thus the approximation ratio is $r = \left(1 - \frac{1}{2^q}\right)$.

$$
\begin{array}{l}
\{\quad \textbf{for each } C \in F \textbf{ do } u(C) := \left(\prod_{\overline{X_j} \in C} p_j\right)\left(\prod_{X_j \in C} (1-p_j)\right) \; ; \\
\quad \boldsymbol{\epsilon} := \mathit{nil} \; ; \\
\quad \mathit{Left} := F \; ; \; \mathit{Var} := \mathbf{X} \; ; \; c := 0 \; ; \\
\quad \textbf{while } (\mathit{Left} \neq \emptyset) \wedge (\mathit{Var} \neq \emptyset) \textbf{ do} \\
\quad \{\quad (*) \text{ choose a } X \in \mathit{Var} \; ; \\
\qquad \mathit{Var} := \mathit{Var} - \{X\} \; ; \\
\qquad VT := \{C \in \mathit{Left} | X \in C\} \; ; \; ST := \sum_{C \in VT} w(C)u(C) \; ; \\
\qquad VF := \{C \in \mathit{Left} | \overline{X} \in C\} \; ; \; SF := \sum_{C \in VF} w(C)u(C) \; ; \\
\qquad \textbf{if } p_X ST \geq (1-p_X)\, SF \textbf{ then} \\
\qquad \{\quad \boldsymbol{\epsilon} := \boldsymbol{\epsilon} * (X \leftarrow 1) \; ; \\
\qquad\quad c := c + \mathrm{card}(VT) \; ; \; \mathit{Left} := \mathit{Left} - VT \; ; \\
\qquad\quad \textbf{for each } C \in VF \textbf{ do } u(C) := \frac{u(C)}{p_X} \; \} \\
\qquad \textbf{else} \\
\qquad \{\quad \boldsymbol{\epsilon} := \boldsymbol{\epsilon} * (X \leftarrow 0) \; ; \\
\qquad\quad c := c + \mathrm{card}(VF) \; ; \; \mathit{Left} := \mathit{Left} - VF \; ; \\
\qquad\quad \textbf{for each } C \in VT \textbf{ do } u(C) := \frac{u(C)}{(1-p_X)} \; \} \; \} \\
\}
\end{array}
$$

$\forall C : u(C) =$ (probability that clause C is unsatisfied)

Figure 3: A greedy probabilistic algorithm.

4 At most two literals per clause

MaxSAT with at most two literals per clause is a NP-complete problem. Indeed any instance Φ of 3SAT (SAT with conjunctive forms of exactly three literals per clause) is transformed into an instance $WC = (CF, \mathbf{w})$ as follows:

If $\Phi = \{l_{i1}, l_{i2}, l_{i2}\}_{i=1}^m$ is an instance of 3SAT,

- let

$$CF = \left\{ \begin{array}{l} \{l_{i1}\}, \{l_{i2}\}, \{l_{i3}\}, \{Z_i\}, \\ \{\overline{l_{i1}}, \overline{l_{i2}}\}, \{\overline{l_{i1}}, \overline{l_{i3}}\}, \{\overline{l_{i2}}, \overline{l_{i3}}\}, \\ \{l_{i1}, \overline{Z_i}\}, \{l_{i2}, \overline{Z_i}\}, \{l_{i3}, \overline{Z_i}\} \end{array} \right\}_{i=1}^m ,$$

  where for each i, Z_i is a new variable,

- to each clause in $C \in CF$ put an unit value $w_C = 1$,

- and consider a threshold $t_0 = 7m$.

Then, Φ is satisfiable if and only if MaxSAT gives a maximum value not less than the threshold t_0.

4.1 A first Max2SAT algorithm

Both Johnson algorithms give an approximation ratio of $\frac{1}{2}$ when $q = 1$. For the special case of clauses with length at most 2 there are special properties.

Proposition 4.1 *Let F be a clausal form with clause with at most 2 literals each. If F is satisfiable then there is always an assignment $\boldsymbol{\epsilon} \in Two^n$ such that it satisfies at least $h \cdot m$ clauses, where m is the number of total clauses and $h = \frac{\sqrt{5}-1}{2}$ is the reciprocal of the* golden ratio.

Proposition 4.2 (Lieberherr & Specker, 1981) *An assignment $\boldsymbol{\epsilon}$ with the above property can be computed in polynomial time.*

In fact, the approximation can be computed by the following algorithm ([HJ]):

Input: A satisfiable clausal form F with m clauses, each of length at most 2.
Output: An assignment $\boldsymbol{\epsilon} \in Two^n$ satisfying at least $h \cdot m$ clauses.

1. For each variable $X \in Var$ whose complement appears as unary, substitute everywhere $\overline{X}$ by X.

   After this step, each unary clause will have a positive literal.

   This step can be done consistently because F is satisfiable.

2. Let p be the first prime number not less than n, let $\mathbf{Y}$ a set of p variables extending $\mathbf{X}$, and let $\{\pi_k^p\}_{0\leq k\leq p}$ be a randomly generated set of $p+1$ permutations of $\mathbf{Y}$.

3. For each $(k_1, k_2, k_3) \in P = [0,p] \times [1, p-1] \times [1,p]$ let us define the assignment $\boldsymbol{\epsilon}_{(k_1,k_2,k_3)} \in Two^p$ as follows:

$$\boldsymbol{\epsilon}_{(k_1,k_2,k_3)} : i \mapsto \begin{cases} 1 & \text{if } [(k_2\pi_{k_1}^p(i) + k_3) \bmod p] \leq k_1, \\ 0 & \text{otherwise.} \end{cases}$$

4. Let $\boldsymbol{\epsilon}_0$ be the assignment that maximizes the weight values over the assignments constructed in last step: $wc(F, \boldsymbol{\epsilon}_0) = \mathrm{Max}\{wc(F, \boldsymbol{\epsilon}_{(k_1,k_2,k_3)}) | (k_1, k_2, k_3) \in P\}$.

The above algorithm is polynomial and approximates the optimum by the ratio h.

4.2 Another algorithm for Max2SAT

In this subsection, F will be a weighted clausal form whose clauses have at most two literals. These procedures are due to M. Yannakakis and are exposed in full detail in [Y].

4.2.1 Elimination of unit clauses

First, let us see how we can transform a weighted clausal form with at most two literals per clause into forms with exactly two literals per clause.

Proposition 4.3 *For any weighted clausal form F, whose clauses are of length at most 2 over the set of variables $\mathbf{X}$, it is possible to effectively build*

- *a subset of n_1 variables $\mathbf{Y} \subset \mathbf{X}$,*
- *an assignment for $\mathbf{Y}$, $\boldsymbol{\sigma}_1 \in Two^{n_1}$,*
- *a non-negative constant k, and*
- *a weighted clausal form $\tilde{F}$ whose clauses are exactly of length 2 over the set of variables $\mathbf{X} - \mathbf{Y}$,*

such that

1. *for every assignment $\boldsymbol{\sigma}_2 \in Two^{n-n_1}$ of the variables in $\mathbf{X} - \mathbf{Y}$:*

$$wc(F, \boldsymbol{\sigma}_1 * \boldsymbol{\sigma}_2) = wc(\tilde{F}, \boldsymbol{\sigma}_2) + k,$$

and

2. *for every assignment* $\boldsymbol{\rho} \in Two^n$, *if* $\boldsymbol{\sigma}_2 = \boldsymbol{\rho}|_{\mathbf{X}-\mathbf{Y}}$ *then* $wc(F, \boldsymbol{\rho} \leq wc(\tilde{F}, \boldsymbol{\sigma}_2) + k$.

This proposition is proved by reduction to the *maximum flow* problem. We sketch succintly the proof:

Given a weighted clausal form F with at most two literals per clause let us construct rhe following network,

- *Nodes*: There are nodes of two types,
  - *extreme nodes*: there is a *source* node, s_0, and a *sink* node, s_1, we will assume $s_1 = \overline{s_0}$,
  - *literal nodes*: for each variable X there are two nodes labelled respectively by X and $\overline{X}$. and
- *Arcs and capacities*: Two arcs of the form $v_1 \to v_2$ and $\overline{v_2} \to \overline{v_1}$ are said to be *corresponding*.

  Each clause C, with weight w_C, determines a pair of corresponding arcs, each of capacity $\frac{w_C}{2}$ as follows,
  - if C is a unit clause, $C = l$, the associated arcs are $s_0 \to l$ and $\overline{l} \to s_1$,
  - if C is a clause with 2 literals, $C = l_1 \vee l_2$, the associated arcs are $\overline{l_1} \to l_2$ and $\overline{l_2} \to l_1$.

  The constructed network is *symmetric* in the sense that corresponding arcs have the same capacity.
- A *flow* is a function f : arcs $\to N$ such that
  - *capacity bounds*: the flow at each arc does not excede the arc capacity,
  - *flow conservation*: the total flow "arriving" at each node equals the flow "leaving" that node.

The Maximum Flow Problem searches for a path "connecting" the extreme nodes, with maximum flow.

Each flow determines a *residual* network R_f: If $e = [a \to b]$ is an arc, with capacity $c(e)$, then both arcs e and its reverse $b \to a$ are included into R_f with respective capacities $c(e) - f(e)$ and $c(e)$, provided that these values are positive. The residual network R_f is symmetric. One can see that the residual network corresponds to clauses with exactly two literals per clause. This network provides the construction of the clausal form $\tilde{F}$ whose existence is claimed in Prop. 4.3. In fact, the following properties hold:

- for any assignment $\boldsymbol{\sigma} \in Two^n$: $wc(F, \boldsymbol{\sigma}) = wc(\tilde{F}, \boldsymbol{\sigma}) + v_M$, where v_M is the value of the maximum flow in the constructed network, and it is time-polynomially computable,

- if f is a maximum flow then its residual network cannot contain a path connecting the extreme nodes s_0 and s_1,
- let R_{0f} be the set of nodes that are accesible from s_0 in the residual network R_f. Then for no variable X both literals X and $\overline{X}$ may belong to R_{0f}. By makin the
- By making True all the literals appearing in R_{0f} one obtains an assignment the claimed assignment $\boldsymbol{\sigma}_1$.

4.2.2 A 3/4 ratio PTAA for Max2SAT

As a consequence of Prop. 4.3 we have the following

Proposition 4.4 *If Max2SAT with exactly two literals per clause can be approximated within ratio r, then so can Max2SAT with at most two literals per clause. Thus, after Johnson's second algorithm, there is a PTAA with ratio $\frac{3}{4}$ for Max2SAT with at most two literals per clause.*

Indeed, suppose that we can find an assignment $\boldsymbol{\sigma}_2 \in \mathit{Two}^{n-n_1}$ of the variables in $\mathbf{X}-\mathbf{Y}$ such that $wc(F',\boldsymbol{\sigma}_2) \geq r \cdot \mathit{opt}(F')$. By 2 in Prop. 4.3, $\mathit{opt}(F) \leq \mathit{opt}(F') + k$. Thus, for $\boldsymbol{\sigma}_1$ as in Prop. 4.3 we have

$$wc(F,\boldsymbol{\sigma}_1 * \boldsymbol{\sigma}_2) = wc(F',\boldsymbol{\sigma}_2) + k \geq r \cdot \mathit{opt}(F') + k \geq r \cdot (\mathit{opt}(F') + k) \geq r \cdot (\mathit{opt}(F)).$$

5 MaxSAT for the general case

5.1 Yannakakis algorithm

Last algorithm may be extended to consider weighted clausal forms with clauses longer than 2. Let F be an arbitrary weighted clausal form.

1. Let $F^{\leq 2}$ be the form consisting of clauses of length at most 2 in F. Let $\tilde{F}^{\leq 2}$, $v_{M,2}$ and $R_{0f,2}$ be the corresponding notions according to prop. 4.3.

   By interchanging some variables with their complements, we may assume that $R_{0f,2}$ contains just positive literals.
2. Let $F_{\mathrm{new}} = \left(F - F^{\leq 2}\right) \cup \tilde{F}^{\leq 2}$. Then, any approximation procedure with ratio r in F_{new} will achieve the same ratio over F.
3. The clauses in F_{new} can be satisfied with a probability of $\frac{3}{4}$ except for those clauses
   - whose length is greater than 2 and have only complemented literals of variables in $R_{0f,2}$, and

- whose length is 3 and have two complemented literals of variables in $R_{0f,2}$.

In any case the corresponding lists of clauses are transformed into equivalent clausal forms that can be approximated within ratio $\frac{3}{4}$ (see [Y]).

5.2 Goemans-Williamson algorithm

The following procedure [GW] will use the same notation introduced in section 2.3. Given a weighted clausal form $F = \{C_i\}_{i=1}^m$ let us consider its corresponding IP problem

$$\text{Maximize} \quad \sum_{i=1}^{m} w_i y_i \qquad \text{with respect to } \mathbf{x}, \mathbf{y},$$

$$\text{subject to:} \quad \begin{cases} y_i \le \sum_{X_j \in C_i} x_j + \sum_{\overline{X_j} \in C_i} (1 - x_j) & i = 1, \ldots, m \\ x_j \in \{0, 1\} & j = 1, \ldots, n \\ 0 \le y_i \le 1 & i = 1, \ldots, m \end{cases}$$

Let LP be its linear programming relaxation and let Y_{IP} and Y_{LP} be the respective maxima of IP and LP.

When applying Johnson's probabilistic algorithm to F, the next proposition holds:

Proposition 5.1 *Let $\tilde{W}_1$ be the expected value corresponding to the probability $p_j = \frac{1}{2}$, $\forall j$, and let $\tilde{W}_2$ be the expected value corresponding to the the probability $p_j = x_j^*$, $\forall j$, where $\mathbf{x}$ is a solution of LP as in section 2.3. Then*

$$\mathrm{Max}(\tilde{W}_1, \tilde{W}_2) \ge \frac{1}{2} \left(\tilde{W}_1 + \tilde{W}_2 \right) \ge \frac{3}{4} Y_{LP}.$$

We may obtain an approximation algorithm of ratio $\frac{3}{4}$ by "mixing" Johnson's probabilistic algorithm with probabilities $p_i = \frac{1}{2}$, $\forall i$, and $p_i = y_i^*$, $\forall i$: Indeed, when selecting a variable X_j, with a probability of $\frac{1}{2}$ choose either of the probability values $p_j = \frac{1}{2}$ or $p_j = f(x_j^*)$, where $f : [0,1] \to [0,1]$ is such that for any feasible solution $(\mathbf{x}, \mathbf{y})$ of LP the following relation holds:

$$\forall i \le m : \ 1 - \prod_{X_j \in C_i} (1 - f(x_j)) \prod_{\overline{X_j} \in C_i} f(x_j) \ge \frac{3}{4} y_i.$$

Let's say that a function $f : [0,1] \to [0,1]$ has the $\frac{3}{4}$-*property* if for any finite sequence of real numbers in $[0,1]$, $\mathbf{x} = (x_1, \ldots, x_k)$ and for any $l \le k$:

$$\forall i \le m : \ 1 - \prod_{j=1}^{l} (1 - f(x_j)) \prod_{j=l+1}^{k} f(x_j) \ge \frac{3}{4} \mathrm{Min} \left(1, \sum_{j=1}^{l} x_j + \sum_{j=l+1}^{k} (1 - x_j) \right).$$

Any function with the $\frac{3}{4}$-property may be used to mix Johnson's algorithms. The following functions are but a few examples having the $\frac{3}{4}$-property:

- any function f satisfying: $1 - 4^{-x} \leq f(x) \leq 4^{-x}$,
- the linear function $f : x \mapsto a + (1-2a)x$, where $a \in \left[2 - \frac{3}{\sqrt[3]{4}}, \frac{1}{4}\right]$, and
- the segment function $f : x \mapsto \begin{cases} \frac{3}{4}x + \frac{1}{4} & \text{if } 0 \leq x \leq \frac{1}{3} \\ \frac{1}{2} & \text{if } \frac{1}{3} \leq x \leq \frac{2}{3} \\ \frac{3}{4}x & \text{if } \frac{2}{3} \leq x \leq 1 \end{cases}$

6 Approximability in the class MaxSNP

A predicate Π is in the class NP if it is logically equivalent to a second order formula $\exists S : \Phi(P, S)$, where S is a variable running over a class of *structures* and Φ is a predicate with no higher order quantifiers.

Indeed, $\Phi(P, S)$ can be assumed of the form $\Phi(P, S) \equiv \forall \mathbf{x} \exists \mathbf{y} : \Psi(P, S, \mathbf{x}, \mathbf{y})$, where Ψ is quantifier free.

For instance, the satisfiability problem SAT can be stated as the formula

$$\exists T \forall c \exists x : (P(c,x) \wedge x \in T) \vee (N(c,x) \wedge x \notin T)$$

where $P(c,x)$ is a predicate asserting that the variable x appears positively in clause c and $N(c,x)$ asserts that the variable x appears negatively in clause c. T stands for a truth assignment for the set of variables. Now, in the case of kSAT, where each clause has a fixed number k of literals, the quantifier "$\exists y$" may be eliminated. Thus kSAT is equivalent to a predicate of the form,

$$\exists T \forall c \bigwedge_{\substack{\{X_{j_1}, \ldots, X_{j_k}\} \in \mathbf{X}^{(k)} \\ \boldsymbol{\epsilon} \in Two^k}} \left(\begin{array}{lll} & \left(\{X_{j_1}^{\epsilon_1}, \ldots, X_{j_k}^{\epsilon_k}\} \subset c\right) & \Rightarrow \\ \bigwedge_{l=1}^{k} (& (\epsilon_l = 0 \Rightarrow X_{j_l} \notin T) & \wedge \\ & (\epsilon_l = 1 \Rightarrow X_{j_l} \in T) &) \end{array} \right).$$

The class SNP (*strict NP*) consists of the predicates $\exists S \forall \mathbf{x} : \Psi(P, S, \mathbf{x}, \mathbf{y})$, where Ψ is quantifier free. SNP is a (proper) class of NP.

For each NP-predicate $\Pi \equiv \forall \mathbf{x} \exists \mathbf{y} : \Psi(P, S, \mathbf{x}, \mathbf{y})$ its *maximization version* is

$$\mathrm{Max}(\Pi) = \underset{S \in \mathrm{Structures}}{\mathrm{Max}} \mathrm{card}\{\mathbf{x} | \exists \mathbf{y} : \Psi(P, S, \mathbf{x}, \mathbf{y})\}.$$

Naturally, MaxNP $= \{\mathrm{Max}(\Pi) | \Pi \in \mathrm{NP}\}$ and MaxSNP $= \{\mathrm{Max}(\Pi) | \Pi \in \mathrm{SNP}\}$.

The notion of Problem Reducibility in MaxSNP should preserve polynomial-time solutions as well as approximability ratios. More precisely, let us say that Π *L-reduces (linearly reduces)* to Π', $\Pi \preceq_L \Pi'$, if there are polynomial-time algorithms f and f' as well as constants $a, b > 0$ such that

- *a bounds ratios of optimality*: For any instances I of Π and I' of Π' the optimal values satisfy the inequality: $opt(I') \leq a \cdot opt(I)$, and
- *b bounds ratios of approximability*: If c' is an approximation to the optimal value $opt(I')$ then f' produces an approximation c of I such that $|c - opt(I)| \leq b|c' - opt(I')|$.

Proposition 6.1 *1. L-reducibility is transitive:* $\Pi \preceq_L \Pi' \wedge \Pi' \preceq_L \Pi'' \Rightarrow \Pi \preceq_L \Pi''$.

2. If $\Pi \preceq_L \Pi'$ *and* Π' *has a PTAA with ratio* r *then* Π *has a PTAA with ratio* $ab \cdot r$.

A problem Π_0 is *MaxSNP-complete* it its in the class MaxSNP and $\forall \Pi \in$ MaxSNP, $\Pi \preceq_L \Pi_0$.

Proposition 6.2 *The following problems are MaxSNP-complete:*

Max3SAT *MaxSAT with length 3 clauses.*

Independent set-B *Find the largest* independent set*, i. e. subgraphs with no edges, in a given graph whose nodes have valency bounded by B.*

Max2SAT *MaxSAT with length 2 clauses.*

MaxCUT *Maximize the weight sum of edges going from one set to another in a partition of the nodes by two sets.*

MaxDirectedCUT *MaxCUT in digraphs.*

An important characteristic property of MaxSNP is given by the next

Proposition 6.3 *Every problem in MaxSNP can be approximated within some fixed ratio.*

Indeed [PY], given an instance of SNP, $\Pi \equiv \exists S \forall \mathbf{x} : \Psi(P, S, \mathbf{x}, \mathbf{y})$, the subformula $\forall \mathbf{x} : \Psi(P, S, \mathbf{x}, \mathbf{y})$ can be written equivalently as a conjunctive form $\bigwedge_{\mathbf{x}} \phi_{\mathbf{x}}$. Let m be the total number of conjuncts, k be the maximum length of the clauses $\phi_{\mathbf{x}}$, and $f_{\mathbf{x}}$ be the ratio of assignments satisfying $\phi_{\mathbf{x}}$ with respect to the total number of assignments, $f_{\mathbf{x}} \geq 2^{-k}$. For any assignment to the variables we will have that it satisfies an average number of conjuncts equal to $\sum_{\mathbf{x}} f_{\mathbf{x}} \geq m \cdot 2^{-k}$.

A procedure similar to Johnson's algorithm for MaxSAT will realize this expected value.

References

[A] Arora, Lund C., Motwani R., Sudan M., Szegedy M.: "Proof verification and hardness of approximation problems", Proceedings of the 33rd IEEE Symposium on Foundations on Computer Science, pp. 14-23, 1992.

[DM] De Ita G., Morales G.: "A note on variations of some MaxSAT algorithms", (unpublished manuscript), 1995.

[F] Feige U., Goldwasser S., Lovász L., Safra S., Szegedy M.: "Approximating clique is almost NP-complete", Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, pp. 2-12, 1991.

[GW] Goemans M., Williamson D.: "New $\frac{3}{4}$-approximation algorithms for maximum satisfiability problem", SIAM Jour. on Discrete Mathematics, vol. 7, nr. 4, pp.656-666, 1994.

[HJ] Hansen P., Jaumard B.: "Algorithms for the maximum satisfiability problem", Computing, vol. 44, pp. 279-303, 1990.

[J] Johnson D.: "Approximation algorithms for combinatorial problems", Jour. of computer and syst. sci. vol. 9, pp. 256-278, 1974.

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

[Y] Yannakakis M.: "On the approximation of Maximum Satisfiability", Journal of Algorithms, vol. 17, pp. 475-502, 1994.