Maximum Satisfiability

Phil Sung

March 17, 2006

The Boolean satisfiability problem (SAT) is the problem of deciding, given a Boolean expression in variables $x_1, \cdots, x_n$, whether some assignment of the variables makes the expression true. SAT is historically notable because it was the first problem proven to be **NP**-complete. (Before this point, the idea of **NP**-completeness had been formulated, but no one had proven that there actually existed any **NP**-complete problems.)

We will consider not arbitrary Boolean expressions but only expressions in *conjunctive normal form* (CNF), i.e. of the form

$$((A_{11} \vee A_{12} \vee \cdots) \wedge (A_{21} \vee A_{22} \vee \cdots) \wedge \cdots)$$

where each *literal* A_{ij} is either a single variable or its negation, and each clause does not contain more than one literal associated with a single variable.

In the MAX-SAT variation of SAT, we do not require that all the disjunctive clauses be satisfied. Instead, we want find an assignment which maximizes the number of satisfied clauses.

1 Maximum Satisfiability

Problem (MAX-SAT)**.** *Let $\{c_i\}$ be a set of Boolean clauses on variables $x_1, \ldots, x_n$ where each clause is a disjunction of literals, each literal being a Boolean variable or its negation. Let there be a nonnegative weight w_c associated with each clause c. Find an assignment of the Boolean variables that maximizes the total weight of the satisfied clauses.*

MAX-SAT is **NP**-complete. Even MAX-2SAT, the restriction to instances in which each clause has at most two literals in it, is **NP**-complete.

1.1 A $1/2$-approximation algorithm

Algorithm 1 (RANDOM)**.** *Set each Boolean variable to be True or False uniformly and independently. Output the resulting assignment.*

Let the random variable W denote the weight of the satisfied clauses, and let W_c be the contribution to W from any particular clause c. Thus we have $W = \sum_c W_c$ and $\mathbf{E}[W_c] = w_c \cdot \Pr[c \text{ is satisfied}]$. For a clause c let $|c|$ be the number of literals in c. Also let $\alpha_k \equiv 1 - 2^{-k}$ for all $k \geq 1$. Observe that $\alpha_k \geq 1/2$.

Lemma 1. *Under RANDOM, any clause c is satisfied with probability $\alpha_{|c|}$.*

Proof. c is satisfied unless all of its literals are false. Each literal is assigned independently and true with probability 1/2, so the probability that all of the literals are false is $2^{-|c|}$. Therefore c is satisfied with probability $1 - 2^{-|c|} = \alpha_{|c|}$. □

Theorem 2. *Under RANDOM,* $\mathbf{E}[W] \geq \frac{1}{2}\text{OPT}$.

Proof. By linearity of expectation, the expected value of the objective function is

$$\mathbf{E}[W] = \sum_c \mathbf{E}[W_c] = \sum_c w_c \alpha_{|c|}. \tag{1}$$

Then we have

$$\sum_c w_c \alpha_{|c|} \geq \sum_c \frac{1}{2} w_c \geq \frac{1}{2}\text{OPT}.$$

The first step follows because $|c| \geq 1$ and $\alpha_k \geq 1/2$ when $k \geq 1$; the second step follows because OPT, the total weight of satisfied clauses, can be no more than the total weight of all clauses. Therefore, RANDOM is a 1/2-approximation algorithm. □

Observe that if the clauses are large, then the probability of satisfying them approaches 1. In particular, if the clause size is lower-bounded by k, then the approximation factor is guaranteed to be α_k in expectation, and this approximation factor grows quickly with k. For example, in MAX-E3SAT, the restriction of MAX-SAT to instances where each clause has *exactly* three literals, RANDOM is a 7/8-approximation algorithm. In fact, no better approximation is possible for this particular problem unless $\mathbf{P} = \mathbf{NP}$.

1.2 A $(1 - 1/e)$-approximation algorithm

We will now solve MAX-SAT with an integer linear program. Declare variables $z_c \in \{0, 1\}$ for each clause and $y_i \in \{0, 1\}$ for each Boolean variable. A y_i which is set to 1 is associated with a True assignment to the corresponding Boolean value; a z_c which is set to 1 is associated with a clause which is satisfied by the assignment. We want to maximize the total weight of the satisfied clauses, subject to the constraint that a clause can only be satisfied if one of its literals is True.

For a clause c, let S_c^+ and S_c^- denote the set of variable indices which appear in c non-negated and negated, respectively. For z_c to be 1, at least one of the terms in S_c^+ must be 1 or one of the terms in S_c^- must be 0. Formulate the ILP as follows; its LP-relaxation can be rounded to give us a good polynomial-time approximation.

$$\begin{aligned} \text{maximize} \quad & \sum_c w_c z_c \\ \text{subject to} \quad & \begin{cases} \forall c: & \displaystyle\sum_{i \in S_c^+} y_i + \sum_{i \in S_c^-} (1 - y_i) \geq z_c \\ \forall c: & z_c \in \{0, 1\} \\ \forall i: & y_i \in \{0, 1\} \end{cases} \end{aligned} \tag{2}$$

Algorithm 2 (LP-RELAX)**.** *Solve the LP-relaxation of* (2)*:*

$$\begin{array}{ll} \textit{maximize} & \displaystyle\sum_c w_c z_c \\ \textit{subject to} & \begin{cases} \forall c: & \displaystyle\sum_{i \in S_c^+} y_i + \sum_{i \in S_c^-} (1 - y_i) \geq z_c \\ \forall c: & 0 \leq z_c \leq 1 \\ \forall i: & 0 \leq y_i \leq 1, \end{cases} \end{array} \tag{3}$$

and denote its solution $(\mathbf{y}^*, \mathbf{z}^*)$*. Independently set each* x_i *to be True with probability* y_i^**.*

Define $\beta_k \equiv 1 - \left(1 - \frac{1}{k}\right)^k$.

Lemma 3. *Under LP-RELAX,* $\mathbf{E}[W_c] \geq \beta_{|c|} w_c z_c^*$ *for any clause* c*.*

Proof. We can assume WLOG that all literals in c are not negated. (If any of them is negated, we can replace it with a new un-negated variable in c and replace its corresponding variable y_m with $y_{m'} = 1 - y_m$ in the LP (3); this has no effect on the LP value or any of the z_c's.) Suppose $c = (x_1 \vee \cdots \vee x_{|c|})$. c is not satisfied iff none of the literals are True. Because x_i is set to True with probability y_i^*, the clause is true with probability

$$\begin{aligned} \Pr[c \text{ is satisfied}] &= 1 - \prod_i (1 - y_i^*) \\ &\geq 1 - \left(\frac{\sum_i (1 - y_i^*)}{|c|} \right)^{|c|} \end{aligned}$$

by the arithmetic-geometric inequality— from $(a_1 a_2 \cdots a_k)^{1/k} \leq (a_1 + \cdots + a_k)/k$, let $a_i = 1 - y_i^*$ and $k = |c|$;

$$\begin{aligned} &= 1 - \left(1 - \frac{\sum_i y_i^*}{|c|} \right)^{|c|} \\ &\geq 1 - \left(1 - \frac{z_c^*}{|c|} \right)^{|c|} \end{aligned}$$

because LP (3) guarantees that $\sum_i y_i^* \geq z_c^*$.

Since $1 - (1 - \frac{z}{|c|})^{|c|}$ is a concave function of z which is equal to 0 at $z = 0$ and $\beta_{|c|}$ at $z = 1$, it must be at least $\beta_{|c|} z$ on $z \in [0, 1]$. Therefore,

$$\Pr[c \text{ is satisfied}] \geq 1 - \left(1 - \frac{z_c^*}{|c|} \right)^{|c|} \geq \beta_{|c|} z_c^*$$

and $\mathbf{E}[W_c] \geq \beta_{|c|} w_c z_c^*$, as desired.

Observe that β_k decreases with k, so if the size of clauses is upper bounded by k, then

$$\mathbf{E}[W] = \sum_c \mathbf{E}[W_c] \geq \sum_c \beta_{|c|} w_c z_c^* \geq \beta_k \sum_c w_c z_c^* = \beta_k \mathrm{OPT}_f \geq \beta_k \mathrm{OPT}.$$

so that LP-RELAX is a β_k-approximation algorithm. When k is unbounded, the guarantee is a factor of $\inf \beta_k = 1 - 1/e$. □

1.3 A 3/4-approximation algorithm

RANDOM and LP-RELAX provide their best bounds on large and small clauses, respectively. In fact, when we combine the two algorithms, we get a better performance bound than either of them alone provides.

Algorithm 3 (LINEAR)**.** *Choose, uniformly and at random, to run either RANDOM or LP-RELAX, and return its Boolean assignment.*

Theorem 4. *LINEAR achieves an approximation factor of* 3/4*.*

Proof. We can condition the expected value of the objective function on which algorithm was chosen:

$$\begin{aligned}\mathbf{E}[W_c] &= \mathbf{E}[W_c|\text{RANDOM}]\Pr[\text{RANDOM}] + \mathbf{E}[W_c|\text{LP-RELAX}]\Pr[\text{LP-RELAX}] \\ &= \frac{1}{2}(\mathbf{E}[W_c|\text{RANDOM}] + \mathbf{E}[W_c|\text{LP-RELAX}]) \\ &= \frac{1}{2}(\alpha_{|c|}w_c + \beta_{|c|}w_c z_c^*) \\ &\geq \frac{\alpha_{|c|}+\beta_{|c|}}{2} w_c z_c^*.\end{aligned}$$

In fact, $\alpha_k + \beta_k = (1-2^{-k}) + (1-(1-\frac{1}{k})^k)$ is at least 3/2 for all values of k. So the expected value of the objective function is

$$\mathbf{E}[W] = \sum_c \mathbf{E}[W_c] \geq \sum_c \frac{3}{4} w_c z_c^* = \frac{3}{4}\text{OPT}_f \geq \frac{3}{4}\text{OPT}$$

as desired. □

This bound is tight, and in fact no algorithm which picks an assignment based only on the LP relaxation can do better. Consider the set of clauses $\{x_1 \vee x_2, x_1 \vee \overline{x_2}, \overline{x_1} \vee x_2, \overline{x_1} \vee \overline{x_2}\}$ and let each clause have unit weight. The LP will set each x_i to be 1/2 and each z_c to be 1 to give $\text{OPT}_f = 4$. But no relaxation can give a value which is better than the true optimum, OPT = 3. We cannot in general show that our rounded assignment is better than $\frac{3}{4}\text{OPT}_f$, and OPT_f is our only upper bound on OPT, so we cannot give better than a 3/4-approximation using the LP relaxation.

1.4 An SDP algorithm for MAX-2SAT

Consider now the restriction of MAX-SAT to instances where each clause has at most two literals (MAX-2SAT). We can also write this as an optimization problem in n variables y_i corresponding to the Boolean variables x_i. Constrain each variable to be -1 (False) or $+1$ (True). Whether a clause is satisfied or not can be written as a function of the variables y_i; for any clause, let $v(c)$ be 1 if c is satisfied and 0 if not. When we restrict the clauses to have only two literals, each clause can be so represented using a quadratic function of the y_i's. To make each term of degree either 2 or 0 (which we will need for our relaxation), we introduce a new variable y_0 and multiply it to any degree-1 terms.

The following identities hold for clauses of one literal:

$$v(x_i) = \frac{1+y_i}{2} \to \frac{1+y_0 y_i}{2} \qquad\qquad v(\overline{x_i}) = \frac{1-y_i}{2} \to \frac{1-y_0 y_i}{2}$$

Algorithm	$\lvert c\rvert = 1$	2	3	4	5	6	(Min)
RANDOM	0.5000	0.7500	0.8750	0.9375	0.9688	0.9844	0.5000
LP-RELAX	1.0000	0.7500	0.7037	0.6836	0.6723	0.6651	0.6321
LINEAR	0.7500	0.7500	0.7894	0.8105	0.8205	0.8247	0.7500
SDP	0.8786	0.8786	0.0000	0.0000	0.0000	0.0000	0.0000
COMBINED	0.7555	0.7555	0.7555	0.7758	0.7854	0.7894	0.7555

Table 1: Expected approximation ratio by clause size

Consider a two-literal clause such as $(x_i \vee \overline{x_j})$. It is true unless both of the literals are not satisfied:

$$\begin{aligned} v(x_i \vee \overline{x_j}) &= 1 - v(\overline{x_i})v(x_j) \\ &= 1 - \frac{1-y_i}{2} \cdot \frac{1+y_j}{2} \\ &= \frac{1+y_i}{4} + \frac{1-y_j}{4} + \frac{1+y_iy_j}{4} \\ &\to \frac{1+y_0y_i}{4} + \frac{1-y_0y_j}{4} + \frac{1+y_iy_j}{4} \end{aligned}$$

We can, of course, write a similar sum for any two-literal clause. In fact $v(c)$ can, for any clause, be written as the sum of multiples of terms of the form $1 \pm y_m y_n$.

We wish to maximize $\sum_c v(c)w_c$ over the y_i's (including y_0); we can make the problem easier by relaxing y_i from ± 1 to the space of unit vectors in $\mathbb{R}^{n+1}$, and replacing the y_i products with dot products. This is the SDP formulation:

$$\begin{aligned} \text{maximize} \quad & \sum_c v(c)w_c = \sum_{i<j} (a_{ij}(1 + \mathbf{y}_i \cdot \mathbf{y}_j) + b_{ij}(1 - \mathbf{y}_i \cdot \mathbf{y}_j)) \\ \text{subject to} \quad & \forall i : \mathbf{y}_i \in \mathbb{R}^{n+1},\ \mathbf{y}_i \cdot \mathbf{y}_i = 1 \end{aligned} \tag{4}$$

The algorithm is similar to the SDP algorithm for MAX-CUT:

Algorithm 4 (SDP)**.** *Solve SDP* (4)*, then pick a hyperplane at random to separates the True variables from the False variables. In particular, given a solution* $\{\mathbf{y}_i^*\}$*, select* $\mathbf{r}$ *uniformly at random from the unit vectors in* $\mathbb{R}^{n+1}$*, and let* x_i *be True if* $\operatorname{sgn}(\mathbf{y}_i^* \cdot \mathbf{r}) = \operatorname{sgn}(\mathbf{y}_0^* \cdot \mathbf{r})$*.*

Intuitively, $\mathbf{y}_0^*$ represents the True value, and we assign a variable to be True if its associated vector falls on the same side of the hyperplane as $\mathbf{y}_0^*$.

Theorem 5. *Under SDP,* $\mathbf{E}[W] \geq 0.87856 \cdot \text{OPT}$*.*

This SDP-based algorithm gives us a 0.87856-approximation for MAX-2SAT; we can adapt it to MAX-SAT by simply discarding all the larger (size ≥ 3) clauses. This algorithm essentially gives us an expected approximation ratio of 0.87856 for clauses of size 1 and 2, and an approximation ratio of 0 for all larger clauses. Nevertheless, by selecting randomly between this and LINEAR, we can improve the approximation ratio. An algorithm, COMBINED, which chooses LINEAR 96% of the time and SDP the rest of the time achieves an approximation ratio of 0.7555.

The best proven ratio for MAX-SAT is 0.7846, and numerical tests on some algorithms suggest a ratio of 0.8331 is possible.

2 The method of conditional expectation

In fact, it is possible to achieve the approximations we showed not just in expectation but in a deterministic way ("derandomizing") using the *method of conditional expectation*. We will demonstrate this method by derandomizing RANDOM, the 1/2-approximation algorithm.

First, observe that we can calculate the expectation of W conditioned on any partial set of assignments to the variables— if a literal is false, then remove it from all the clauses in which it appears; if it is true, then ignore the clauses which contain it, as they are already satisfied. Then the conditional expectation of W is the unconditioned expectation of W (weight of satisfied clauses) in the reduced set of clauses, which is given by (1), plus the weight of the already-satisfied clauses.

We can now write the expected value of W as a weighted average of conditional expectations:

$$\frac{1}{2}\text{OPT} \leq \mathbf{E}[W] = \mathbf{E}[W|x_1]\Pr[x_1] + \mathbf{E}[W|\overline{x_1}]\Pr[\overline{x_1}].$$

Since the left-hand expression is a weighted average of the conditional expectations, one of the conditional expectations must be at least the average; we can calculate which it is because we know how to evaluate the conditional expectations. Suppose for example that $\mathbf{E}[W|x_1] \geq \mathbf{E}[W]$. Then we can write

$$\frac{1}{2}\text{OPT} \leq \mathbf{E}[W|x_1] = \mathbf{E}[W|x_1, x_2]\Pr[x_2] + \mathbf{E}[W|x_1, \overline{x_2}]\Pr[\overline{x_2}].$$

Similarly, we can iterate through all the variables, always choosing the assignment that gives a higher expected W. Eventually we will find some assignment of *all* the variables such that the conditional expectation of W is at least OPT/2. Of course, this the conditional expectation in this case is simply the value of W for that particular assignment, and selecting that assignment get us the desired approximation factor deterministically. This can be done in polynomial time.

We can represent this procedure as walking down a full binary tree of depth n, where each leaf represents the objective value associated with a complete assignment of the variables, and nodes higher up represent conditional expectations associated with less and less complete variable assignments. We start at the root with no variables set and we wish to walk down a path where the conditional expectations. The weight at each node is the average of its children, and at each node all we need to do is calculate the value at the node's children to determine which direction to go.

This strategy is easily extended to LP-RELAX. It can in principle be used with any randomized assignment scheme. However, in practice, calculating the conditional probabilities can be difficult for a general scheme.