E0 249: Approximation Algorithms, Spring 2022 February 21, 2022

Week 8, Lecture 1

SAT

Instructor: Anand Louis *Scribe: Siddhartha Sarkar*

1 Introduction

In this lecture, we will discuss the boolean satisfiability (SAT) problem through which we will illustrate a number of techniques that are useful in many cases. We will look at two algorithms: one that works very well for a certain regime of parameters and the other works well for a completely different regime of parameters. Then we will look at a third algorithm that sort of does the best of both worlds and ends up performing better than both of the previous algorithms. So in today's lecture, we intend to illustrate this idea that how we can get significantly better of two worlds under consideration.

Definition 1 (MAX-SAT problem)**.** *Given a set of boolean variables $x_1, x_2, \ldots, x_n$, a clause is an "OR" of the boolean variables or their negations. Given a set of m clauses, the goal is to compute the assignment that satisfies the largest fraction of clauses.*

Definition 2 (MAX-k-SAT problem)**.** *Given a set of boolean variables $x_1, x_2, \ldots, x_n$, a clause is an "OR" of the boolean variables or their negations. Given a set of m clauses where every clause has at most k ($k \geq 2$) literals, i.e., boolean variables or their negations, the goal is to compute the assignment that satisfies the largest fraction of clauses.*

Essentially MAX-k-SAT is the restricted version of MAX-SAT.

2 First Algorithm: Random Assignment

A simple algorithm for MAX-k-SAT is to independently set x_i to *true* or *false* with probability 1/2, for each $i \in [n]$.

Algorithm 1: Random Assignment

1 For each $i \in [n]$, independently set x_i to *true* or *false* with probability 1/2.

Here the clause C_i is a conjunction of $|C_i|$ literals. A clause $C_i = x_{i_1} \vee \neg x_{i_2} \vee \ldots \vee \neg x_{i_k}$ will be satisfied if at least one of the literals turns out to be *true*. In other words, it will not be satisfied if each of the literals is set to *false*. Therefore,

$$\mathbb{P}[C_i \text{ is satisfied}] = 1 - \frac{1}{2^{|C_i|}}$$

By linearity of expectation, the expected number of clauses that are satisfied is given by

$$\mathbb{E}[\text{number of clauses satisfied}] = \sum_{i \in [m]} \left(1 - \frac{1}{2^{|C_i|}}\right)$$

For the MAX-k-SAT problem, if $|C_i| = k$ for each clause, then,

$$\mathbb{E}[\text{number of clauses satisfied in } k\text{-SAT}] = m \cdot \left(1 - \frac{1}{2^k}\right)$$

Since, the maximum number of clauses satisfied is at most m, i.e., $OPT \leq m$, the above algorithm is a $(1 - \frac{1}{2^k})$-approximation for MAX-k-SAT when each clause has exactly k literals.

In general, $|C_i| \geq 1$ and hence $\frac{1}{2^{|C_i|}} \leq 1/2$. Therefore,

$$\begin{aligned}\mathbb{E}[\text{number of clauses satisfied}] &= \sum_{i\in[m]} \left(1 - \frac{1}{2^{|C_i|}}\right) \\ &\geq \sum_{i\in[m]} \left(1 - \frac{1}{2}\right) = \frac{m}{2}\end{aligned}$$

Therefore, the algorithm above is a $\frac{1}{2}$-approximation for MAX-SAT.

3 Second Algorithm: Randomized LP Rounding

We will write an integer program for the MAX-SAT problem. To do so, let us introduce variable y_i to indicate the value of x_i, i.e.,

$$y_i = \begin{cases} 1 & \text{if } x_i \text{ is } \textit{true} \\ 0 & \text{if } x_i \text{ is } \textit{false} \end{cases}$$

Similarly, introduce variable z_j to indicate whether clause C_j is satisfied. For a clause C_j, let P_j be the set of indices that occur positively and N_j be the set of indices that occur negated. So,

$$C_j = (\vee_{i\in P_j} x_i) \vee (\vee_{i\in N_j} \neg x_i)$$

So, we will add the following constraint for clause C_j in the linear program.

$$z_j \leq \sum_{i\in P_j} y_i + \sum_{i\in N_j} (1 - y_i)$$

Since we want to maximize the number of clauses satisfied, we write the integer linear program as below

$$\max \quad \sum_{j\in[m]} z_j$$

subject to

$$\begin{aligned} z_j &\leq \sum_{i\in P_j} y_i + \sum_{i\in N_j} (1 - y_i) \quad \forall j \in [m] \\ z_j &\in \{0, 1\} \quad \forall j \in [m] \\ y_i &\in \{0, 1\} \quad \forall i \in [n] \end{aligned}$$

An optimal solution to the above integer program gives us an optimal solution to the MAX-SAT problem. We relax the integrality constraints to get the following LP.

$$\max \quad \sum_{j\in[m]} z_j \tag{1}$$

subject to

$$z_j \leq \sum_{i\in P_j} y_i + \sum_{i\in N_j} (1 - y_i) \quad \forall j \in [m] \tag{2}$$

$$0 \leq z_j \leq 1 \quad \forall j \in [m]$$
$$0 \leq y_i \leq 1 \quad \forall i \in [n]$$

We solve the LP relaxation (1). For each i, independently set x_i to *true* with probability y_i. If the optimal solution to the LP happens to be an integer solution, each y_i is either 0 or 1. Then we either set x_i to *true* or set it to *false*. This solution in fact satisfies the maximum number of clauses. This is just a sanity check.

Algorithm 2: Randomized LP Rounding

1 Solve the linear programming relaxation.
2 For each $i \in [n]$, independently set variable x_i to *true* with probability y_i.

Now let us analyze the probability of an arbitrary clause C_j being satisfied. Suppose $|C_j| = k$. As before, since clause C_j is an OR of a number of literals, it will be easier to compute the probability that it is not satisfied.

$$\begin{aligned}
\mathbb{P}[C_j \text{ is not satisfied}] &= \prod_{i \in P_j} (1 - y_i) \prod_{i \in N_j} (1 - (1 - y_i)) \\
&\leq \left(\frac{1}{k} \left(\sum_{i \in P_j} (1 - y_i) + \sum_{i \in N_j} (1 - (1 - y_i)) \right) \right)^k \quad \text{(Applying AM-GM inequality)} \\
&= \left(1 - \frac{1}{k} \left(\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \right) \right)^k \quad \text{(By rearranging the terms)} \\
&\leq \left(1 - \frac{z_j}{k} \right)^k \quad \text{(By constraint (2))}
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\mathbb{P}[C_j \text{ is satisfied}] = 1 - \mathbb{P}[C_j \text{ is not satisfied}] \\
&\geq 1 - \left(1 - \frac{z_j}{k} \right)^k
\end{aligned}$$

While this is a good bound, this is not immediately useful. The expected number of clauses satisfied will be sum of these probabilities over all the clauses j. Then we would like to compare the LP value of (1) with this expected number of clauses. The objective function of the LP is a linear polynomial in the z_j's whereas the bound we have got is a non-linear polynomial in the z_j's. Therefore, they are not easy to be compared. Therefore, we have to find a way to lower bound this quantity by some linear polynomial of the z_j's. That will help us compare the expected number of clauses satisfied with the optimal value of the LP. Here is a lemma that accomplishes the same.

Lemma 3. *For $x \in [0, 1]$,*

$$1 - \left(1 - \frac{x}{k}\right)^k \geq \left(1 - \left(1 - \frac{1}{k}\right)^k \right) x$$

Proof. Consider the function $f(x)$ as below which is concave in $[0, 1]$.

$$f(x) = 1 - \left(1 - \frac{x}{k}\right)^k$$

We have

$$f'(x) = \left(1 - \frac{x}{k} \right)^{k-1}$$

$$f''(x) = -\frac{k-1}{k}\left(1-\frac{x}{k}\right)^{k-2}$$

Therefore $f''(x) \leq 0$ for all $x \in [0,1]$. Thus $f(x)$ is concave in $[0,1]$. Therefore the function does not lie below the line segment joining $(0, f(0))$ and $(1, f(1))$. Therefore,

$$\begin{aligned}
&\frac{f(x)-f(0)}{x-0} \geq \frac{f(1)-f(0)}{1-0} \\
&\implies f(x) \geq x(f(1)-0) \\
&\implies f(x) \geq \left(1-\left(1-\frac{1}{k}\right)^k\right)x
\end{aligned}$$

□

By applying Lemma (3), we can write

$$\mathbb{P}[C_j \text{ is satisfied}] \geq \left(1-(1-\frac{1}{k})^k\right)z_j \geq (1-\frac{1}{e})z_j$$

By linearity of expectation,

$$\begin{aligned}
\mathbb{E}[\text{number of clauses satisfied}] &\geq \sum_{j\in[m]}(1-\frac{1}{e})z_j \\
&= \left(1-\frac{1}{e}\right)OPT_{LP} \\
&\geq \left(1-\frac{1}{e}\right)OPT
\end{aligned}$$

Therefore, the Algorithm (2) is an $\left(1-\frac{1}{e}\right)$-approximation for the MAX-SAT problem. This is already an improvement over the 1/2-approximation we have seen earlier.

4 Third Algorithm: The Best of Both Worlds

Now let us look at the two algorithms more closely. For a clause C_j having k literals, we have

$$\mathbb{P}[\text{Algorithm 1 satisfies } C_j] = \left(1-\frac{1}{2^k}\right)$$

$$\mathbb{P}[\text{Algorithm 2 satisfies } C_j] \geq \left(1-(1-\frac{1}{k})^k\right)z_j$$

Therefore, Algorithm 1 is better for *large* values of k, Algorithm 2 is better for *small* values of k. So what can we do to get the best of both worlds? The following algorithm is surprisingly powerful and does exactly what we intend to do.

Algorithm 3: Combining the two algorithms

1 Choose a random algorithm from {Algorithm 1, Algorithm 2}.
2 Run it on the instance and return its output.

The following equation is crucial and it captures the probability of a clause being satisfied by this algorithm.

$$\begin{aligned}\mathbb{P}[\text{Algorithm 3 satisfies } C_j] &= \frac{1}{2}\mathbb{P}[\text{Algorithm 1 satisfies } C_j] + \frac{1}{2}\mathbb{P}[\text{Algorithm 2 satisfies } C_j] \\ &\geq \frac{1}{2}\left(1 - \frac{1}{2^k}\right) + \frac{1}{2}\left(1 - (1 - \frac{1}{k})^k\right) z_j \\ &\geq \left(\frac{1}{2}\left(1 - \frac{1}{2^k}\right) + \frac{1}{2}\left(1 - (1 - \frac{1}{k})^k\right)\right) z_j \quad \text{(Since } z_j \in [0,1]\text{)}\end{aligned}$$

Now we can analyse the performance of Algorithm (3). The Table 1 summarises the probability of a clause C_j being satisfied.

k	$\left(1 - \frac{1}{2^k}\right)$	$\left(1 - (1 - \frac{1}{k})^k\right)$	$\frac{1}{2}\left(1 - \frac{1}{2^k}\right) + \frac{1}{2}\left(1 - (1 - \frac{1}{k})^k\right)$
1	1/2	1	3/4
2	3/4	3/4	3/4
≥ 3	$\geq 7/8$	$\geq (1 - 1/e)$	$\geq 3/4$

Table 1: Probability of a clause C_j with k literals being satisfied.

Thus, we have

$$\mathbb{P}[\text{Algorithm 3 satisfies } C_j] \geq \frac{3}{4} z_j$$

Therefore,

$$\mathbb{E}[\text{number of clauses satisfied by Algorithm 3}] \geq \frac{3}{4} OPT_{LP} \geq \frac{3}{4} OPT$$

It is noteworthy that choosing Algorithm 1 with probability p and Algorithm 2 with probability $(1-p)$ for some $p \neq 1/2$ does not improve the approximation ratio of Algorithm 3. The current state of the art for the MAX-SAT problem is better than 3/4.

4.1 MAX-2SAT

MAX-2SAT is the restricted version of MAX-SAT where each clause contains at most 2 literals. We have already seen a 3/4-approximation for MAX-2SAT in the previous section. Here, we will improve on that. We write the integer linear program for the MAX-2SAT problem where each clause has exactly 2 literals and the goal is to find a truth assignment of the boolean variables that maximizes the the number of clauses satisfied. For each boolean variable x_i, we consider a variable $y_i \in \{+1, -1\}$. The boolean variable x_i is true if and only if $y_i = +1$. For each clause, one can write a quadratic objective function. For example, the clause $\neg x_a \vee x_b$ is not satisfied if and only if $\frac{1}{4}(1 + y_a)(1 - y_b) = 1$. Therefore, the clause $\neg x_a \vee x_b$ is satisfied if and only if $1 - \frac{1}{4}(1 + y_a)(1 - y_b) = 1$. For each clause of the MAX-2SAT instance there is a corresponding quadratic term that captures whether the clause is satisfied. Therefore, we can write the integer linear program for MAX-2SAT as

$$\max \quad \sum_{j \in [m]} val(C_j)$$

subject to

$$\begin{aligned} y_i^2 &= 1 \quad \forall i \in [n] \\ y_i &\in \{+1, -1\} \quad \forall i \in [n] \end{aligned}$$

where $val(C_j)$ is the corresponding quadratic term for the clause C_j.

A solution to this quadratic program will give an optimal solution to the MAX-2SAT problem. We relax the Integer Linear Program almost the same way as we did for the MAX-CUT Integer Linear Program. We introduce a *one* vector v_0 to play the role of 1 in the objective function. For the clause $C_j = \neg x_a \vee x_b$, we replace 1 by v_0, y_a by vector v_a and y_b by vector v_b, and consider the dot product instead.

$$\begin{aligned} val(C_j) &= 1 - \frac{1}{4}\langle v_0 + v_a, v_0 - v_b\rangle \\ &= 1 - \frac{1}{4}(\langle v_0, v_0\rangle + \langle v_0, v_a\rangle - \langle v_0, v_b\rangle - \langle v_a, v_b\rangle) \\ &= \frac{1}{4}(1 - \langle v_0, v_a\rangle) + \frac{1}{4}(1 + \langle v_0, v_b\rangle) + \frac{1}{4}(1 + \langle v_a, v_b\rangle) \end{aligned}$$

Thus, the value of a 2 literal clause consists of a linear combination of vector terms of the form $(1 + \langle v_i, v_j\rangle)$ or $(1 - \langle v_i, v_j\rangle)$ for $0 \leq i < j \leq n$. We can obtain the vector program relaxation of the ILP as

$$\max \quad \sum_{j \in [m]} val(C_j)$$

subject to

$$\begin{aligned} ||v_i||^2 = 1, \quad & 1 \leq i \leq n \\ ||v_0||^2 = 1 & \\ v_i \in \mathbb{R}^{n+1}, \quad & 0 \leq i \leq n \end{aligned}$$

Here, $val(C_j)$ is the corresponding vector term for the clause C_j. To round the solution of the above program, we apply the Goemans Williamson method as shown below.

Algorithm 4: Goemans Williamson rounding algorithm for MAX-2SAT

1 Sample a random Gaussian $g \sim \mathcal{N}(0,1)^{n+1}$.
2 Let $S_g = \{i \in [n] : \langle v_i, g\rangle$ has the same sign as $\langle v_i, g\rangle\}$.
3 Set all variables in S_g to *true* and variables in $[n] \setminus S_g$ to *false*.

The rounding algorithm is very similar to what we did before in MAX-CUT. We need to analyse the terms in the objective function. The term $(1 - \langle v_0, v_a\rangle)$ gets rounded to 2 with probability θ_{0a}/π and we have $1 - \langle v_0, v_a\rangle = 1 - \cos\theta_{0a}$. The term $(1 + \langle v_0, v_b\rangle)$ gets rounded to 2 with probability $(\pi - \theta_{0b})/\pi$ and we have $1 + \langle v_0, v_b\rangle = 1 + \cos\theta_{0b} = 1 - \cos(\pi - \theta_{0b})$.

Now the same lower bound as in the analysis of the SDP rounding of MAX-CUT is applicable. This gives us an $\alpha_{GW} = 0.878$ approximation for the MAX-2SAT problem. For the MAX-3SAT problem, Håstad [1] proved that it is NP-hard to get a better approximation ratio than 7/8. In fact, it was proved that for MAX-k-SAT, getting a better than $1 - \frac{1}{2^k}$ approximation is NP-hard, where $k \geq 3$. This landmark paper was awarded the Gödel Prize in 2011. Such problems where a random assignment is the best we can do are called approximation resistant problems. So, MAX-k-SAT is an approximation resistant problem. Another example of an approximation resistant problem is MAX-k-XOR. The PCP theorem essentially says that it is NP-hard to distinguish whether an instance is satisfiable or whether one can satisfy only 0.99 fraction of the clauses there.

References

[1] J. Håstad, *Some optimal inapproximability results*, J. ACM **48** (2001), p. 798–859. URL: `https://doi.org/10.1145/502090.502098`, `doi:10.1145/502090.502098`.