ADVANCED ALGORITHMS (I)

CHIHAO ZHANG

1. MaxSAT

In the first part of this course, we will focus on approximation algorithms for **NP**-hard problems. The running example of this lecture is the problem of MaxSAT

MaxSAT	
Input:	A CNF formula $\phi = C_1 \wedge C_2 \cdots \wedge C_m$.
Problem:	Compute an assignment that satisfies maximum number of clauses.

We will propose three (approximation) algorithms for MaxSAT today.

2. Flipping a Coin

The following strategy seems to be stupid: We toss an independent fair coin for each variable and set its value according to the outcome of the toss, i.e., each variable is set to true/false with probability $\frac{1}{2}$.

It turns out that the performance of this simple strategy is not very bad. We assume the variables of the formula ϕ are among $\{x_1, \ldots, x_n\}$. Let X denote the number of clauses satisfied by our algorithm. Note that X is a random variables since we are tossing coins. Then by the linearity of expectation,

$$\mathbf{E}[X] = \sum_{i=1}^{m} \mathbf{Pr}[C_i \text{ is satisfied}] = \sum_{i=1}^{m} \left(1 - 2^{-\ell_i}\right) \geq \frac{m}{2}, \tag{1}$$

where ℓ_i is the *length* of the ith clause, i.e., the number of distinct variables appearing in C_i.

We want to show that $\mathbf{E}[X]$ is not too bad comparing to the optimal solution. Therefore, we need to establish an *upper bound* for the optimal solution.

A trivial upper bound in our case is that

$$\mathbf{OPT} \leq m. \tag{2}$$

Therefore, we obtain $\mathbf{E}[X] \geq \frac{\mathbf{OPT}}{2}$ by combining (1) and (2).

Can we improve this? The worst case of the above analysis is when $\ell_i = 1$ for some C_i. We have the following observation:

- if for some variable x, only one of x or $\bar{x}$ appeared as a clause, say x, then we can increase the probability that x is set to true when tossing the coin;
- if both x and $\bar{x}$ appeared as clauses, then we can improve the upper bound (2), since these two clauses cannot be both satisfied in any assignment.

Therefore, we can assume without loss of generality that we have more positive singleton clauses than negative singleton clauses, i.e., more clauses of the form $C = x$ than $C = \bar{y}$. Consider the collection of variables x such that both x and $\bar{x}$ are clauses, let

$$S = \{x \in \{x_1, \ldots, x_n\} \,:\, \text{both } x \text{ and } \bar{x} \text{ are clauses}\},$$

and $t = |S|$, then we have a new upper bound for **OPT**:

$$\mathbf{OPT} \leq m - t \tag{3}$$

If we use C to denote the set of all clauses of ϕ, then by our assumption, every clause in $C' \triangleq C \backslash (\{x : x \in S\} \cup \{\bar{x} : x \in S\})$ is either positive singleton or contains at least two literals. Now we can toss an independent unfair coin with probability $p \geq \frac{1}{2}$ to true for each variable, then

$$\mathbf{E}[X] = t + \sum_{C \in C'} \mathbf{Pr}[C \text{ is satisfied}] \geq t + (m - 2t) \min\left\{p, 1 - p^2\right\}. \tag{4}$$

The first t in (4) is because exact t clauses in $C \setminus C'$ are satisfied in *any* assignment. For the remaining $m - 2t$ clauses $C \in C'$: (1) if it is a singleton, then it is satisfied with probability p; (2) otherwise, the worst case is when $C = \bar{y} \vee \bar{z}$ for some variables y and z, then it is satisfied with probability $1 - p^2$.

Combining (3) and (4), we obtain

$$\mathbf{E}[X] \geq t + (\mathbf{OPT} - t) \min\left\{p, 1 - p^2\right\} \geq \min\left\{p, 1 - p^2\right\} \cdot \mathbf{OPT}.$$

Therefore, we can optimize $\min\left\{p, 1 - p^2\right\}$ by letting p satisfy $p = 1 - p^2$ to obtain

$$\mathbf{E}[X] \geq \alpha \cdot \mathbf{OPT}$$

where $\alpha \approx 0.618$.

3. Clever Coins via Linear Programming

In previous two algorithms, we toss identical coins for each variable. This might not be optimal since intuitively, for those variables whose appearances are mostly positive, we prefer to set it true. This motivates us to use different coins for each variable. It is not an easy task to decide the probability of each coin, we now use *linear programming* to help us to make the decision.

The first step is to write an instance of MaxSAT as an instance of *integer programming*. For every $i \in [n]$, we introduce a variable y_i indicating whether the variable x_i in ϕ is set true; for every $j \in [m]$, we introduce a variable z_j indicating whether the clause C_j is satisfied. Then MaxSAT is equivalent to

$$\begin{aligned}
\max \quad & \sum_{j=1}^{m} z_j \\
\text{subject to} \quad & \sum_{i \in P_j} y_i + \sum_{k \in N_j} (1 - y_k) \geq z_j, \quad \forall j \in [m] \text{ s.t. } C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{k \in N_j} \bar{x}_k \\
& z_j \in \{0, 1\}, \quad \forall j \in [m] \\
& y_i \in \{0, 1\}, \quad \forall i \in [n]
\end{aligned}$$

The above is nothing but reformulate the problem as an integer programming instances, and it is of course **NP**-hard to solve. The difficulty comes from the non-linear constraints $y_i \in \{0, 1\}$ and $z_j \in \{0, 1\}$. We can *relax* these constraints and obtain the following *linear program*.

$$\begin{aligned}
\max \quad & \sum_{j=1}^{m} z_j \\
\text{subject to} \quad & \sum_{i \in P_j} y_i + \sum_{k \in N_j} (1 - y_k) \geq z_j, \quad \forall j \in [m] \text{ s.t. } C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{k \in N_j} \bar{x}_k \\
& 0 \leq z_j \leq 1, \quad \forall j \in [m] \\
& 0 \leq y_i \leq 1, \quad \forall i \in [n]
\end{aligned}$$

Linear programming can be solved in polynomial-time. Therefore, we can obtain an optimal solution $\left\{y_i^*\right\}_{i \in [n]}, \left\{z_j^*\right\}_{j \in [m]}$ of the LP. Although the LP is not equivalent to MaxSAT, it is close to. Therefore, the optimal solution of the LP is informative. Ideally, y_i^* indicates how likely the variable x_i is set to true in the optimal solution. Therefore, we toss a coin with probability y_i^* to true for every variable x_i.

A typical upper bound for LP based algorithms is the optimal solution of the relaxed linear program. Since the constrinats of the linear program is weaker than the integer program, we have

$$\mathbf{OPT} \le \mathbf{OPT}(\mathrm{LP}) = \sum_{j=1}^{m} z_j^*. \tag{5}$$

On the otherhand, we have for every $C_j = \bigvee_{i\in P_j} x_i \vee \bigvee_{k\in N_j} \bar{x}_k$,

$$\begin{aligned}
\mathbf{Pr}\left[C_j \text{ is } \textit{not} \text{ satisfied}\right] &= \prod_{i\in P_j}(1-y_i^*)\prod_{k\in N_j} y_k^* \\
&\overset{①}{\le} \left(\frac{1}{\ell_j}\left(\sum_{i\in P_j}(1-y_i^*)+\sum_{k\in N_j} y_k^*\right)\right)^{\ell_j} \\
&= \left(\frac{1}{\ell_j}\left(\ell_j - \left(\sum_{i\in P_j} y_i^* + \sum_{k\in N_j}(1-y_k^*)\right)\right)\right)^{\ell_j} \\
&\overset{②}{\le} \left(1-\frac{z_j^*}{\ell_j}\right)^{\ell_j},
\end{aligned}$$

where ① uses *the inequality of arithmetic and geometric means* and ② follows from the fact that z_j^* and y_i^*s satisfy the corresponding constraints in the LP.

Therefore, we obtain

$$\mathbf{E}\left[X\right] = \sum_{j=1}^{m} \mathbf{Pr}\left[C_j \text{ is satisfied}\right] \ge \sum_{j=1}^{m}\left(1-\left(1-\frac{z_j^*}{\ell_j}\right)^{\ell_j}\right). \tag{6}$$

Remember that we need to compare (6) with the upper bound (5), i.e., a linear function of z_j^*. This motivates us to lower bound (6) with a linear function. Let $h(z) \triangleq 1-\left(1-\frac{z}{\ell}\right)^{\ell}$, then it is easy to verify that the function $h(\cdot)$ is concave in $[0,1]$[1] for any integer $\ell \ge 1$. Thus we have $h(z) \ge h(1)\cdot z$ and

$$\mathbf{E}\left[X\right] \ge \sum_{j=1}^{m}\left(1-\left(1-\frac{1}{\ell_j}\right)^{\ell_j}\right) z_j^* \ge \left(1-e^{-1}\right)\sum_{j=1}^{m} z_j^* \ge \left(1-\frac{1}{e}\right)\cdot \mathbf{OPT}.$$

4. Remark

The main reference of this lecture is [WS11, Chapter 5].

References

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011. 3

[1] $h''(z) < 0$ when $z \in [0,1]$