AI2615 算法设计与分析 2020-2021 春季学期

Lecture 13 – Max-SAT and Max-Cut

2021 年 5 月 21 日

Lecturer: 张驰豪 *Scribe:* 陶表帅

1 Max-SAT

Consider a CNF formula

$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m.$$

We will use ℓ_i to denote the number of the literals in the i-th clause. We will use 1 and 0 to represent the Boolean value "true" and "false". We will use m and n to denote the number of clauses and the number of variables respectively. In addition, we assume that each clause does not contain a variable and its negation without loss of generality. In particular, if a clause contains both x_i and $\bar{x}_i$, we know this clause is always satisfied, and we can just remove this clause. Finally, we assume that ϕ does not contain two identical clauses.

There is a polynomial time algorithm for deciding whether ϕ has a satisfying assignment if we have $\ell_i \le 2$ for each $i = 1, \ldots, m$, while this decision problem is NP-complete even if $\ell_i = 3$ for each $i = 1, \ldots, m$. In this section, we consider the following maximization problem.

Problem 1 (Max-SAT). Given a CNF formula ϕ, find an assignment that satisfies as many clauses as possible.

Max-SAT is NP-hard even if $\ell_i \le 2$ for each $i = 1, \ldots, m$. In this section, we will learn four approximation algorithms for max-SAT. We will use `OPT` to denote the maximum number of clauses that can be satisfied by a certain assignment, i.e., the optimal solution.

1.1 Algorithm 1: Uniformly at Random Assignments

Suppose, for each variable x_i, we assign $x_i = 1$ with probability 0.5 and assign $x_i = 0$ otherwise. That is, we flip a fair coin to decide the value for each variable x_i. Suppose the coin flips for the variables are independent. For each clause C_i that contains ℓ_i literals, it is satisfied with probability $1 - 2^{-\ell_i}$. To see this, each literal evaluates to false with probability 0.5, so the probability that all the literals are false is $2^{-\ell_i}$.

Let $\sigma \in \{0,1\}^n$ be a random assignment sampled from the distribution described above. Let $v(\sigma)$ be the number of the clauses satisfied by σ. We calculate the expected number of satisfied clauses by the linearity

of expectation:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\sigma}[v(\sigma)] &= \mathop{\mathbb{E}}_{\sigma}\left[\sum_{i=1}^{m} \mathbf{1}\left(C_i \text{ is satisfied by } \sigma\right)\right] \\
&= \sum_{i=1}^{m} \mathop{\mathbb{E}}_{\sigma}\left[\mathbf{1}\left(C_i \text{ is satisfied by } \sigma\right)\right] && \text{(linearity of expectation)} \\
&= \sum_{i=1}^{m} \Pr\left(C_i \text{ is satisfied by } \sigma\right) \\
&= \sum_{i=1}^{m} \left(1 - 2^{-\ell_i}\right) && \text{(as we have just seen)} \\
&\geq \frac{1}{2}m. && \text{(since } \ell_i \geq 1 \text{ for each } i\text{)}
\end{aligned}$$

Therefore, the expected number of satisfied clauses is at least $\frac{1}{2}m$. This implies that there exists an assignment that satisfies at least $\frac{1}{2}m$ clauses (think about this: if the average height of the students in a class is 1.7 meters, there must exist a student with height at least 1.7 meters).

On the other hand, we know that $\texttt{OPT} \leq m$ for sure. Therefore, the assignment σ that satisfies at least $\frac{1}{2}m$ clauses is a 0.5-approximation:

$$v(\sigma) \geq \frac{1}{2}m \geq \frac{1}{2}\texttt{OPT}.$$

Now, the only remaining problem is, how to find such a σ? This will use a common technique based on *conditional expectation*.

Consider the assignment for the variable x_1. We have

$$\begin{aligned}
\frac{1}{2}m &\leq \mathop{\mathbb{E}}_{\sigma}[v(\sigma)] \\
&= \mathop{\mathbb{E}}_{x_1 \sim \{0,1\}}\left[\mathop{\mathbb{E}}_{\sigma}[v(\sigma) \mid x_1]\right] \\
&= \Pr(x_1 = 1)\mathop{\mathbb{E}}_{\sigma}[v(\sigma) \mid x_1 = 1] + \Pr(x_1 = 0)\mathop{\mathbb{E}}_{\sigma}[v(\sigma) \mid x_1 = 0] \\
&= \frac{1}{2}\left(\mathop{\mathbb{E}}_{\sigma}[v(\sigma) \mid x_1 = 1] + \mathop{\mathbb{E}}_{\sigma}[v(\sigma) \mid x_1 = 0]\right).
\end{aligned}$$

This implies at least one of $\mathbb{E}_\sigma[v(\sigma) \mid x_1 = 1]$ and $\mathbb{E}_\sigma[v(\sigma) \mid x_1 = 0]$ is greater than or equal to $\frac{1}{2}m$. On the other hand, we can evaluate $\mathbb{E}_\sigma[v(\sigma) \mid x_1 = 1]$ and $\mathbb{E}_\sigma[v(\sigma) \mid x_1 = 0]$ in polynomial time. In fact, by substitute $x_1 = 1$ or $x_1 = 0$ into ϕ, we obtain a CNF formula with $n - 1$ variables, and we can use the same method above to calculate the expected number of satisfied clauses. We will finalize the assignment to x_1 such that the expected number of satisfied clauses is at least $\frac{1}{2}m$.

After deciding the assignment to x_1, we can iteratively decide the assignments to $x_2, x_3, \ldots, x_n$. Throughout this process, we can always make sure the expected number of satisfied clauses is at least $\frac{1}{2}m$. When all the variables' assignments are finalized, the number of satisfied clauses is determined, and is at least $\frac{1}{2}m$. Thus, this gives a polynomial time 0.5-approximation algorithm.

This conditional expectation technique can be applied in general to convert an expectation observation to a deterministic algorithm.

If we have $\ell_i = 3$ for all i, we have $1 - 2^{-\ell_i} = \frac{7}{8}$, and the algorithm above gives a $\frac{7}{8}$-approximation. This is in fact the best we can do (assuming P $\neq$ NP). Håstad [2001] proved that the max-3-SAT problem is NP-hard to approximate to within factor $(\frac{7}{8} + \varepsilon)$ for any constant $\varepsilon > 0$.

However, for general max-SAT problem without $\ell_i = 3$, we can do better than a 0.5-approximation. We will see some approximation algorithms with better approximation guarantees.

1.2 Algorithm 2: Flipping Biased Coins

When analyzing the expected number of satisfied clauses, we have used the following lower bound:

$$1 - 2^{-\ell_i} \geq \frac{1}{2}.$$

This is quite a loose bound. In fact, the bound is only tight for $\ell_i = 1$, i.e., the clause C_i is a *singleton* that contains only one literal. If we have $\ell_i \geq 2$, we have a much better lower bound $\frac{3}{4}$ for $1 - 2^{-\ell_i}$. This gives us an intuition that those singletons are the bottleneck for improving the approximation guarantee when we are using this random assignment technique.

A natural idea for improving Algorithm 1 is to use a biased coin for each variable x_i, with the probability based on the occurrences of x_i and $\bar{x}_i$ in ϕ. For example, if we have a singleton clause x_i and a clause $\bar{x}_j \vee x_i$, increasing the probability for $x_i = 1$ will increase the satisfying probability for each of the two clauses, which is beneficial. As another example, if we have a singleton clause x_i and a singleton clause $\bar{x}_i$, we know that exactly one of the two clauses is satisfied regardless of the assignment to x_i. In this case, we know that $\texttt{OPT} \leq m - 1$ since at least one clause is not satisfied, and the approximation guarantee can be improved based on this tighter upper bound for $\texttt{OPT}$.

Let $S = \{x_i \mid \text{both } x_i \text{ and } \bar{x}_i \text{ appear as singleton clauses}\}$, and let $t = |S|$. Our first observation is $\texttt{OPT} \leq m - t$. We rewrite ϕ as

$$\phi = \left[\bigwedge_{x_i \in S} (x_i \wedge \bar{x}_i) \right] \wedge C'.$$

We know that the number of satisfied clauses in the first part $\bigwedge_{x_i \in S} (x_i \wedge \bar{x}_i)$ is exactly t, regardless of how we assign values to variables in S.

For singletons in C', if we know some x_i appears in C', we know $\bar{x}_i$ is not in C' (for otherwise $x_i \in S$); if we know $\bar{x}_i$ appears in C', we know x_i is not in C'. Let

$$z_i = \begin{cases} \bar{x}_i & \text{if } \bar{x}_i \text{ appears in } C' \\ x_i & \text{otherwise} \end{cases},$$

and we substitute all x_i's by z_i's. After this, all the singletons in C' are not negations. Remembering that the singletons are the bottleneck, intuitively, increasing the probability for $z_i = 1$ is more beneficial. Let p be the probability for $z_i = 1$.

Now, we evaluate the probability that each clause $C_j = \left(\bigvee_{i \in P_j} z_i\right) \vee \left(\bigvee_{k \in N_j} \bar{z}_k\right)$ is satisfied, where P_j is the set of indices for those variables who appear as literals in C_j and N_j is the set of indices for those variables whose negations appear as literals in C_j. For each singleton clause, it is satisfied with probability p. For each non-singleton clause, it is satisfied with probability

$$1 - \left(\prod_{i \in P_j} (1-p)\right) \cdot \left(\prod_{k \in N_j} p\right) = 1 - (1-p)^{|P_j|} p^{|N_j|}.$$

For non-singletons, this is minimized when $|P_j| = 0$ and $|N_j| = 2$, and the satisfying probability for each non-singleton clause is at least $1 - p^2$. Putting these together, the probability that a clause is satisfied is at least $\min\{1-p^2, p\}$, and this lower bound is tightest for $p = \frac{\sqrt{5}-1}{2} \approx 0.618$.
By the same calculation in the previous section, the expected number of satisfied clauses in C' is at least $0.618(m-2t)$ (notice that C' has $m-2t$ clauses). Thus, by $\texttt{OPT} \leq m - t$, we have

$$\mathop{\mathbb{E}}_{\sigma}[v(\sigma)] \geq t + 0.618 \cdot (m-2t) \geq t + 0.618 \cdot (\texttt{OPT} - t) \geq 0.618 \cdot \texttt{OPT}.$$

By the same conditional expectation technique in the previous section, this observation can be converted to a polynomial time 0.618-approximation. To see this for one more time, we have

$$0.618(m-2t) \leq p \mathop{\mathbb{E}}_{\sigma}[v(\sigma) \mid z_1 = 1] + (1-p) \mathop{\mathbb{E}}_{\sigma}[v(\sigma) \mid z_1 = 0],$$

and we can conclude that at least one of $[v(\sigma) \mid z_1 = 1]$ and $\mathbb{E}_\sigma[v(\sigma) \mid z_1 = 0]$ is greater than or equal to $0.618(m-2t)$ (otherwise, the right-hand side of the inequality is strictly less than $p \cdot 0.618(m-2t) + (1-p) \cdot 0.618(m-2t) = 0.618(m-2t)$, which invalidates the inequality). We find the assignment for z_1 that yields the larger conditional expectation, and we subsequently decide each of $z_2, z_3, \ldots, z_n$ one-by-one.

1.3 Algorithm 3: Heterogeneous Probabilities

We have used the same probability p for all the variables in Algorithm 2. A natural way to improve the approximation guarantee is to use different, customized probabilities for different variables. We will use the same technique of Integer Program formulation and Linear Program relaxation in the previous lectures. We will use y_i to represent a variable x_i and z_j to represent a clause C_j. The max-SAT problem can be formulated by the following integer program:

$$\begin{aligned} \max \quad & \textstyle\sum_{j=1}^{m} z_j \\ \text{subject to} \quad & \forall j \in [m] : \textstyle\sum_{i \in P_j} y_i + \sum_{k \in N_j} (1 - y_k) \geq z_j \\ & \forall i \in [n] : y_i \in \{0, 1\} \\ & \forall j \in [m] : z_j \in \{0, 1\} \end{aligned} \tag{1}$$

Recall in previous lectures that solving an integer program is NP-hard while solving a linear program can be done in polynomial time, and a common technique is the linear program relaxation. We relax this

integer program to the following linear program:

$$\begin{aligned} \max \quad & \textstyle\sum_{j=1}^{m} z_j \\ \text{subject to} \quad & \forall j \in [m] : \textstyle\sum_{i \in P_j} y_i + \sum_{k \in N_j} (1 - y_k) \geq z_j \\ & \forall i \in [n] : y_i \in [0,1] \\ & \forall j \in [m] : z_j \in [0,1] \end{aligned} \tag{2}$$

and let $\{y_i^*\}_{i \in [n]} \cup \{z_j^*\}_{j \in [m]}$ be its optimal solution. Let $\texttt{OPT}(IP)$ be the optimal objective value for the integer program, which equals to the optimal number of satisfied clauses $\texttt{OPT}$. Let $\texttt{OPT}(LP)$ be the optimal objective value for the linear program. We have $\texttt{OPT}(IP) \leq \texttt{OPT}(LP)$, since the linear program has a larger feasible region than the integer program.

A natural way to interpret each y_i^* is to view it as the probability that x_i is assigned 1. In the probability distribution defined by this, for each clause C_j, we have

$$\begin{aligned} \Pr\big(C_j \text{ is not satisfied}\big) &= \left(\prod_{i \in P_j} (1 - y_i^*)\right)\left(\prod_{k \in N_j} y_k^*\right) \\ &\leq \left(\frac{1}{\ell_j}\left(\sum_{i \in P_j} (1 - y_i^*) + \sum_{k \in N_j} y_k^*\right)\right)^{\ell_j} && \text{(by the AM-GM inequality)} \\ &= \left(\frac{1}{\ell_j}\left(\ell_j - \left(\sum_{i \in P_j} y_i^* + \sum_{k \in N_j} (1 - y_k^*)\right)\right)\right)^{\ell_j} \\ &\leq \left(1 - \frac{z_j^*}{\ell_j}\right)^{\ell_j}. && \text{(by the constraint } \textstyle\sum_{i \in P_j} y_i + \sum_{k \in N_j} (1 - y_k) \geq z_j\text{)} \end{aligned}$$

Therefore, we have

$$\begin{aligned} \mathop{\mathbb{E}}_{\sigma}[v(\sigma)] &\geq \sum_{j=1}^{m}\left(1 - \left(1 - \frac{z_j^*}{\ell_j}\right)^{\ell_j}\right) \\ &\geq \sum_{j=1}^{m} z_j^*\left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right) && (\dagger) \\ &= \texttt{OPT}(LP) \cdot \sum_{j=1}^{m}\left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right) \cdot \\ &\geq \texttt{OPT} \cdot \sum_{j=1}^{m}\left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right) \\ &\geq \left(1 - \frac{1}{e}\right)\texttt{OPT}. \end{aligned}$$

To see the step (†), notice that the function $h(z) = 1 - (1 - \frac{z}{\ell})^{\ell}$ defined on $[0,1]$ is concave, so the curve for $h(z)$ is above the line passing through the two points $(0, h(0))$ and $(1, h(1))$. The expression for this line is $\left(1 - (1 - \frac{1}{\ell})^{\ell}\right) z$.

Therefore, we have seen that the expected number of the satisfied clauses is at least $(1 - 1/e)\texttt{OPT} \approx 0.632 \cdot \texttt{OPT}$. By the conditional expectation technique, we have a polynomial time 0.632-approximation algorithm.

1.4 Algorithm 4: Combining Algorithm 1 and Algorithm 3

In fact, this approximation guarantee can be further improved. Our random assignment in Algorithm 1 guarantees that

$$\mathop{\mathbb{E}}_{\sigma}[v(\sigma)] \geq \sum_{i=1}^{m}\left(1-2^{-\ell_i}\right),$$

and our random assignment in Algorithm 3 guarantees that

$$\mathop{\mathbb{E}}_{\sigma}[v(\sigma)] \geq \sum_{i=1}^{m}\left(1-\left(1-\frac{1}{\ell_i}\right)^{\ell_i}\right)z_j^*.$$

The intuition here is that, for Algorithm 1, the bottleneck comes from those singletons, while for Algorithm 3, the bottleneck comes from the opposites—the clauses with large numbers of literals.

Based on this intuition, we can just implement both Algorithm 1 and Algorithm 3 and take the better solution. Let σ_1 and σ_3 be the random assignments from Algorithm 1 and Algorithm 3 respectively. We have

$$\begin{aligned}
\mathbb{E}[\max\{v(\sigma_1), v(\sigma_3)\}] &\geq \mathbb{E}\left[\frac{1}{2}v(\sigma_1)+\frac{1}{2}v(\sigma_3)\right] \\
&= \frac{1}{2}\left(\mathbb{E}[\sigma_1]+\mathbb{E}[\sigma_3]\right) \\
&\geq \frac{1}{2}\left(\sum_{i=1}^{m}\left(1-2^{-\ell_i}\right)+\sum_{i=1}^{m}\left(1-\left(1-\frac{1}{\ell_i}\right)^{\ell_i}\right)z_i^*\right) \\
&\geq \frac{1}{2}\sum_{i=1}^{m} z_i^*\left(\left(1-2^{-\ell_i}\right)+\left(1-\left(1-\frac{1}{\ell_i}\right)^{\ell_i}\right)\right) \\
&\geq 0.75\sum_{i=1}^{m} z_i^* \qquad \text{(believe Chihao that this is correct!)} \\
&= 0.75\cdot \texttt{OPT}(LP) \geq 0.75\cdot \texttt{OPT},
\end{aligned}$$

which implies this new algorithm is a 0.75-approximation.

2 Max-Cut

Problem 2 (Max-Cut). Given an undirected graph $G(V,E)$, find a partition $\{L,R\}$ of V such that the number of edges between L and R, $E(L,R)$, is maximized.

This problem is similar to the min-cut problem in the previous lectures, but this time, we are maximizing the number of edges in between. We have seen that the min-cut problem can be solved in polynomial time, by solving the max-flow problem. However, the max-cut problem is NP-hard. In this section, we will learn an elegant approximation algorithm to this problem.

The very first idea may be formulating the problem by an integer program, and then relax it to a linear program. Suppose we have a variable x_u for each vertex u, such that $x_u = 0$ if $u \in L$ and $x_u = 1$ if $u \in R$.

In this case, for each edge $\{u, v\}$, we earn 1 if the values of x_u and x_v are different. This is represented as $x_u \oplus x_v$. However, an x-or operator is difficult to be interpreted as a linear operator.
An alternative way to do this is to have $x_u \in \{-1, 1\}$. Then we are going to maximize the objective $\sum_{\{u,v\}\in E} \frac{1}{2}(1 - x_u x_v)$. This gives us the following quadratic integer program:

$$\begin{aligned} \max \quad & \textstyle\sum_{\{i,j\}\in E} \frac{1}{2}(1 - x_i x_j) \\ \text{subject to} \quad & \forall i \in V : x_i \in \{-1, 1\} \end{aligned} \tag{3}$$

Then we relax x_i such that $x_i \in [-1, 1]$. This gives us a *quadratic program*:

$$\begin{aligned} \max \quad & \textstyle\sum_{\{i,j\}\in E} \frac{1}{2}(1 - x_i x_j) \\ \text{subject to} \quad & \forall i \in V : x_i \in [-1, 1] \end{aligned} \tag{4}$$

However, the objective here is quadratic and non-convex. Solving a quadratic program is still NP-hard.
We will learn a different way to relax (3), by relaxing x_i to a n-dimensional vector, so that we have a vector program:

$$\begin{aligned} \max \quad & \textstyle\sum_{\{i,j\}\in E} \frac{1}{2}\left(1 - x_i^\top x_j\right) \\ \text{subject to} \quad & \forall i \in V : x_i^\top x_i = 1 \end{aligned} \tag{5}$$

This vector program is clearly a relaxation of (3): if we restrict each vector x_i such that the first entry must be either 1 or -1 and all the remaining entries must be 0, we have exactly (3).
This vector program is a special case of a *positive semi-definite program*. Before we introduce this, let us review some linear algebra basics.

2.1 Linear Algebra Basics

Definition 3. A symmetric square matrix A is *positive semi-definite*, denoted by $A \succeq 0$, if $x^\top A x \geq 0$ for all $x \in \mathbb{R}^n$.

We prove the following proposition.

Proposition 4. *The followings are equivalent:*

1. $A \succeq 0$.
2. *All the eigenvalues of A are non-negative.*
3. *There exists $U = [u_1 \quad u_2 \quad \cdots \quad u_n]$ where $u_i \in \mathbb{R}^n$ such that $A = U^\top U$.*

To prove this, we make use of the following important theorem.

Theorem 5 (Spectral Decomposition Theorem). *An $n \times n$ symmetric matrix has n real eigenvalues $\lambda_1, \ldots, \lambda_n$ with the corresponding eigenvectors $v_1 \ldots, v_n$ which are orthonormal. Moreover, we have*

$$A = V \Lambda V^\top = \sum_{i=1}^{n} \lambda_i v_i v_i^\top,$$

where

$$V = [v_1 \quad v_2 \quad \cdots \quad v_n] \qquad \textit{and} \qquad \Lambda = \begin{bmatrix} \lambda_1 & & & & \\ & \lambda_2 & & \mathbf{0} & \\ & & \lambda_3 & & \\ & \mathbf{0} & & \ddots & \\ & & & & \lambda_n \end{bmatrix}.$$

Now we can prove Proposition 4.

Proof of Proposition 4. ($1 \implies 2$) Suppose $A \succeq 0$. For each eigenvalue λ with eigenvector v, by the definition of positive semi-definite matrices, we have $0 \le v^\top A v = v^\top \lambda v = \lambda \|v\|^2$, which implies $\lambda \ge 0$ (notice that an eigenvector v cannot be a zero vector, so $\|v\|^2 > 0$).
($2 \implies 3$) Suppose all eigenvalues of A are non-negative. By Theorem 5, we have $A = V\Lambda V^\top = (V\sqrt{\Lambda})(V\sqrt{\Lambda})^\top$, where

$$\sqrt{\Lambda} := \begin{bmatrix} \sqrt{\lambda_1} & & & & \\ & \sqrt{\lambda_2} & & \mathbf{0} & \\ & & \sqrt{\lambda_3} & & \\ & \mathbf{0} & & \ddots & \\ & & & & \sqrt{\lambda_n} \end{bmatrix}.$$

We then have $A = U^\top U$ where $U = (V\sqrt{\Lambda})^\top$.
($3 \implies 1$) Suppose there exists $U = [u_1 \quad u_2 \quad \cdots \quad u_n]$ where $u_i \in \mathbb{R}^n$ such that $A = U^\top U$. For any $x \in \mathbb{R}^n$, we have $x^\top A x = x^\top U^\top U x = (Ux)^\top (Ux) = \|Ux\|^2 \ge 0$, which implies $A \succeq 0$. □

2.2 Positive Semi-Definite Program

The general form of a positive semi-definite program is as follows:

$$\begin{aligned} \max \quad & C \cdot X \\ \text{subject to} \quad & A_k \cdot X \le b_k \qquad k = 1, \ldots, m \\ & X \succeq 0 \end{aligned} \tag{6}$$

where $\cdot$ is the Frobenius inner product.
A positive semi-definite program can be solved in polynomial time. It is more general than a linear program. For example, the linear program

$$\begin{aligned} \max \quad & 2x - 3y \\ \text{subject to} \quad & x + y \le 2 \\ & 3x - y \le 1 \\ & x \ge 0, y \ge 0 \end{aligned} \tag{7}$$

can be rewritten as the following positive semi-definite program

$$\begin{aligned} \max \quad & \begin{bmatrix} 2 & 0 \\ 0 & -3 \end{bmatrix} \cdot \begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix} \leq 2 \\ & \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix} \leq 1 \\ & \begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix} \succeq 0 \end{aligned} \tag{8}$$

A positive semi-definite program is equivalent to a vector program. The general form (6) can be rewritten as the following general form vector program:

$$\begin{aligned} \max \quad & \textstyle\sum_{1 \leq i,j \leq n} C(i,j) u_i^\top u_j \\ \text{subject to} \quad & \textstyle\sum_{1 \leq i,j \leq n} A_k(i,j) u_i^\top u_j \leq b_k \\ & \forall i : u_i \in \mathbb{R}^n \end{aligned} \tag{9}$$

Here, we have decomposed the positive semi-definite matrix X to $X = U^\top U$, where $U = [u_1 \quad u_2 \quad \cdots \quad u_n]$, according to Proposition 4.

Now, coming back to the vector program (5), we know that it is equivalent to a positive semi-definite program, which can be solved in polynomial time. Suppose $\{x_1^*, \ldots, x_n^*\}$ is an optimal solution, and let $\texttt{OPT}(VP)$ be the value of the optimal objective. Since we have seen that the vector program is a relaxation, we have $\texttt{OPT}(VP) \geq \texttt{OPT}$, where $\texttt{OPT}$ is the optimal value for the max-cut instance (i.e., the maximum number of edges between L and R). It remains to convert $\{x_1^*, \ldots, x_n^*\}$ to a max-cut solution.

We find a random separating hyper-plane in $\mathbb{R}^n$ (that passes through the origin). Those x_i^* on one side of the plane correspond to $i \in L$ and those x_j^* on the other side correspond to $j \in R$.

Before we move on, notice that, to sample a random hyper-plane in $\mathbb{R}^n$, it suffices to sample a normal vector of this plane with a unit length. This can be done by sampling $r = (r_1, \ldots, r_n)$ such that each

$$r_i \sim \mathcal{N}(0,1)$$

is sampled from a normal distribution, and then normalize the vector with $\frac{r}{\|r\|}$. The probability density function is then

$$\Pr(r_1, \ldots, r_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{r_i^2}{2}\right) = (2\pi)^{-\frac{n}{2}} \exp\left(-\frac{\|r\|^2}{2}\right).$$

Therefore, fixing the length $\|r\|$, this is a uniform distribution, as the probability density function is a constant function.

Finally, for each x_i^* and x_j^*, a random hyper-plane separates them with probability equals to the angle between the two vectors, divided by π. This probability is $\frac{\arccos((x_i^*)^\top x_j^*)}{\pi}$. Therefore, the expected number of edges between A and B is

$$\sum_{\{i,j\}\in E} \Pr(i \text{ and } j \text{ are separated}) = \sum_{\{i,j\}\in E} \frac{\arccos((x_i^*)^\top x_j^*)}{\pi}.$$

On the other hand, we have

$$\texttt{OPT} \le \texttt{OPT}(VP) = \sum_{\{i,j\}\in E} \frac{1}{2}\left(1 - (x_i^*)^\top x_j^*\right).$$

Let

$$\alpha^* = \min_{-1\le x\le 1} \frac{2\arccos x}{\pi(1-x)} \approx 0.878.$$

We have an α^*-approximation algorithm.

Interesting enough, if we assume the *unique game conjecture* (search this on the Internet if you are interested in), this is the best we can do. Khot et al. [2007] showed that, assuming the unique game conjecture, it is NP-hard to approximate the max-cut problem to within a factor of $\alpha^* + \varepsilon$ for any constant $\varepsilon > 0$.

As a side note, the unique game conjecture sometimes gives stronger inapproximability results than those inapproximability results based on P $\neq$ NP. For example, we have seen in the previous lecture that there is a 2-approximation algorithm for the vertex cover problem. The unique game conjecture suggests that this is the best we can do (achieving $(2-\varepsilon)$-approximation is NP-hard for any $\varepsilon > 0$) Khot and Regev [2008], while the best inapproximability factor for P $\neq$ NP is $\sqrt{2}-\varepsilon$ (for any $\varepsilon > 0$) Khot et al. [2017].

参考文献

Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001. 3

Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008. 10

Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007. 10

Subhash Khot, Dor Minzer, and Muli Safra. On independent sets, 2-to-2 games, and grassmann graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 576–589, 2017. 10