APPROXIMATE MAX-CUTS VIA MATRICES

Suppose we wish to find an approximate maximum cut of a weighted graph G with vertices $V = \{1, 2, \ldots n\}$ and weight A_{ij} on edge (i, j). This problem is known to be NP-hard. There is a polynomial time 0.87-approximation algorthm. Here we will describe a polynomial time algorithm which yields an answer with **additive** error at most $\epsilon M n^2$, where $M = \max_{ij} |A_{ij}|$. The method described here is based on matrices and basic Linear Algebra, and is thus not a typical graph algorithm. The algorithm actually extends to all MAX-r-CSP problems and gives algorithms with similar error bounds. Also, the algorithm can be modified to work with just a random subset S of vertices and all edges incident to S, where $|S| \in O(1/\epsilon^4)$. This modification is quite non-trivial and we do not go into this here.

Decomposing the matrix A The first part of the algorithm will express the matrix as the sum of $O(1/\epsilon^2)$ simple matrices, called "cut" matrices. A cut matrix C is specified by two susbets V_1, V_2 of V and a real number d. Denoted $Cut(S, T; d)$, it is defined as :

$$C_{ij} = \begin{cases} d & \text{if } i \in V_1, \quad j \in V_2, \\ 0 & \text{otherwise.} \end{cases}$$

Notation For any $S, T \subseteq V$, we let $A(S, T)$ denote $\sum_{i \in, j \in T} A_{ij}$. Also, we denote by $||A||_C$ the quantity $\max_{S,T \subseteq V} |A(S, T)|$.

Lemma Given any $\epsilon > 0$, we can find in polynomial time $s \leq 4/\epsilon^2$ cut matrices $D_1, D_2, \ldots D_s$ such that

$$||A - (D_1 + D_2 + \ldots D_s)||_C \leq \epsilon M n^2.$$

Proof We will find the matrices $D_1, D_2, \ldots$ one by one. First, note that we have an algorithm (from the last lecture) which will find subsets S_1, T_1 of rows and columns respy such that

$$|A(S_1, T_1)| \geq \max_{S,T} |A(S, T)| - \frac{\epsilon}{4} M n^2.$$

We run this algorithm on A. If for the resulting S_1, T_1, we have

$$|A(S_1, T_1)| \leq \frac{\epsilon}{2} M n^2,$$

then I claim we are done. (Why ??) So, suppose not. Now, our first cut matrix D_1 will be $Cut(S_1, T_1; d_1)$, where d_1 will be the average of the entries $A_{ij}, i \in S_1, j \in T_1$.

We will then repeat the process with the matrix $A - D_1$. Why does this algorithm terminate in $O(1/\epsilon^2)$ iterations ? First note that in going from A to $A - D_1$, we are just subtracting from the entries $A_{ij}, i \in S_1, j \in T_1$ their average. In general, subtracting their average from a set of real numbers lowers their sum of sqaures. More precisely, if $a_1, a_2, \ldots a_n$ are n reals and $\bar{a}$ their average, then we have

$$\sum_i (a_i - \bar{a})^2 = \sum_i a_i^2 - n\bar{a}^2.$$

Using this, we get

$$\sum_{i,j\in V}(A-D_1)^2_{ij}-\sum_{ij}A^2_{ij}=\sum_{i\in S_1,j\in T_1}\left((A-D_1)^2_{ij}-A^2_{ij}\right)$$
$$=-|S_1||T_1|d_1^2\leq -|S_1|T_1|(\epsilon^2n^4M^2/4|S_1|^2|T_1|^2)\leq -\epsilon^2n^2M^2/4.$$

This proves termination in $4/\epsilon^2$ steps and thus the lemma. Why ?

For a cut matrix $Cut(S,T;d)$, we call $|d|$ its weight.

Claim We can modify the algorithm in the last lemma so that we have that the weight of each cut matrix $D_1, D_2, \ldots D_s$ is at most $2M$.

Proof We will acheive this by making sure that $|S_1|, |T_1| \geq n/2$. So, suppose $|S_1| < n/2$. Then note that $A(S_1,T_1)+A(V\setminus S_1,T_1)=A(V,T_1)$. Since $|A(S_1,T_1)|>\frac{\epsilon}{2}Mn^2$, we thus have that either $|A(V\setminus S_1,T_1)|\geq\frac{\epsilon}{4}Mn^2$ or $|A(V,T_1)|\geq\frac{\epsilon}{4}Mn^2$. So we replace S_1 by of one $V\setminus S_1$ or V and then we have

$$|A(S_1,T_1)|\geq\frac{\epsilon}{4}Mn^2,\qquad |S_1|\geq n/2.$$

Similarly, we may do the same to T_1 and then assume that

$$|A(S_1,T_1)|\geq\frac{\epsilon}{8}Mn^2,\qquad |S_1|,|T_1|\geq n/2.$$

Now, we know that from the sum of squares, we subtract $d_1^2|S_1||T_1|$ which has to be at most n^2M^2 (why?). Since $|S_1|,|T_2|\geq n/2$, we now have $|d_1|\leq 2M$ as required. We have to repeat this at every step; now the number of steps will still be $O(1/\epsilon^2)$, but with a different constant.

Now back to max cut. In what follows, x will range over $n-$ vectors with each component equal to 0 or 1 and $\underline{1}$ will denote the $n-$vector of all 1's. All our vectors will be considered column vectors and for a vector u, its transpose, which is a row vector will be denoted u^T. We run the algorithm in the previous lemma and find $D_1, D_2, \ldots D_s$. Let E (for error) denote $A-(D_1+D_2+\ldots D_s)$.

Claim For any cut $(S,\bar{S})$ in the graph, the weight of the cut is $x^TA(\underline{1}-x)$.

Proof Left to the reader.

Claim For any x, $|x^TE(\underline{1}-x)|\leq\epsilon n^2M$.

Proof Left to the reader.

Thus, now we can solve the max-cut problem with edge weights given by $D_1+D_2+\ldots D_s$ instead of A incurring an error of at most ϵn^2M. We will see that it is easier to solve the max-cut problem with the weight matrix being $D_1+D_2+\ldots D_s$.

First for the moment, suppose we had only one cut matrix $D_1=Cut(S_1,T_1;d_1)$. Then for any vector x,

$$x^TD_1(\underline{1}-x)=d_1x(S_1)(\underline{1}-x)(T_1),$$

where for any vector y and set S, $y(S) = (\sum_{i \in S} y_i)$. Since $|d_1| \leq 2M$, and we are seeking an $O(\epsilon M n^2)$ additive error, it suffices to know $x(S_1)$ and $(\underline{1} - x)(T_1)$, each within additive error $O(\epsilon n)$. There are thus only $O(1/\epsilon)$ "rounded" values of each of these quantities to try; we can try each one, see if it is feasible and take the best.

Now in general, there are $2s$ sets $S_1, S_2, \ldots S_s$ and $T_1, T_2, \ldots T_s$ and for any cut $(S, \bar{S})$, if x is the 0-1 vector with $x_i = 1$ for $i \in S$ and $x_i = 0$ for $i \notin S$, then the weight according to $D_1 + D_2 + \ldots D_s$ of the cut $(S, \bar{S})$ is

$$\sum_{t=1}^{s} d_t x(S_t)(|T_t| - x(T_t)).$$

So, it suffices to determine which $2s$ tuples of values of $\{x(S_t), x(T_t)\}$ can be realized by a 0-1 vector x. Now since each $|d_t| \leq 2M$, it suffices to know the $x(S_t), x(T_t)$ each to within $O(\epsilon^3 n)$ (Why ??); so there are only $O(1/\epsilon^3)^{O(1/\epsilon^2)}$ candidate $2s$ tuples which we need to try. We try each of these in turn and check (see below) for each of them whether it is realizable.

We can partition V into at most 2^{2s} disjoint sets $P_1, P_2, \ldots$ such that each of the S_t, T_t is the union of some of the P_i. If we have an integer variable y_i for each i, (which is the number of elements of P_i we include in our candidate set), we can clearly write an integer program in these variables which will determine whether a $2s$ tuple can be realized. An integer program in m variables can be solved in time $m^{O(m)}$, so this process will take time doubly exponential in $1/\epsilon$ (and polynomial in n). The dependence on $1/\epsilon$ can be improved to singly exponential, by noting that we can solve the linear programming relaxation instead of the integer program. Since there are only $O(s)$ constraints (besides the lower and upper bounds on the variables), there will only be $O(s)$ fractional variables in a basic feasible solution (if the LP is feasible) and rounding them causes very little error.