6.854J / 18.415J Advanced Algorithms
Fall 2008

18.415/6.854 Advanced Algorithms November 19, 2008

Approximaion Algorithms: MAXCUT

Lecturer: Michel X. Goemans

1 MAX-CUT problem

MAX-CUT Problem: Given a graph $G = (V, E)$ and weights on the edges $w : E \to R^+$, find a cut $(S : \bar{S}), S \subseteq V$ that maximizes $w(S : \bar{S}) = \sum_{e \in (S:\bar{S})} w(e)$.
MIN-CUT Problem: find a cut $(S : \bar{S})$ that minimizes $w(S : \bar{S})$.

There is a polynomial algorithm for the MIN-CUT problem: use the min $s - t$ cut algorithm on each pair of vertices (or, better, for a fixed s), and take the smallest of them. However, the MAX-CUT problem is NP-hard, and we'll try several ways of designing approximation algorithms for it.

2 Idea #1: Local Search

Algorithm: Start from any cut $(S : \bar{S})$. Define the neighborhood $N(S : \bar{S})$ of the cut to be the MOVE neighborhood: all the cuts that result from moving one vertex from one side of the cut to the other side. Consider a locally maximum cut for this neighborhood.

Lemma 1 *If $(S : \bar{S})$ is a local maximum for the MOVE neighborhood, then $w(S : \bar{S}) \geq \frac{1}{2}w(E) \geq \frac{1}{2}OPT$.*

Proof of lemma 1: Look at a vertex $i \in V$. Let C_i be the set of all edges $(i, j) \in E$ that are part of the cut $(S : \bar{S})$ (that is if $i \in S$ then $j \in \bar{S}$ and vice versa). Let A_i be the set of all edges $(i, j) \in E$ that are **not** part of the cut $(S : \bar{S})$. Since moving any single vertex i to the other side of the cut does not improve the weight of the cut, we know that:

$$w(C_i) \geq w(A_i).$$

Summing over all vertices i, we get:

$$\sum_{i \in V} w(C_i) \geq \sum_{i \in V} w(A_i),$$

or $2w(S : \bar{S}) \geq 2w(E \backslash (S : \bar{S}))$. Rearranging, we get:

$$4w(S : \bar{S}) \geq 2w(E)$$

or

$$w(S : \bar{S}) \geq \frac{1}{2}w(E) \geq \frac{1}{2}OPT.$$

□

Remarks:

(a) The bound of 1/2 cannot be improved for this MOVE neighborhood: Consider a k-vertex cycle, where k is a multiple of 4, as the graph G (with unit weights). The best cut will include

all edges. However, if we start from a cut in which the edges of the cycle alternate in and out of the cut, we have a locally optimum solution with only $k/2$ edges in the cut.

(b) The local search algorithm based on the MOVE neighborhood for MAX-CUT takes exponentially many steps in the worst-case. This is true even for graphs that are 4-regular (each vertex has exactly 4 neighbors) (Haken and Luby [1]). For 3-regular graphs the algorithm is polynomial (Poljak [4]).

(c) To capture the complexity of local search, Johnson, Papadimitriou and Yannakakis [3] have defined the class PLS (Polynomial Local Search). Members of this class are optimization problems of the form $\max\{f(x) : x \in S\}$ together with a neighborhood $N : S \to 2^S$. We say that $v \in S$ is a local optimum if $c(v) = \max\{c(x) : x \in N(v)\}$. To be in PLS, we need to have polynomial-time algorithms for (i) finding a feasible solution, (ii) deciding if a solution is feasible and if so computing its cost, and (iii) deciding if a better solution in the neighborhood $N(v)$ of a solution v exists and if so finding one. They introduce a notion of reduction, and this leads to PLS-complete problems for which any problem in PLS can be reduced to it. Their notion of reduction implies that if, for one PLS-complete problem, one has a polynomial-time algorithm for finding a local optimum then the same true for all PLS problems. In particular, MAX-CUT with the MOVE neighborhood is PLS-complete [5]. Furthermore, it follows from Johnson et al. [3] that the obvious local search algorithm is not an efficient way of finding a local optimum for a PLS-complete problem; indeed, for any PLS-complete problem, there exist instances for which the local search algorithm of repeatedly finding an improved solution takes exponential time. The result of Haken and Luby above is thus just a special case. Still, this does not preclude other ways of finding a local optimum.

3 Idea #2: Random Cut

Algorithm: There are $2^{|V|}$ possible cuts. Sample a cut randomly using a uniform distribution over all possible cuts in the graph: $\forall v \in V,\ Pr(v \in S) = \frac{1}{2}$, independently for all vertices $v \in V$.

Lemma 2 *This randomized algorithm gives a cut with expected weight that is* $\geq \frac{1}{2}OPT$.

Proof of lemma 2:

$$\begin{aligned} E[w(S:\bar{S})] &= E[\sum_{e\in E} w(e)I(e \in (S:\bar{S}))] = \sum_{e\in E} w(e)\cdot Pr(e \in (S:\bar{S})) \\ &= \sum_{e\in E} w(e)\cdot\frac{1}{2} = \frac{1}{2}w(E). \end{aligned}$$

□

Using the method of *conditional expectations*, we can transform this randomized algorithm into a deterministic algorithm. The basic idea is to use the following identity for a random variable f and event A:

$$\begin{aligned} E[f] &= E[f|A]Pr(A) + E[f|\bar{A}]Pr(\bar{A}) = E[f|A]Pr(A) + E[f|\bar{A}](1 - Pr(A)) \\ &\leq \max\{E[f|A], E[f|\bar{A}]\}. \end{aligned}$$

In our setting, we consider the vertices in a specific order, say $v_1, v_2, \cdots$, and suppose we have already decided/conditioned on the position (i.e. whether or not they are in S) of $v_1, \cdots, v_{i-1}$. Now, condition on whether $v_i \in S$. Letting $f = w(S:\bar{S})$, we get:

$$\begin{aligned} &E[f|\{v_1, \cdots, v_{i-1}\} \cap S = C_{i-1}] \\ &\leq \max(E[f|\{v_1, \cdots, v_{i-1}\} \cap S = C_{i-1}, v_i \in S], E[f|\{v_1, \cdots, v_{i-1}\} \cap S = C_{i-1}, v_i \notin S]). \end{aligned}$$

Both terms in the max can be easily computed and we can decide to put v_i on the side of the cut which gives the maximum, i.e. we set C_i to be either C_{i-1} or $C_{i-1} \cup \{v_i\}$ in such a way that:

$$E[f|\{v_1, \cdots, v_{i-1}\} \cap S = C_{i-1} \leq E[f|\{v_1, \cdots, v_i\} \cap S = C_i].$$

When we have processed all inequalities, we get a cut $(C_n : \bar{C_n})$ such that

$$\frac{1}{2}w(E) \leq E[f] \leq w(C_n : \bar{C_n}),$$

and this provides a deterministic 0.5-approximation algorithm.

Examining this derandomized version more closely, we notice that we will place v_i on the side of the cut that maximizes the total weight between v_i and the previous vertices $\{v_1, v_2, \cdots, v_{i-1}\}$. This is therefore a simple greedy algorithm.

Remarks:

(a) The performance guarantee of the randomized algorithm is no better than 0.5; just consider the complete graph on n vertices with unit weights. Also, the performance guarantee of the greedy algorithm is no better than 0.5 int he worst-case.

4 Idea #3: LP relaxation

Algorithm: Start from an integer-LP formulation of the problem:

$$\begin{aligned}
\max \quad & \sum_{e \in E} w(e)x_e \\
s.t. \quad & x_e \in \{0,1\} \quad \forall e \in E \\
& \sum_{e \in F} x_e + \sum_{e \in C \setminus F} (1 - x_e) \leq |C| - 1 \quad \forall \textit{cycle } C \subseteq E \; \forall F \subseteq C, \; |F| \textit{ odd} \\
\Leftrightarrow \quad & \sum_{e \in F} x_e - \sum_{e \in C \setminus F} x_e \leq |F| - 1 \quad \forall \textit{cycle } C \subseteq E \; \forall F \subseteq C, \; |F| \textit{ odd}
\end{aligned}$$

Since we have a variable x_e for each edge (if $x_e = 1$ than $e \in (S : \bar{S})$), we need the second type of constraints to guarantee that S is a legal cut. The validity of these constraints comes from the fact that any cycle and any cut must intersect in an even number of edges. even number of edges that are in the cut.

Next, we relax this integer program into a LP:

$$\begin{aligned}
\max \quad & \sum_{e \in E} w(e)x_e \\
s.t. \quad & 0 \leq x_e \leq 1 \quad \forall e \in E \\
& \sum_{e \in F} x_e - \sum_{e \in C \setminus F} x_e \leq |F| - 1 \quad \forall \textit{cycle } C \subseteq E \; \forall F \subseteq C, \; |F| \textit{ odd.}
\end{aligned}$$

This isa relaxation of the maximum cut problem, and thus provides an upper bound on the value of the optimum cut. We could try to solve this linear program and devise a scheme to "round" the possibly fractional solution to a cut.

Remarks:

(a) This LP can be solved in a polynomial time. One possibility is to use the ellipsoid algorithm as the separation problem over these inequalities can be solved in polynomial time (this is not trivial). Another possibility is to view the feasible region of the above linear program as the projection of a polyhedral set $Q \subseteq \mathbb{R}^{n^2}$ with $O(n^3)$ number of constraints; again, this is not obvious.

(b) If the graph G is planar, then all extreme points of this linear program are integral and correspond to cuts. We can therefore find the maximum cut in a planar graph in polynomial time (there is also a simpler algorithm working on the planar dual of the graph).

(c) There exist instances for which $\frac{OPT}{LP} \sim \frac{1}{2}$ (or $\exists G = (V, E),\ w(e) = 1, OPT \leq n(\frac{1}{2} + \epsilon),\ LP \geq n(1-\epsilon)$), which means that any rounding algorithm we could come up with will not guarantee a factor better than $\frac{1}{2}$.

5 Idea #4: SDP relaxation

The idea is to use semidefinite programming to get a more useful relaxation of the maximum cut problem. This is due to Goemans and Williamson [2].

Instead of defining variables on the edges as we did in the previous section, let's use variables on the vertices to denote which side of the cut a given vertex is. This leads to the following quadratic integer formulation of the maximum cut problem:

$$\begin{aligned} \max \quad & \sum_{(i,j)\in E} w(i,j)\frac{1-y_i y_j}{2} \\ s.t. \quad & y_i \in \{1,-1\}^n \ \forall i \in V. \end{aligned}$$

Here we have defined a variable y_i for each vertex $i \in V$ such that $y_i = 1$ if $i \in S$ and $y_i = -1$ otherwise. We know that an edge (i, j) is in the cut $(S : \bar{S})$ iff $y_i y_j = -1$, and this explains the quadratic term in the objective function.

We can rewrite the objective function in a slightly more convenient way using the *Laplacian* of the graph. The Laplacian matrix L is defined as follows:

$$l_{ij} = \begin{cases} 0 & (i,j) \notin E \\ -w(i,j) & i \neq j, (i,j) \in E \\ \sum_{k:k\neq i} w(i,k) & i = j. \end{cases}$$

that is, the off-diagonal elements are the minus the weights, and the diagonal elements correspond to the sum of the weights incident to the corresponding vertex. Using the Laplacian matrix, we can rewrite equivalently the objective function in the following way:

$$\begin{aligned} y^T L y &= \sum_{i=1}^n \sum_{j=1}^n y_i y_j l_{ij} = \sum_{i=1}^n y_i^2 \sum_{k\neq i} w(i,k) - \sum_{(i,j)\in E} y_i y_j w(i,j) \\ &= 2w(E) - \sum_{(i,j)\in E} y_i y_j w(i,j) = 4\left(\sum_{(i,j)\in E} w(i,j)\frac{1-y_i y_j}{2}\right), \end{aligned}$$

and thus

$$\sum_{(i,j)\in E} w(i,j)\frac{1-y_i y_j}{2} = \frac{1}{4} y^T L y.$$

Thus the maximum cut value is thus equal to

$$\max\{\frac{1}{4}y^T LY : y \in \{0,1\}^n\}.$$

If the optimization was over all $y \in \mathbb{R}^n$ with $||y||_2^2 = n$ then we would get that

$$\max\{\frac{1}{4}y^T LY : y \in \mathbb{R}^n, ||y||^2 = n\} = \frac{n}{4}\lambda_{max}(L),$$

where $\lambda_{max}(L)$ is the maximum eigenvalue of the matrix L. This shows that $OPT \leq \frac{n}{4}\lambda_{max}(L)$; this is an eigenvalue bound introduced by Delorme and Poljak.

Using semidefinite programming, we will get a slightly better bound. Using the *Frobenius inner product*, we can again reformulate the objective function as:

$$\frac{1}{4}y^T Ly = \frac{1}{4}L \bullet (yy^T),$$

or as

$$\frac{1}{4}L \bullet Y$$

if we define $Y = yy^T$. Observe that $Y \succeq 0$, Y has all 1's on its diagonal, and its rank is equal to 1. It is easy to see that the coverse is also true: if $Y \succeq 0$, $rank(Y) = 1$ and $Y_{ii} = 1$ for all i then $Y = yy^T$ where $y \in \{-1, 1\}^n$. Thus we can reformulate the problem as:

$$\begin{aligned} \max \quad & \frac{1}{4}L \bullet Y \\ s.t. \quad & rank(Y) = 1, \\ & \forall i \in V: \ Y_{ii} = 1, \\ & Y \succeq 0. \end{aligned}$$

This is almost a semidefinite program except that the rank condition is not allowed. By removing the condition that $rank(Y) = 1$, we relax the problem to a semidefinite program, and we get the following SDP:

$$\begin{aligned} SDP = \max \quad & \frac{1}{4}L \bullet Y \\ s.t. \quad & \forall i \in V: \ Y_{ii} = 1, \\ & Y \succeq 0. \end{aligned}$$

Obviously, by removing the condition that $rank(Y) = 1$ we only increase the space on which we maximize, and therefore the value (simply denoted by SDP) to this semidefinite program is an upper bound on the solution to the maximum cut problem.

We can use the algorithms we described earlier in the class to solve this semidefinite program to an arbitrary precision. Either the ellipsoid algorithm, or the interior-point algorithms for conic programming. Remember that semidefinite programs were better behaved if they satisfied a *regularity condition* (e.g., they would satisfy strong duality). Our semidefinite programming relaxation of MAXCUT is particularly simple and indeed satisfies both the primal and dual regularity conditions:

(a) **Primal regularity conditions** $\exists Y \succ 0$ *s.t.* $Y_{ii} = 1$ $\forall i$. This condition is obviously satisfied (consider Y = I).

(b) **Dual regularity condition:** First consider the dual problem -

$$\begin{aligned} \min \quad & \frac{1}{4}\sum_{i\in V} z_i \\ s.t. \quad & \begin{pmatrix} z_1 & 0 & ... & 0 \\ 0 & z_2 & ... & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & ... & z_n \end{pmatrix} - L \succeq 0, \end{aligned}$$

where $z_i \in \mathbb{R}$ for all $i \in V$. The regulation condition is that there exist z_i's such that $\begin{pmatrix} z_1 & 0 & ... & 0 \\ 0 & z_2 & ... & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & ... & z_n \end{pmatrix} - L \succ 0$. This is for example satisfied if, for all i, $z_i > \lambda_{max}(L)$.

Remark: If we add the condition that $z_1 = z_2 = ... = z_n$ to the dual then the smallest value z_i can take is equal to $\lambda_{max}(L)$, and we derive that:

$$OPT \leq SDP \leq \frac{n}{4}\lambda_{max}(L),$$

and therefore this SDP bound improves upon the eigenvalue bound.

We will start the next lecture by proving the following theorem.

Theorem 3 ([2]) *For all $w \geq 0$, we have that $\frac{OPT}{SDP} \geq 0.87856$.*

In order to prove this theorem, we will propose an algorithm which derives a cut from the solution to the semidefinite program. To describe this algorithm, we first need some preliminaries. From the Cholesky's decomposition, we know that:

$$\begin{aligned} Y \succeq 0 \quad & \Leftrightarrow \quad \exists V \in R^{k\times n},\ k = rank(Y) \leq n,\ s.t.\ Y = V^T V \\ & \Leftrightarrow \quad \exists v_1, ..., v_n\ s.t.\ Y_{ij} = v_i^T v_j,\ v_i \in \mathbb{R}^n. \end{aligned}$$

Therefore, we can rewrite the SDP as a 'vector program':

$$\begin{aligned} max \quad & \sum_{(i,j)\in E} w(i,j)\frac{1 - v_i^T v_j}{2} \\ s.t. \quad & \forall i \in V:\ \|v_i\| = 1 \\ & \forall i \in V:\ v_i \in \mathbb{R}^n. \end{aligned}$$

To be continued...

References

[1] A. Haken and M. Luby, "Steepest descent can take exponential time for symmetric connection networks", *Complex Systems*, 1988.

[2] M.X. Goemans and D.P. Williamson, Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming, J. ACM, 42, 1115–1145, 1995.

[3] D.S. Johnson, C.H. Papadimitriou and M. Yannakakis, "How easy is local search", *Journal of Computer and System Sciences*, **37**, 79–100, 1988.

[4] S. Poljak, "Integer Linear Programs and Local Search for Max-Cut", *SIAM J. on Computing*, **24**, 1995, pp. 822-839.

[5] A.A. Schäffer and M. Yannakakis, "Simple local search problems that are hard to solve", *SIAM Journal on Computing*, **20**, 56–87, 1991.