COMPSCI 532: Design and Analysis of Algorithms	October 30, 2015
Lecture 17	
Lecturer: Debmalya Panigrahi	*Scribe: Allen Xiao*

1 Overview

In this lecture, we introduce semidefinite programming through the semidefinite programming algorithm for MAXCUT.
Note: missing diagrams.

2 Maximum Cut Problem

Definition 1. *For a graph $G = (V, E)$ with edge weights c_e, the **maximum cut problem** is to find a cut $S \subseteq V$ such that the weight of edges across $(S, \overline{S})$ is maximized.*

Before we get into semidefinite programming, we will quickly examine several simple approximations for maximum cut.

2.1 Greedy algorithm

We can think of the cut as a partition of verticesl $C = (S, V \setminus S)$ whiere $S \subseteq V$. We can switch between different cuts by moving vertices across the cut, in to or out of S. Moving v across the cut swaps its cut edges with its non-cut edges. This increase the value of the cut when the total weight of its non-cut edges exceeds the weight of the cut edges. If $v \in S$ (and similarly if $v \in V \setminus S$):

$$\sum_{e \in \delta(v) \cap (S \times S)} w(e) > \sum_{e \in \delta(v) \cap (S \times (V \setminus S))} w(e)$$

These greedy moves give us a simple optimization procedure:

1. Begin with an arbitrary cut (e.g. $S = \emptyset$).
2. While there are greedy moves improving the cost, perform them.

This procedure terminates since the cut value increases monotonically. But is not guaranteed to terminate optimally. There exist examples with "local optima", with suboptimal value but where no single-vertex swap improves the value.

To analyze the approximation ratio, observe that at optimality:

$$\sum_{e \in \delta(v) \cap (S \times S)} w(e) \leq \sum_{e \in \delta(v) \cap (S \times (V \setminus S))} w(e) \qquad \forall v \in S$$

and similarly for $v \in V \setminus S$. We can find an equivalent form by adding the weight of cut edges to both sides:

$$\sum_{e \in \delta(v) \cap (S \times S)} w(e) + \sum_{e \in \delta(v) \cap (S \times (V \setminus S))} w(e) \leq \sum_{e \in \delta(v) \cap (S \times (V \setminus S))} w(e) + \sum_{e \in \delta(v) \cap (S \times (V \setminus S))} w(e)$$

Simplifying:

$$\sum_{e \in \delta(v)} w(e) \leq 2 \sum_{e \in \delta(v) \cap C} w(e)$$

If we sum over all vertices:

$$\sum_{v} \sum_{e \in \delta(v) \cap C} w(e) \geq \frac{1}{2} \sum_{v} \sum_{e \in \delta(v)} w(e)$$

The left hand side is exactly twice the value of the cut, while the right hand side (sum of degree cuts) counts every edge twice.

$$2w(C) \geq \frac{1}{2} \left(2 \cdot \sum_{e} w(e) \right)$$

Since OPT uses a subset of edges:

$$2w(C) \geq \sum_{e} w(e) \geq \text{OPT}$$

This greedy algorithm therefore has an approximation factor of 2. Many algorithms for maximum cut also achieve a 2-approximation, including the next two we discuss.

2.2 Naive randomized algorithm

Instead of making informed choices about the graph, how good do we do when we simply assign vertices to S or $V \setminus S$ independently and uniformly at random? That is, for each $v \in V$:

$$\Pr(v \in S) = \Pr(v \in V \setminus S) = \frac{1}{2}$$

Then, for any edge (v, w), the probability it is a cut edge is the probability that the vertices were assigned to opposite partitions.

$$\Pr((v, w) \in C) = \frac{1}{2}$$

The expected weight of the cut is:

$$\begin{aligned} \sum_{e \in C} w(e) &= \sum_{e \in E} w(e) \Pr(e \in C) \\ &= \frac{1}{2} \sum_{e \in E} w(e) \\ &\geq \frac{1}{2} \text{OPT} \end{aligned}$$

Still a 2-approximation, on expectation.

2.3 Alternate greedy algorithm

The final 2-approximation we examine is an attempt to improve on the greedy algorithm. In this version, we iteratively insert vertices into the partition which maximizes the weight of neighbors so far.

Let the order in which vertices are added be $v_1, \ldots v_n$. Let $\delta(v_i, -) = \{(v, w) \mid w \in \{v_1, v_2, \ldots, v_{i-1}\}\}$, the edges between v_i and vertices added before v_i. A property similar to the one we derived in the first greedy algorithm holds here, but over $\delta(v, -)$:

$$\sum_{e \in \delta(v,-) \cap C} w(e) \geq \frac{1}{2} \sum_{e \in \delta(v,-)} w(e)$$

Summing over v again gives us the 2-approximation:

$$\sum_{v} \sum_{e \in \delta(v) \cap C} w(e) \geq \frac{1}{2} \sum_{v} \sum_{e \in \delta(v)} w(e)$$

The difference between these sums (with $\delta(v, -)$) and the previous ones with $\delta(v)$ is that edges are not double counted.

$$w(C) \geq \frac{1}{2} \sum_{e \in E} w(e) \geq \frac{1}{2} \text{OPT}$$

2.4 Maximum cut linear program

The following linear program for maximum cut still does not do better than a 2-approximation. Obviously, we wish to minimize over the weight of cut edges. How do we ensure the edge set we pick forms a valid cut?

First, observe that the edges of a valid cut form a bipartite graph. One of the core properties of bipartite graphs is that they have no odd cycles. The constraint we use is as follows: for every cycle Γ, the number of cut edges in the cycle should be even. We will express this the following way: For all $F \subset \Gamma$ (proper subset) with odd size:

1. F is missing a cut edge of Γ, or
2. At least one edge of F is not a cut edge.

We can express this the following way:

$$\begin{aligned}
\max \quad & \sum_{e \in E} w_e x_e \\
\text{s.t.} \quad & \forall \Gamma \in G, F \subset \Gamma, |F| \equiv 1 \pmod 2 \\
& \sum_{e \in F} x_e - \sum_{e \in (\Gamma \setminus F)} x_e \leq |F| - 1 \\
& x_e \geq 0 \qquad \forall e \in E
\end{aligned}$$

2.5 Relaxation into a quadratic program

The algorithm we will introduce for beating 2 will use something called semidefinite programming. This is a very slight relaxation of linear programs, and a class of non-linear optimization problems for which we have efficient algorithms (interior point methods, for example).

We return to the maximum cut problem. Suppose we relax the linearity constraint, and represent the cut as a partitioning of vertices instead:

$$y_v = \begin{cases} 1 & v \in S \\ -1 & v \in V \setminus S \end{cases}$$

If we let ourselves take products between y_v, the program for maximum cut is quite simple:

$$\begin{aligned} \max \quad & \sum_{(u,v)\in E} w_{uv} \left(\frac{1 - y_u y_v}{2} \right) \\ \text{s.t.} \quad & y_v \in \{\pm 1\} \qquad\qquad \forall v \in V \end{aligned}$$

Here, the function $(1 - y_u y_v)/2$ is a sort of parity function (XOR) between y_u and y_v.

$$\frac{1 - y_u y_v}{2} = \begin{cases} 1 & y_u = -y_v \\ 0 & y_u = y_v \end{cases}$$

Thus, the objective includes all (u, v) which are in the cut, as desired. What we have here, however, is a *quadratic program*. In general, quadratic programs are NP-hard to solve, but the form of the problem here falls in the subclass of *semidefinite programs*, which we can solve efficiently.

3 Semidefinite Programming

We will construct a generic semidefinite programs by starting from a linear program. Recall the matrix form of a linear program:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ik} x_i \geq b_k \quad \forall k \\ & x_i \geq 0 \qquad\qquad \forall i \end{aligned}$$

Suppose that x is a $\mathbb{R}^{n^2}$ vector representing a $\mathbb{R}^{n \times n}$ square matrix X. We can rewrite the program to index according to this matrix (by i, j instead of just i).

$$\begin{aligned} \min \quad & \sum_{i,j} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{i,j} a_{ijk} x_{ij} \geq b_k \qquad \forall k \\ & x_{ij} \geq 0 \qquad\qquad \forall i, j \end{aligned}$$

The nonnegativity constraint ($x_{ij} \geq 0$) says that X may only have nonnegative entries. We will relax this constraint, instead saying that X must be *positive semidefinite*.

Definition 2. *A square, symmetric matrix X is **positive semi-definite** (PSD) if all the eigenvalues of X are nonnegative. Equivalently, X can be decomposed such that each entry is the dot product of two vectors in $(\vec{v}_1, \ldots, \vec{v}_n) \in \mathbb{R}^{n \times n}$.*

$$x_{ij} = \vec{v}_i \cdot \vec{v}_j \qquad \forall i, j$$

If X is positive semidefinite, then by the decompsition definition there is some set of $\mathbb{R}^n$ vectors $(\vec{v}_1, \ldots, \vec{v}_n)$ where we can write the program as:

$$\begin{aligned} \min \quad & \sum_{i,j} c_{ij}(\vec{v}_i \cdot \vec{v}_j) \\ \text{s.t.} \quad & \sum_{i,j} a_{ijk}(\vec{v}_i \cdot \vec{v}_j) \geq b_k \quad \forall k \end{aligned}$$

Definition 3. *A **semidefinite program** (SDP) is an optimization problem of the form:*

$$\begin{aligned} \min_{\vec{v}_1, \ldots, \vec{v}_n \in \mathbb{R}^n} \quad & \sum_{i,j} c_{ij}(\vec{v}_i \cdot \vec{v}_j) \\ \textit{s.t.} \quad & \sum_{i,j} a_{ijk}(\vec{v}_i \cdot \vec{v}_j) \geq b_k \quad \forall k \end{aligned}$$

3.1 Maximum cut with SDP

The following maximum cut algorithm and analysis is due to Goemans and Williamson [GW95]. Recall the quadratic maximum cut program:

$$\begin{aligned} \max \quad & \sum_{(u,v) \in E} w_{uv} \left(\frac{1 - y_u y_v}{2} \right) \\ \text{s.t.} \quad & y_v \in \{\pm 1\} \qquad \forall v \in V \end{aligned}$$

To mold this into an SDP, we will need to make two changes:

1. Convert the y_i into n-dimensional vectors $\vec{y}_i$. We do this in the most naive way possible:

$$\vec{y}_i = \left\{ \begin{pmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} +1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right\}$$

2. Make a fractional relaxation, instead of $y_i \in \{\pm 1\}$, have $0 \leq y_i \leq 1$. In $\mathbb{R}^n$, we instead require $\vec{y}_i$ to be a unit vector.

$$\vec{y}_i \cdot \vec{y}_i = 1$$

After these modifications, the $\vec{y}_i$ are points on the n-dimensional unit sphere. The final SDP is:

$$\begin{aligned} \max \quad & \sum_{(i,j) \in E} w_{ij} \left(\frac{1 - \vec{y}_i \cdot \vec{y}_j}{2} \right) \\ \text{s.t.} \quad & \vec{y}_i \cdot \vec{y}_i = 1 \qquad \forall i \in V \\ & \vec{y}_i \in \mathbb{R}^n \qquad \forall i \in V \end{aligned}$$

As with linear programs, we must round fractional solutions of SDP back into integral ones. Here, we must partition V and send each vertex to S or $V \setminus S$. Each $\vec{y}_i$ is a point on the unit sphere in $\mathbb{R}^n$, we wil round by picking a random half-space $\vec{r}$ which partitions the sphere into two hemispheres. Let $\vec{r}$ be a uniform random point on the sphere:

$$\begin{cases} i \in S & \vec{y}_i \cdot \vec{r} \geq 0 \\ i \in V \setminus S & \vec{y}_i \cdot \vec{r} < 0 \end{cases}$$

An edge (i, j) ends up in the cut if its endpoints are on opposite sides of the hyperplane induced by $\vec{r}$.

Finding a uniform random direction $\vec{r} \in \mathbb{R}^n$ is a problem known as *hypersphere point picking*, which we will not discuss in this note. It is insufficient to choose each of n coordinates independently, or even use independent samples of the polar coordinates (neither will be uniform on the sphere).

3.2 Maximum cut SDP analysis

Suppose we have an edge (i, j). To analyze the approximation ratio, we will compare its contribution to the SDP versus the expected rounded contribution. Let $\theta \in [0, \pi]$ be the minimum angle between $\vec{y}_i$ and $\vec{y}_j$ in the SDP fractional solution.

$$\begin{aligned} & w_{ij} \left(\frac{1 - \vec{y}_i \cdot \vec{y}_j}{2} \right) \\ = \; & w_{ij} \left(\frac{1 - \|\vec{y}_i\| \|\vec{y}_j\| \cos\theta}{2} \right) \\ = \; & w_{ij} \left(\frac{1 - \cos\theta}{2} \right) \end{aligned}$$

After rounding, in expedctation, the probability that $\vec{y}_i$ and $\vec{y}_j$ are split by the $\vec{r}$ is proportional to the angle between them.

$$w_{ij} \Pr\left((i, j) \in C\right) = w_{ij} \left(\frac{\theta}{\pi} \right)$$

On this single edge, the gap between the SDP solution and the rounded solution is the ratio of the right hand sides. The worst case, for $0 \leq \theta \leq \pi$, can be solved numerically to:

$$\max_{0 \leq \theta \leq \pi} \frac{\theta/\pi}{(1 - \cos\theta)/2} \approx 0.878$$

Summing over all edges, the expected approximation ratio is:

$$\text{ALGO} \geq 0.878\text{SDP} \geq 0.878\text{OPT}$$

This algorithm can also be derandomized, for the same approximation ratio.

References

[GW95] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.