Lecture 1

February 14

1.1 Max Cut

Given a graph $G = (V, E)$, our goal is to maximize $\sum_{(i,j)\in E} w_{i,j} \cdot \mathbb{1}[(i,j) \text{ is a cut}]$. This has applications in, for example, VLSI. This is NP-Complete, so we will use an approximation.

Greedy: Here is a randomized algorithm. Place each vertex on one side of the cut with probability 1/2 independently. Then,

$$\begin{aligned}\text{ALG} &= \sum w_{i,j}\mathbb{1}[\text{ALG cuts } (i,j)],\\ \mathbb{E}[\text{ALG}] &= \sum_{(i,j)\in E} w_{i,j}\mathbb{P}[\text{ALG cuts } (i,j)]\\ &= \sum_{(i,j)\in E} w_{i,j}\cdot\frac{1}{2} = \frac{1}{2}|E| \geq \frac{1}{2}\cdot\text{OPT}.\end{aligned}$$

When we see vertex v_i, put it in S or T, maximizing the number of edges cut.

Claim: This also gives a 1/2-approximation.

Proof: At each step, we cut at least 1/2 of the edges. $\square$

1.1.1 LP for Max Cut

We will have a variable x_i for each $i \in V$, $x_i \in \{0,1\}$. $z_{i,j}$ is the indicator for cutting $(i,j) \in E$. We have the constraints

$$\begin{aligned} z_{i,j} &\leq x_i + x_j,\\ z_{i,j} &\leq 2 - x_i - x_j,\end{aligned}$$

with the objective

$$\max \sum_{(i,j)\in E} w_{i,j}\cdot z_{i,j}.$$

To perform the LP relaxation, we let $x_i \in [0,1]$.

An **Integrality Gap** (α) means $\exists G\ \text{OPT}(G) \leq \alpha\cdot\text{LP}(G)$. The gap instance here is $x_i = 1/2\ \forall i \in V$. Max-Cut has a 1/2-integrality gap. Consider K_n. The LP cuts

$$\binom{n}{2} = \frac{n(n-1)}{2} = \frac{n^2}{2}(1 - O(1))$$

edges, whereas

$$\mathrm{OPT} = \left(\frac{n}{2}\right)^2 = \frac{n^2}{4}.$$

Hence,

$$\mathrm{OPT} \le \frac{1}{2}\mathrm{LP}.$$

This is no better than the greedy algorithm!

Theorem 1.1 (Chan, Lee, Raghavendra, Steurer)**.** *There is no polynomial-sized LP for max cut with integrality gap better than* $1/2$*.*

In fact, we need $\ge n^{\log n}$ constraints. The technique is to show that we might as well use a specific LP (Sherali Adams). This result has been improved to require $\ge 2^{\varepsilon n}$ constraints.

1.2 Semidefinite Programming (SDP)

A common mantra is:

$$\begin{aligned} \text{SDP} &\leftrightarrow \text{quadratic programming},\\ \text{LP} &\leftrightarrow \text{integer linear programming}. \end{aligned}$$

This is generally not a very good mantra, but suitable for today.

Here is the QP for max cut:

$$\begin{aligned} x_i \in \{\pm 1\} &\leftrightarrow x_i^2 = 1,\\ z_{i,j} &= \frac{1}{2}(1 - x_i \cdot x_j),\\ \max \sum_{(i,j)\in E} & w_{i,j} \cdot \frac{1}{2}(1 - x_i x_j). \end{aligned}$$

From $z_i \in \mathbb{Z}$, we will now have $v_i \in \mathbb{R}^n$. The SDP is

$$\begin{aligned} v_1, \dots, v_n &\in \mathbb{R}^n,\\ \langle v_i, v_i\rangle = 1 &\leftrightarrow \|v_i\|_2^2 = 1,\\ \max \sum & w_{i,j} \cdot \frac{1}{2}(1 - \langle v_i, v_j\rangle), \end{aligned}$$

where $n = |V|$.

1. Why can we solve this?
2. What is the advantage? What's going on?

This is indeed a relaxation. If $x_1 = 1$, then a feasible solution is to take the column vector with a 1 in the first entry and 0s otherwise; similarly for $x_2 = -1$.

Consider v_i, v_j, unit vectors.

$$\begin{aligned} \|v_i - v_j\|_2^2 &= \|v_i\|_2^2 + \|v_j\|_2^2 - 2\langle v_i, v_j\rangle\\ &= 2(1 - \langle v_i, v_j\rangle). \end{aligned}$$

Since we have $\|v_i\|_2 = 1$, our objective is actually

$$\max \frac{1}{2} \sum w_{i,j} \|v_i - v_j\|_2^2.$$

The interpretation is that the SDP pushes vectors apart on the sphere when the corresponding vertices are connected by an edge.

The SDP can achieve an ≈ 0.878-approximation, which is $> 1/2$.

The algorithm chooses a random hyperplane, setting $x_i = 1$ for all v_i above the hyperplane and $x_i = -1$ for all v_i below the hyperplane.

$$\begin{aligned} \text{ALG} &= \sum w_{i,j} \mathbb{1}[\text{ALG cuts } i, j] \\ \mathbb{E}[\text{ALG}] &= \sum w_{i,j} \cdot \mathbb{P}[\text{ALG cuts } i, j] \end{aligned}$$

The probability that an edge is cut is proportional to the angle between the vectors $\theta(v_i, v_j)$. Hence, $\mathbb{P}[i, j \text{ cut}] = \theta(v_i, v_j)/\pi$.

$$= \sum w_{i,j} \frac{\theta(v_i, v_j)}{\pi}$$

One has $\cos(\theta(i, j)) = \langle v_i, v_j \rangle$.

$$= \sum w_{i,j} \frac{\arccos(\langle v_i, v_j \rangle)}{\pi}$$

One can prove analytically that for all $-1 \le x \le 1$, $\arccos(x)/\pi \ge 0.878((1/2)(1 - \langle v_i, v_j \rangle))$.

$$\begin{aligned} &\ge \sum w_{i,j} \cdot 0.878 \left(\frac{1}{2} (1 - \langle v_i, v_j \rangle) \right) \\ &= 0.878 \cdot \text{SDP} \ge 0.878 \cdot \text{OPT}. \end{aligned}$$

If the Unique Games Conjecture (UCG) is true, then it is NP-Hard to improve over 0.878.

Raghavendra: "If the naïve SDP has integrality gap α, then assuming UGC, it is NP-Hard to improve over α."

1.3 Solving SDPs

1.3.1 Positive Semidefinite Matrices

Why can we (efficiently) solve for vectors?

Definition 1.2. A **positive semidefinite matrix (PSD)** A is a $n \times n$ symmetric matrix such that all eigenvalues of A are non-negative.

There are equivalent definitions.

A is PSD $\iff A = VV^\top \iff x^\top A x \ge 0 \; \forall x \in \mathbb{R}^n$.

We have

$$A = \sum \lambda_i \cdot u_i u_i^\top \qquad \text{(spectral theorem)}$$

$$
\begin{aligned}
&= U \begin{bmatrix} \ddots & & 0 \\ & \lambda_i & \\ 0 & & \ddots \end{bmatrix} U^\top \\
&= U \begin{bmatrix} \ddots & & 0 \\ & \sqrt{\lambda_i} & \\ 0 & & \ddots \end{bmatrix} \begin{bmatrix} \ddots & & 0 \\ & \sqrt{\lambda_i} & \\ 0 & & \ddots \end{bmatrix} U^\top \\
&= VV^\top
\end{aligned}
$$

because $\lambda_i \geq 0$.

If $A = VV^\top$, then

$$
\begin{aligned}
x^\top Ax &= x^\top VV^\top x \\
&= \left\|V^\top x\right\|_2^2 \\
&\geq 0.
\end{aligned}
$$

If $x^\top Ax \geq 0 \; \forall x \in \mathbb{R}^n$, for any eigenvector $u_i \in \mathbb{R}^n$,

$$u_i^\top Au_i = \lambda_i \geq 0.$$

Let $A_{i,j} = \langle v_i, v_j \rangle$. We can rewrite the SDP as

$$
\begin{aligned}
\max \sum_{(i,j)\in E} w_{i,j} \cdot \frac{1}{2}(1 - A_{i,j}) \\
A_{i,i} = 1, \\
A_{i,j} = A_{j,i} \; \forall i, j, \\
A \text{ is PSD } (A \succeq 0).
\end{aligned}
$$

1.3.2 Ellipsoid Algorithm

Given any convex set $K \subseteq \mathbb{R}^n$, the algorithm finds some $x \in K$. If the objective value is OBJ $= m$, then we can use binary search to find the optimal objective value.

We need as input an "oracle" for separating hyperplanes. (This is why convexity is necessary.) We repeatedly guess points and ask the oracle for separating hyperplanes. The runtime is logarithmic in the "niceness of K". Convex sets can only be exponentially bad, so the overall runtime is a polynomial.

We already know that the linear constraints give us a convex set. We will prove that the set of PSD matrices is convex.

K is convex $\iff \forall x_1, x_2 \in K \; c \cdot x_1 + (1-c) \cdot x \in K \; \forall 0 < c < 1$. Let $X_1, X_2 \in \{\text{PSD matrices}\}$. Then, for all $v \in \mathbb{R}^n$, we must prove

$$v^\top(cX_1 + (1-c)X_2)v \geq 0$$

(recall that A is PSD $\iff v^\top Av \geq 0 \; \forall v$). However, we have

$$v^\top(cX_1 + (1-c)X_2)v = \underbrace{c}_{\geq 0} \cdot \underbrace{v^\top X_1 v}_{\geq 0} + \underbrace{(1-c)}_{\geq 0} \cdot \underbrace{v^\top X_2 v}_{\geq 0} \geq 0,$$

so $cX_1 + (1-c)X_2$ is also PSD.