College of Computer & Information Science
Northeastern University
CS G399: Algorithmic Power Tools I

Fall 2007
Handout 9 November 2007
Scribe: Jingjing Duan

Lecture Outline:

- SDP
- MAX-CUT

Last class, we discussed fractional chromatic and clique number. We showed that

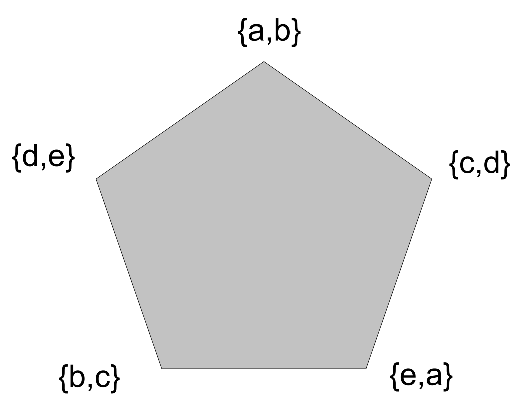
$$\omega(G) \leq \sup \sqrt[n]{\omega(G^n)} = \mathrm{L}\Theta(G) \leq \chi_f(G) \leq \chi,$$

where $\mathrm{L}\Theta(G)$, the Lovasz theta function of G, is the largest value of $\sqrt{\frac{1}{\cos\theta}}$ achieved by a nice orthonormal representation of G (recall that all the vectors in a nice orthonormal representation subtend an angle θ with a particular unit vector e).

We now consider an alternative defintion of the fractional chromatic number.

Definition 1. An (a, b)-fold coloring is an assignment $S : V \to 2^U$, $|U| = b$, such that $|S(v)| = a$, for all v, and $S(v_1) \cap S(v_2) \neq \emptyset$ if $(v_1, v_2) \in E$. The a-fold chromatic number equals $\min \frac{b}{a}$, where the minimum is over all (a, b)-fold colorings.

An example makes this clear.



In other words, assign objects to each node such that two connected nodes don't share any objects.

Clearly, an optimal $(1, b)$-fold coloring gives the normal chromatic number.

Definition 2. A graph G is (a, b)-choosable if for any assignment of a list of a objects to each of its vertices there is a subset of b objects of each list so that subsets corresponding to adjacent vertices are disjoint. The a-choosable chromatic number of G is $\min \frac{b}{a}$, over all (a, b) such that G is (a, b)-choosable.

It can be shown that the fractional chromatic number χ_f, the best a-fold chromatic number, and the best a-choosable chromatic number are all equivalent. Formally,

$$\inf_a a\text{-fold coloring} = \inf_a a\text{-choosable chromatic number} = \chi_f.$$

1 SDP

Semidefinite programming (SDP) is a subfield of convex optimization concerned with the optimization of a linear objective function over the intersection of the cone of positive semidefinite matrices with an affine space.

A simple comparison between LP and SDP:

LP	SDP
$\geq$	$\succeq$
vector $\cdot$	matrix $\cdot$

In SDP, we have Frobenius inner product of two matrices, which is the component-wise inner product of two matrices as though they are vectors. In other words, it is the sum of the entries of the Hadamard product, that is,

$$A \cdot B = \sum_i \sum_j A_{ij} B_{ij} = \text{trace}(A^T B) = \text{trace}(AB^T)$$

Further, the trace of a square matrix A is defined to be the sum of the elements on the main diagonal of A, i.e.,

$$\text{tr}(A) = a_{11} + a_{22} + \cdots + a_{nn} = \sum_i a_{ii}$$

Primal-dual of SDP:

$$\begin{array}{c|c} \min\ C^T x & \max\ yB \\ \sum x_i A_i \succeq B & yA_i = C_i \\ & y \succeq 0 \end{array}$$

Here A and B are positive semidefinite matrices. They have to be symmetric. They have the following properties:

- $\forall x, x^T A x \succeq 0$
- eigen values are non-negative
- $A = U^T U$, for some U
- A is a non-negative linear combination of xx^T, for some vector x
- symmetric minor determinant is non-negative

A little example:

$$\begin{array}{c} \min x_1 \\ \begin{pmatrix} x_1 & 1 \\ 1 & x_2 \end{pmatrix} \succeq 0 \end{array} \quad \Longleftrightarrow \quad \begin{array}{c} \min x_1 \\ x_1 x_2 \succeq 1 \\ x_1, x_2 \succeq 0 \end{array}$$

SDP is equivalent to Vector Programming. A vector program is a program in which we optimize functions of linear inner products of vectors subject to linear constraints. More specifically, a Vector Program can be formulated as follows.

$$\min \sum_{1 \leq i,j \leq n} c_{ij} v_i^T v_j$$

$$\sum_{1 \leq i,j \leq n} a_{ij}^{(k)} v_i^T v_j = b_k, 1 \leq k \leq m$$

$$v_i \in \mathcal{R}^n, 1 \leq i \leq n$$

2 MAX-CUT

The Max-Cut problem is defined as: given a graph $G = (V, E)$, a weight function $w : E \to Z^+$. The goal is to determine a cut $(S, \overline{S})$, where $S \neq \emptyset$ or V, such that $w(S, \overline{S})$ is maximized.

A straightforward greedy algorithm will give a $\frac{1}{2}-$ratio approximation algorithm:

- Suppose V will be divided into 2 subsets V_1 and V_2. Initially, V_1 is empty.
- Pick the node of the highest degree and put it into V_1.
- See whether it increases the weight of edges crossing.
- Keep doing this.

There are 2 other ways to approximate the max-Cut problem:

- Local improvement: Start from an arbitrary partition and repeatedly move a vertex from one side to the other if the max-cut improves.
- Random Optimization: Randomly partition the nodes into two sets.

They also give a $\frac{1}{2}$-ratio approximation.

An attempt at writing an LP for Max-Cut problem would be:

$$\max_{e=ij} \sum y_{ij}$$

such that

$$y_{ij} \leq |x_i - x_j|$$

$$0 \leq x_i \leq 1$$

Actually, it doesn't quite work out, since it is not easy to deal with $|x_i - x_j|$.