

## Lecture 12: Using LP duality to approximate vertex cover; Semi-Definite Programming for Max Cut and 3-coloring

### 1 LP duality and the primal-dual method

Every linear programming problem of the form:

$$\begin{aligned} & \text{minimize } \sum_{j=1}^n c_j x_j \\ & \text{subject to } \sum_{j=1}^n a_{i,j} x_j \geq b_i, i = 1, \dots, m, j = 1, \dots, n \\ & \forall x_j, x_j \geq 0 \end{aligned}$$

has a corresponding dual problem of the following form:

$$\begin{aligned} & \text{maximize } \sum_{i=1}^m b_i y_i \\ & \text{subject to } \sum_{i=1}^m a_{i,j} y_i \geq c_j, i = 1, \dots, m, j = 1, \dots, n \\ & \forall y_i, y_i \geq 0. \end{aligned}$$

**Theorem 1.0.1 (LP duality)** *If  $\mathbf{x}^* = (x_1^*, x_2^*, \dots, x_n^*)$  is an optimal solution of a linear program and  $\mathbf{y}^* = (y_1^*, y_2^*, \dots, y_m^*)$  is an optimal solution to its dual, then  $\sum_{j=1}^n c_j x_j^* = \sum_{i=1}^m b_i y_i^*$ .*

Several problems discussed in the course, such as maximum network flow, and many other common problems, such as minimax games, are special cases of LP duality.

## 2 Using LP duality to approximate vertex cover

### 2.1 Vertex cover as an integer program

The vertex cover problem is a special case of the set cover problem. The sets are edges of a graph, and the objective is to find the smallest set of vertices that cover all edges, in other words, touch at least one of their endpoints.

More formally: given a graph  $G = (V, E)$ , and a set of rational vertex weights  $V \rightarrow \mathbf{Q}^*$ , choose a minimum-weight set of vertices such that for all edges in  $E$ , at least one endpoint is at a chosen vertex.

Vertex cover can be formulated as an integer program. We set each  $x_i$  in the solution  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  equal to 1 if the corresponding vertex is in the covering set, and to 0 if it is not. Thus for each edge  $st$ , we have the constraint  $x_s + x_t \geq 1$  and we wish to minimize  $\sum_{s \in V} w(s)x_s$ .

Since this problem is a special case of set cover, we know that there exists a randomized  $O(\log n)$ -approximation algorithm (see Lecture 8). We now show a 2-approximation.

### 2.2 2-approximation

Let  $\mathbf{x}$  and  $\mathbf{y}$  be feasible solutions to the primal and dual respectively. Then  $x$  and  $y$  are optimal if and only if both of the following **complementary slackness** conditions hold:

1. Primal complementary slackness condition: for each  $1 \leq j \leq n$ , either  $x_j = 0$  or  $\sum_{i=1}^m a_{i,j}y_i = c_j$ .
2. Dual complementary slackness condition: for each  $1 \leq i \leq m$ , either  $y_i = 0$  or  $\sum_{j=1}^n a_{i,j}x_j = b_j$ .

To obtain the approximation, we will relax the dual condition, but bound it; thus more feasible dual solutions will be feasible in the primal.

Relaxed dual complementary slackness condition: for each  $1 \leq i \leq m$ , either  $y_i = 0$  or  $\sum_{j=1}^n a_{i,j}x_j = \alpha b_j$  where  $\alpha > 1$ .

**Claim 2.2.1** *If  $x$  and  $y$  are primal and dual feasible solutions satisfying the primal and relaxed dual complementary slackness conditions, then  $\sum_{j=1}^n c_j x_j \leq \alpha \sum_{i=1}^m b_i y_i$ .*

**Proof 2.2.2**  $\sum_{j=1}^n c_j x_j \leq \sum_{j=1}^n (\sum_{i=1}^m a_{i,j} y_i) x_j$

*by the primal slackness condition. Now*

$$\sum_{j=1}^n (\sum_{i=1}^m a_{i,j} y_i) x_j = \sum_{j=1}^n (\sum_{i=1}^m a_{i,j} x_j) y_i \leq \alpha \sum_{i=1}^m b_i y_i$$

*by the relaxed dual slackness condition.*

**Definition 2.2.3** *Let  $f$  be the frequency of the most covered element (the number of times it is covered). For vertex cover, this value cannot be greater than 2, since an edge can only touch two vertices.*

The primal slackness condition for the vertex cover problem can be given as  $\forall s \in V, x_s \neq 0 \rightarrow \sum_e y_{e \in E} = w(v)$ . The relaxed dual condition is  $\forall e \in E, y_e \neq 0 \rightarrow \sum_s x_s \leq \alpha$ . Setting  $\alpha = f \leq 2$ , the relaxed dual condition is always satisfied.

[...]

### 3 Semi-Definite Programming

A matrix  $A \in \mathbf{R}^{n \times n}$  is said to be positive semidefinite if for all vectors  $\mathbf{x} \in \mathbf{R}^n$ ,  $\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$ . A semidefinite program is an optimization problem of the following form:

$$\text{Maximize or minimize } \sum_{i,j} c_{ij} (v_i \cdot v_j)$$

subject to  $\forall k, \sum_{i,j} a_{ijk} (v_i \cdot v_j) = b_k$  where  $\mathbf{V}$  is a matrix in  $\mathbf{R}^{n \times n}$  such that  $\mathbf{x} = \mathbf{V}^T \mathbf{V}$  and  $v_1, \dots, v_n$  are the columns of  $\mathbf{V}$ .

We will assume without proof that semidefinite programming is in  $\mathbf{P}$  within an additive error of  $\epsilon$ .

## 3.1 Approximation for Max Cut Problem

### 3.1.1 Max Cut as Integer Program

Recall the **max cut problem**: given a graph  $G = (V, E)$  with edge weights  $w_{ij}$ , divide the nodes into two sets so as to maximize the weight of the edges in the cut between them.

We will use semidefinite programming to obtain a 0.878-approximation for max cut. The basic idea will be to map each vertex to a vector in a unit sphere, ensuring that vertices connected by edges have large angles between them. Thus, cutting the sphere with a plane will produce a cut with high weight.

First, note that max cut can be given by the following integer program:

$$\begin{aligned} & \text{maximize } 1/2 * \sum_{i < j} w_{ij}(1 - y_i y_j) \\ & \text{subject to } \forall y_i : y_i \in \{-1, 1\}. \end{aligned}$$

We assign  $y_i = 1$  or  $y_i = -1$  based on the side of the cut on which each vertex is located. For any pair of vertices,  $(1 - y_i y_j)$  will be 0 if they are on the same side of the cut, and 2 if they are on opposite sides, thus halving will result in exactly  $w_{ij}$  being added to the sum whenever the edge  $ij$  is part of the cut.

### 3.1.2 Semidefinite Relaxation

We now relax the problem to a semidefinite version. The program is as follows:

$$\begin{aligned} & \text{maximize } 1/2 * \sum_{i < j} w_{ij}(1 - v_i \cdot v_j) \\ & \text{subject to } \forall i : v_i \cdot v_i = 1 \end{aligned}$$

where each  $v_i$  is a vector in  $\mathbf{R}^n$ . The previous integer program is just the case when each  $v_i$  is a one-dimensional vector of unit length.

Solving the semidefinite program lets us obtain an optimal set of vectors  $v_i$ . To obtain a cut from this set of vectors, we first choose a random vector

$r$  uniformly from the unit sphere in  $\mathbf{R}^n$ . If  $r \cdot v_i \geq 0$  we set  $y_i$  in the original integer program to 1, otherwise we set it to -1. The expected weight of a cut produced in this way is:

$$E[W(S)] = \sum_{i < j} w_{ij} P[y_i \neq y_j]$$

The probability that two vectors  $v_i$  and  $v_j$  have dot products with  $r$  of opposite sign (i.e. they fall on opposite sides of it) is

$$P[y_i \neq y_j] = \arccos v_i \cdot v_j / \pi$$

Thus

$$E[W(S)] = \sum_{i < j} w_{ij} \arccos v_i \cdot v_j / \pi$$

The numerical value of

$$\min_{0 \leq x \leq 1} \frac{\frac{1}{\pi} \arccos x}{\frac{1}{2}(1-x)} \text{ is } \geq 0.878.$$

Therefore  $E[W(S)] \geq 0.878 \sum_{i < j} w_{ij} \frac{1}{2}(1 - v_i v_j) \geq 0.878 * OPT$ .

## 3.2 Approximation for 3-coloring

A graph  $G = (V, E)$  is **3-colorable** if and only if

$\exists c : V \rightarrow 1, 2, 3$  such that  $c(x) \neq c(y)$  when  $\exists$  an edge  $xy \in E$ . Approximation algorithms for 3-coloring take a 3-colorable graph as input and return a  $c$ -coloring.

### 3.2.1 $O(\sqrt{n})$ -approximation (Widgerson)

An approximation algorithm that returns an  $O(\sqrt{n})$ -coloring is as follows:

While the maximum degree in a graph  $G$  is greater than some value  $\delta$ , select a vertex in  $G$  of degree greater than  $\delta$ , and two-color its neighborhood with two new colors. Once there are no more such vertices left, use  $\delta + 1$  colors to color the remaining graph.

**Lemma 3.2.1** *A graph with maximum degree  $\delta$  can always be colored with*

$\delta + 1$  colors in polynomial time.

The algorithm is correct since we can always two-color the neighborhood of any vertex, since the original graph  $G$  is 3-colorable. Each time a new vertex is selected, at least  $n/\delta$  vertices are colored. Thus the total number of colors used is  $\frac{2n}{\delta} + \delta + 1$ . If we set  $\delta = \sqrt{(n)}$ , we use  $2\sqrt{n} + \sqrt{n} + 1 = O(\sqrt{n})$  colors.

### 3.2.2 Approximation using semidefinite programming

Similarly to the approach to max cut, we assign to each vertex a vector in  $\mathbf{R}^n$  and define the following semidefinite program:

minimize  $\alpha$

subject to  $v_i \cdot v_j \leq \alpha \forall (i, j) \in E; \forall i \in V, v_i \cdot v_i = 1$ .

Once we have solved this program and made the assignment, the algorithm is as follows:

1. Choose  $t = 2 + \log \Delta$  random hyperplanes where  $\Delta$  is the graph's maximum degree.
2. Color each of the resulting  $2^t$  regions with a different color.

We now need to determine how many vertices sharing an edge may have ended up in the same region.

[...]