Topics in Theoretical Computer Science April 16, 2013

Lecture 14

Lecturer: Ola Svensson *Scribes: Cijo Jose and James Newling*

Copied from the lecture notes of the "Approximation Algorithms and Hardness of Approximation" course (2013):

1. Sections 1-3: "Introduction to SDP" scribed by Marwa El Halabi.
2. Section 4: "SDP (Graph Coloring)" scribed by Marwa El Halabi and Slobodan Mitrovic.

1 Semidefinite Programming and Graph Partitioning

Thus far we have seen how linear programs can aid in devising approximation algorithms for $\mathcal{NP}$-hard optimization problems. In this lecture we introduce a more general class of relaxations which allows variables to be vectors instead of scalars. In particular we introduce a relaxation for problems that can be formulated as *strict quadratic programs*.

Definition 1 (Quadratic program) *A quadratic program (QP) is an optimization problem whose objective function is quadratic, subject to quadratic constraints. If in addition all monomials are of degree 0 or 2, then it is a* ***strict quadratic program****.*

Example 1.1 *Quadratic program*

$$\begin{aligned} &\max \sum_{i \in V} x_i^2 + 5 \sum_{i \in V} x_i \\ &\text{s.t } x_i = x_i^2, x_i \in 0, 1 \quad \forall i \in V. \end{aligned} \tag{1}$$

Example 1.2 *Strict quadratic program*

$$\begin{aligned} &\max \sum_{(i,j) \in E} x_i x_j \\ &\text{s.t } x_i^2 = 1 \quad \forall i \in V. \end{aligned} \tag{2}$$

1.1 Maximum Cut

Given an undirected graph $G = (V, E)$, the goal of the maximum cut problem is to find a partition of the vertices, $(S, \bar{S})$, that maximizes the number of edges crossing the cut, i.e. edges with one endpoint in S and the other endpoint in $\bar{S}$. We denote the number of edges crossing the maximum cut by $\text{OPT}_{\text{Max-Cut}}$. The max cut problem is know to be $\mathcal{NP}$-Hard, so our goal is to find a good polynomial time approximation algorithm for it. Note that $|E|$ is an upper bound on $\text{OPT}_{\text{Max-Cut}}$. Thus, we can achieve a $\frac{1}{2}$-approximation for max cut simply by placing each vertex in S or $\bar{S}$ with probability $1/2$.

The max cut problem can be formulated as a quadratic program:

$$\begin{aligned} &\max \sum_{(i,j) \in E} x_i(1 - x_j) + x_j(1 - x_i) \\ &\text{s.t } x_i \in \{0, 1\} \quad \forall i \in V. \end{aligned} \tag{3}$$

where each variable x_i is set to one if vertex i is in set S, and to zero if in set $\bar{S}$. Note that an edge with both ends in the same set will not contribute to the objective function.

If we relax the integer constraint in this QP, we have the following formulation:

$$\begin{aligned} \max & \sum_{(i,j)\in E} x_i(1-x_j) + x_j(1-x_i) \\ \text{s.t } & x_i \in [0,1] \quad \forall i \in V. \end{aligned} \tag{4}$$

If we can solve this relaxed version optimally, we will still be able to find a max cut. Consider the solution for the relaxed QP, OPT_{rel}. For any vertex $h \in V$, with fractional value assigned to x_h, we can rewrite the objective function (3) as follows. Let $\delta(h)$ denote all the vertices adjacent to vertex h.

$$\sum_{i,j\in E\setminus\delta(h)} x_i(1-x_j) + x_j(1-x_i) + x_h \overbrace{\sum_{j\in\delta(h)} (1-x_j)}^{A} + (1-x_h) \overbrace{\sum_{j\in\delta(h)} x_j}^{B}.$$

Then if $A \geq B$, we round x_h to one, otherwise we round it to zero. Let's denote by OPT_{rd} the solution we get after rounding OPT_{rel}. Note that $\text{OPT}_{\text{rel}} \leq \text{OPT}_{\text{rd}}$, so by solving the relaxed QP for max cut and rounding the solution, we obtain an integral solution that is at least as good as OPT_{rel}, which is at least as good as the true optimal value $\text{OPT}_{\text{max-cut}}$. We can deduce from this that solving this particular relaxed version is also $\mathcal{NP}$-hard, since it boils down to solving exactly the $\mathcal{NP}$-hard max cut problem in polynomial time. So relaxing the integrality constraint does not help here. Instead, we'll relax the max cut problem to a semidefinite program.

Definition 2 (Semidefinite program (SDP)) *A semidefinite program is a convex optmization problem concerned with the optmization of a linear objective function over the intersection of cone of positive semdefinite matrices with an affine space. A cone is defined as a subset of vector space that is closed under multiplication by positive scalars . In other words the subset C of a vector space V is a cone if and only if λx belongs to C for any x in C and any positive scalar λ. If C is positive semidefinite then the cone is called as positive semidefinite cone.*

We recall the definition of positive semidefinite matrix:

Definition 3 *A symmetric matrix $X \in \mathbb{R}^{n\times n}$ is positive semidefinite ($X \succeq 0$) iff for all $y \in \mathbb{R}^n$, $y^T X y \geq 0$.*

Theorem 4 *$X \in \mathbb{R}^{n\times n}$ is a symmetric matrix, then the following are equivalent:*

1. *X is positive semidefinite.*
2. *All eigenvalues of X are non negative.*
3. *$\exists V \in \mathbb{R}^{m\times n}$, $m \leq n$, s.t $X = V^T V$.*

Note that we can compute the eigendecomposition of a symmetric matrix $X = Q\Lambda Q^T$ in polynomial time, thus we can test for positive definiteness in polynomial time. However, the decomposition $V^T V$ is not polynomial time computable, since taking the square root of the diagonal matrix Λ can lead to irrational values. We can get an arbitrarily good approximation of this decomposition, so we can assume we have the exact decomposition in polynomial time, given that this inaccuracy can be included in the approximation factor.

By replacing the variable x_{ij} in a SDP by the inner product of the two vectors v_i and v_j in the decomposition $X = V^T V$ corresponding to entry x_{ij}, we obtain an equivalent vector program:

$$\begin{aligned} \max & \sum_{i,j} c_{ij} x_{ij} \\ \text{s.t } & \sum_{i,j} a_{ijk} x_{ij} = b_k \\ & x_{ij} = x_{ji} \\ & X \succeq 0 \end{aligned} \qquad \Longleftrightarrow \qquad \begin{aligned} \max & \sum_{i,j} c_{ij} (v_i \cdot v_j) \\ \text{s.t } & \sum_{i,j} a_{ijk} (v_i \cdot v_j) = b_k \\ & v_i \in \mathbb{R}^n \end{aligned}$$

The max cut problem admits a strict quadratic program formulation, which can be relaxed to a vector program:

$$\begin{aligned} &\max \sum_{(i,j)\in E} \frac{1 - v_i \cdot v_j}{2} \\ &\text{s.t } v_i \in \{-1, 1\} \end{aligned} \quad \Longrightarrow \quad \begin{aligned} &\max \sum_{(i,j)\in E} \frac{1 - v_i \cdot v_j}{2} \\ &\text{s.t } v_i \cdot v_i = 1 \\ &v_i \in \mathbb{R}^n \end{aligned}$$

1.2 Random Hyperplane Rounding

Given a solution $\{v_u | \ \forall u \in V\}$ to the vector program of max cut with value OPT_v, we round our solution as follows:

- Pick a vector r uniformly at random from a unit hyper sphere.
- For all $u \in V$: $\begin{cases} r \cdot v_u \geq 0 & \Rightarrow u \to S \\ r \cdot v_u < 0 & \Rightarrow u \to \bar{S} \end{cases}$.

This rounding procedure is called **random hyperplane rounding**.

Theorem 5 *There exist a polynomial time algorithm that achieves a 0.878-approximation of the maximum cut with high probability.*

To prove Theorem 5,we will start with the following lemmas.

Lemma 6 *Let $x_1, ..., x_n$ be random variables picked independently from a $\mathcal{N}(0, 1)$. Let $d = \sqrt{x_1^2 + x_2^2 ... + x_n^2}$ then $r = (x_1/d, ..., x_n/d)$ is a random vector distributed uniformly on the n dimensional unit hypersphere.*

Proof The distribution of random variables $x_1, ..., x_n$ that we have picked from $\mathcal{N}(0, 1)$ has the following probablility density function (PDF)

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{\frac{-{x_i}^2}{2}} \tag{5}$$

$$= \frac{1}{2\pi^{\frac{n}{2}}} e^{\frac{-\sum_{i=1}^{n} {x_i}^2}{2}} \tag{6}$$

It is clear that the PDF only depends on the length of the vector not on the direction, which implies that if we sample a vector from the above distribution then it has equal probability of being in any direction and normalizing the vector by its length will yield a uniformly distributed vector on the n-dimensional unit hypersphere. ■

Lemma 7 *The projections of r on to unit vectors e_1 and e_2 are independent iff e_1 and e_2 are orthogonal.*

Proof Let's denote r_1 and r_2 the projections of r onto e_1 and e_2, respectively. The projections of a Gaussian random vector are also Gaussian random vectors, so it's sufficient to have $E[r_1 r_2] = 0$ for r_1 and r_2 to be independent. Since $E[r_1 r_2] = E[(e_1^T r)(r^T e_2)] = e_1^T E[rr^T] e_2 = e_1^T e_2$, lemma follows directly. ■

Corollary 8 *Let r' be the projection of r onto a 2-dimensional plane, then $\frac{r'}{\|r'\|}$ is uniformly distributed on a unit circle in the plane.*

Lemma 9 *The probability that edge (i,j) is cut is $\frac{\arccos(v_i \cdot v_j)}{\pi} = \frac{\theta_{ij}}{\pi}$, where θ_{ij} is the angle between vectors v_i and v_j.*

Proof Project vector r onto the plane containing v_i and v_j. It is easy to see that edge (i,j) is cut iff r falls within the area formed by the vectors perpendicular to v_i and v_j, which has area equal to $2\theta_{ij}/(2\pi)$. ■

Lemma 10 *For $x \in [-1,1]$, one can show that: $\frac{\arccos(x)}{\pi} \geq 0.878\left(\frac{1-x}{2}\right)$.*

Let W be a random variable denoting the weight of the cut we obtain from solving the vector program for max cut and then applying the random hyperplane rounding. Then:

$$\begin{aligned}
E[W] &= E\left[\sum_{(i,j)\in E} \Pr\left[\text{edge } (i,j) \text{ is cut}\right]\right] \\
&= \sum_{(i,j)\in E} \frac{\theta_{ij}}{\pi} && \text{(by lemma 9)} \\
&= \sum_{(i,j)\in E} \frac{\arccos(v_i \cdot v_j)}{\pi} \\
&\geq 0.878\ \frac{1-(v_i \cdot v_j)}{2} && \text{(by lemma 10)} \\
&= 0.878\ \mathrm{OPT}_v \\
&\geq 0.878\ \mathrm{OPT}_{\text{max-cut}}
\end{aligned}$$

Given this expected value, one can show the existence of an algorithm that achieves a 0.878-approximation of the maximum cut in polynomial time, with high probability. This concludes the proof of Theorem 5.

Finally, we give an example to show that this approximation factor is almost tight. Consider a 5-cycle graph. $\mathrm{OPT}_{\text{max-cut}} = 4$, while the optimal solution for the SDP is placing the 5 vector in a 2-dimensional plane with an angle $\frac{2\pi}{5}$ between each two vectors. The approximation factor achieved in this case is 0.884.

1.3 Correlation Clustering

Given an undirected graph $G = (V,E)$, we assign for each edge $(i,j) \in E$ the weights W_{ij}^+ and W_{ij}^- to denote how similar or different the endpoints of this edge are, respectively. (In our analysis we'll assume each edge have only one of these two kind of weights, but the approximation algorithm will still work in general.) The goal of the correlation clustering problem is to find a partition of the graph into clusters of similar vertices. In other words, we aim to maximize the following objective function:

$$\max \sum_{i,j \text{ are in the same cluster}} W_{ij}^+ + \sum_{i,j \text{ are in different clusters}} W_{ij}^-.$$

We denote the optimal value by OPT_{cc}.

The correlation clustering problem also admits a simple $\frac{1}{2}$-approximation algorithm by picking the best of the following two procedures:

1. Form one cluster: $S = V$.
2. Set each vertex to be in its own cluster.

Note that if the total sum of W_{ij}^+ in the graph is greater than the total sum of W_{ij}^-, choice 1 will guarantee at least half OPT_{cc}, otherwise choice 2 will.

The correlation clustering problem admits an exact formulation that can be relaxed to a vector program:

$$\begin{aligned} &\max \sum_{(i,j)\in E} \left(W_{ij}^+(v_i \cdot v_j) + W_{ij}^-(1 - v_i \cdot v_j)\right) \\ &\text{s.t } v_i \in \{e_1, e_2, \cdots, e_n\} \ \ \forall i \in V \end{aligned} \quad \Longrightarrow \quad \begin{aligned} &\max \sum_{(i,j)\in E} \left(W_{ij}^+(v_i \cdot v_j) + W_{ij}^-(1 - v_i \cdot v_j)\right) \\ &\text{s.t } v_i \cdot v_i = 1 \\ &\quad v_i \cdot v_j \geq 0 \\ &\quad v_i \in \mathbb{R}^n \end{aligned}$$

where e_k denotes the unit vector with the kth entry set to one. In the exact formulation, v_i is set to e_k if vertex i belongs to cluster k.

1.4 Rounding Algorithm for Correlation Clustering

Given a solution $\{v_u | \ \forall u \in V\}$ to the vector program of correlation clustering with value OPT_v, we round our solution as follows:
- Pick two random vectors r_1 and r_2 s.t each entry is drawn from the standard distribution $\mathcal{N}(0, 1)$.
- Form 4 clusters as follows:

$$\begin{aligned} R_1 &= \{i \in V : \ r_1 \cdot v_i \geq 0 \text{ and } r_2 \cdot v_i \geq 0\} \\ R_2 &= \{i \in V : \ r_1 \cdot v_i \geq 0 \text{ and } r_2 \cdot v_i \leq 0\} \\ R_3 &= \{i \in V : \ r_1 \cdot v_i \leq 0 \text{ and } r_2 \cdot v_i \geq 0\} \\ R_4 &= \{i \in V : \ r_1 \cdot v_i \leq 0 \text{ and } r_2 \cdot v_i \leq 0\}. \end{aligned}$$

Theorem 11 *There exists a polynomial time algorithm that achieves a $\frac{3}{4}$-approximation of the correlation clustering problem with high probability.*

Proof We start with a useful lemma:

Lemma 12 *For $x \in [0, 1]$, one can show that $\frac{(1-\frac{acos(x)}{\pi})^2}{x} \geq 0.75$ and $\frac{1-(1-\frac{acos(x)}{\pi})^2}{1-x} \geq 0.75$.*

Let x_{ij} be a random variable that takes the value one if i and j are in the same cluster. Note that the probability that v_i and v_j are not separated by either r_1 or r_2, is $(1 - \frac{\arccos(v_i \cdot v_j)}{\pi})^2$. Thus,

$$E[x_{ij}] = (1 - \frac{\arccos(v_i \cdot v_j)}{\pi})^2$$

Let W be a random variable denoting the weight of the clustering we obtain from solving the vector program of correlation clustering and then applying the random 2-hyperplane rounding.

$$\begin{aligned} E[W] &= E[\sum_{(i,j)\in E} W_{ij}^+ x_{ij} + W_{ij}^-(1 - x_{ij})] \\ &= \sum_{(i,j)\in E} W_{ij}^+(1 - \frac{\arccos(v_i \cdot v_j)}{\pi})^2 + W_{ij}^-(1 - (1 - \frac{\arccos(v_i \cdot v_j)}{\pi})^2) \\ &\geq 0.75 \sum_{(i,j)\in E} W_{ij}^+(v_i \cdot v_j) + W_{ij}^-(1 - v_i \cdot v_j) \qquad \text{(by lemma 12)} \\ &= 0.75 \ \text{OPT}_v \\ &\geq 0.75 \ \text{OPT}_{cc}. \end{aligned}$$

Given this expected value, one can show the existence of an algorithm that achieves a 0.75-approximation for the correlation clustering problem in polynomial time, with high probability. ■

2 Exercise 1

Show that the QP: $\max \sum_{(i,j)\in E}(1-x_i)x_j + x_i(1-x_j)$ *such that* $x_i \in [0,1] \forall i \in V$ *solves max cut exactly.* Probabilistically: Consider rounding the solution of the QP x as follows: $x_i^* = 1$ with probability x_i and $x_i^* = 0$ with probability $1 - x_i$. The expected value of the objective function is then $\sum_{i,j} \mathbf{E}\left((1-x_i^*)x_j^* + x_i^*(1-x_j^*)\right) = \sum_{(i,j)\in E}(1-x_i)x_j + x_i(1-x_j)$ by independence of the x_i^* and x_j^*. Thus the expected value after rounding is the optimal before rounding. But as the optimal rounding can never be less than the optimal before rounding, the value after rounding must always be the value before rounding. Thus there is optimal solution which is binary, solving max cut.

3 Exercise 2

Part a: Formulate an SDP relaxation for MAX 2-SAT. Hint: Introduce a fake variable x_0 *that represents true. Part b: Give an* α_{GW}*-approximation algorithm for MAX 2-SAT. Fact: For any* $0 \le \theta \le \pi$*, we have* $(1-\theta/\pi) \ge \frac{\alpha_{GW}}{2}(1+cos(\theta))$

Solution also to be found on page 263 of Vazirani's Approximation Algorithms[1]. We wish to assign true/false to $x_1 \dots x_n$ such that as many of the m clauses. We introduce variables $y_i \in \{1,-1\}$ for $i = 1 \dots n$ and an additional dummy variable $y_0 \in \{1,-1\}$ to represent true, so that Boolean variable x_i is true iff $y_i = y_0$. We will define the *value* of variable x_i as: $v(x_i) = 1$ if x_i is true and $c(x_i) = 0$if x_i is false, so

$$v(x_i) = \frac{1+y_iy_0}{2}$$
$$v(\bar{x_i}) = \frac{1-y_iy_0}{2}.$$

A clause of the form $x_i \vee x_j$ is unsatisfied (false) iff both x_i and x_j are false, so

$$\begin{aligned} v(x_i \vee x_j) &= 1 - v(\bar{x_i})v(\bar{x_j}) \\ &= \dots \\ &= \frac{1}{4}\left[(1+y_iy_0) + (1+y_jy_0) + (1-y_iy_j)\right]. \end{aligned}$$

For a clause of the form $\bar{x_i} \vee x_j$, we have

$$v(\bar{x_i} \vee x_j) = \frac{1}{4}\left[(1-y_iy_0) + (1+y_jy_0) + (1+y_iy_j)\right],$$

with similar expressions for the cases $x_i \vee \bar{x_j}$ and $\bar{x_i} \vee \bar{x_j}$. Maximising the number of satisfied clauses this corresponds to choosing $y_1 \dots y_n$ so as to maximise

$$\sum_{\text{clauses}} (1 \pm y_iy_0) + (1 \pm y_jy_0) + (1 \pm y_iy_j), \tag{7}$$

where in each term in the sum the signs depend on the form of the clause. Eq. (7) is a special case of the more general,

$$\sum_{\substack{i=0\dots n \\ j=0\dots n}} a_{ij}(1+y_iy_j)/2 + b_{ij}(1-y_iy_j)/2.$$

[1] www.cc.gatech.edu/fac/Vijay.Vazirani/book.pdf

We now relax $y_i \in \{-1, 1\}$ to allow $y_i \in R^{n+1}$ with $\|y_i\|_2 = 1$, resulting in a SDP which we solve exactly in polynomial time. It remains to map the returned vectors $y_1 \dots y_n$ to Boolean values, for which the same rounding as in max-cut is used. That is, select $r \in S^n$ uniformly at random and choose x_i according to $\text{sign}(r \cdot y_i)$. Such rounding is roughly motivated by an argument for metric preservation. We show that this is an α_{GW}-approximation algorithm:

As shown when used for max-cut,

$$\begin{aligned}\mathbf{P}(v_i \text{ and } v_j \text{ are separated}) &= \frac{\theta_{ij}}{\pi} \\ &\geq \frac{\alpha_{GW}}{2}\left(1 - \cos(\theta_{ij})\right).\end{aligned}$$

Similarly, using the fact provided,

$$\begin{aligned}\mathbf{P}(v_i \text{ and } v_j \text{ are not separated}) &= 1 - \frac{\theta_{ij}}{\pi} \\ &\geq \frac{\alpha_{GW}}{2}\left(1 + \cos(\theta_{ij})\right).\end{aligned}$$

We use these inequalities in the following calculation,

$$\begin{aligned}&\mathbf{E}(\text{value of returned solution}) \\ &= \sum_{ij} a_{i,j}\mathbf{P}(v_i \text{ and } v_j \text{ are separated}) + \sum_{ij} b_{i,j}\mathbf{P}(v_i \text{ and } v_j \text{ are not separated}) \\ &= \sum_{ij} a_{ij}(1 - \frac{\theta_{ij}}{\pi}) + b_{ij}\frac{\theta_{ij}}{\pi} \\ &\geq \sum_{ij} a_{ij}\frac{\alpha_{GW}}{2}\left(1 + \cos(\theta_{ij})\right) + \sum_{ij} b_{ij}\frac{\alpha_{GW}}{2}\left(1 - \cos(\theta_{ij})\right) \\ &= \alpha_{GW}\left[\sum_{ij} a_{ij}\left(\frac{1 + v_i \cdot v_j}{2}\right) + \sum_{ij} b_{ij}\left(\frac{1 - v_i \cdot v_j}{2}\right)\right] \\ &= \alpha_{GW} OPT_{SDP}.\end{aligned}$$

As $OPT \leq OPT_{SDP}$, we conclude that $\mathbf{E}(\text{value of returned solution}) \geq \alpha_{GW} OPT$, concluding that we indeed have an α_{GW}-approximation algorithm.

4 Coloring 3-Colorable Graphs

What follows in these notes was not covered in 2014 and is included for the interested reader (notes from 2013) In previous lectures, we illustrated how to formulate the max cut problem as a vector programs (or SDP), by relaxing the variables in the exact formulation from scalars to vectors. We showed that solving the SDP up to an arbitrarily small additive error and then rounding the vector solution to a feasible solution for the original problem provides a polynomial time approximation algorithm for this problem. In this lecture, we will continue exploring SDP by studying the problem of coloring a 3-colorable graph, which is a special instance of the general graph coloring problem.

In the graph coloring problem, we are given an undirected graph $G = (V, E)$. Our goal is to find a legal coloring (also called proper coloring, or simply coloring) of the graph with the minimum number of colors. A legal coloring of a graph is a coloring where each vertex is assigned a color such that no two adjacent vertices have the same color. This problem is known to be $\mathcal{NP}$-hard and even finding an approximation with a factor better than $n^{1-\epsilon}$, for any $\epsilon > 0$ is $\mathcal{NP}$-hard. Instead, we focus our attention

on special instances of the graph coloring problem in which we are given a k-colorable graph, and our goal is to find a legal coloring of this graph with the fewest number of colors. In what follows, we will assume G is a 3-colorable graph. Note that it is still $\mathcal{NP}$-hard to decide if a graph can be colored with three colors. It is also $\mathcal{NP}$-hard to find a coloring of a 3-colorable graph with at most five colors. In our discussion, we will focus on finding a coloring better than n.

4.1 Coloring with $O(\sqrt{n})$ Colors

We denote the maximum degree of a graph by Δ.

Lemma 13 *We can efficiently color any graph with $\Delta + 1$ colors, by greedily coloring the graph.*

Given a 3-colorable graph G, we can find a legal coloring using at most $O(\sqrt{n})$ colors.

1. If G has a vertex v with degree $d(v) \geq \sqrt{n}$, we use three new colors to color this vertex and its neighbors $\delta(v)$. Since G is 3-colorable, the neighbors of any vertex v form a bipartite graph (since none of these vertices can have the same color as vertex v). Thus, we can color the set $\delta(v)$ using two colors. We use a third color to color vertex v. We repeat this step until no vertices with degree higher than $\sqrt{n}$ remain.
2. Once G has $\Delta < \sqrt{n}$, we can apply Lemma **??** to color the rest of the graph using $\sqrt{n}$ colors.

Note that Step 1 will be executed at most $\frac{n}{\sqrt{n}}$-times, since at each iteration we are coloring at least $\sqrt{n}$ vertices. At each of these iterations we are using three new colors. So at the end of this phase, we will have used at most $3\sqrt{n}$ colors. In the second step, we use only $\sqrt{n}$ colors. So in total we will color the graph with at most $4\sqrt{n}$ colors.

In the following sections, we note that we will often use $\tilde{O}()$ notation to hide log factors.

4.2 $\tilde{O}(n^{0.631})$ Colors

To find a coloring of a k-colorable graph, we need to partition the graph into sets such that all edges are cut. We want to formulate this problem as an SDP, so we represent each vertex by a vector $v_i \in \mathbb{R}^n$. As we have seen in the previous lecture, if we are partitioning the vertices randomly, two vertices are more likely to be separated if their corresponding vectors are away from each other, which corresponds to a small inner product, so our goal is to minimize the quantity $v_i \cdot v_j$ for any edge $(i, j) \in E$. We formulate the following vector program for the graph coloring problem for a k-colorable graph:

$$\begin{aligned} &\min \lambda \\ &\text{s.t } v_i \cdot v_j \leq \lambda \quad \forall (i,j) \in E \\ &\quad v_i \cdot v_i = 1 \quad \forall i \in V \\ &\quad v_i \in \mathbb{R}^n \end{aligned}$$

When G is 3-colorable, there exists a feasible solution of the above SDP for $\lambda = -\frac{1}{2}$. To see this, note that any legal 3-coloring of G will partition the vertex set into three *independent sets* (an independent set is a set where no two vertices are connected by an edge) where each set correspond to a color. Thus an optimal exact solution for the above SDP would consist of mapping each independent set to one of the 'color' vectors, which are each at an angle of $\frac{2\pi}{3}$ from each other. Thus, any legal coloring of a 3-colorable graph is a feasible solution of the following SDP:

$$\begin{aligned} &v_i \cdot v_j \leq -\frac{1}{2} \quad \forall (i,j) \in E \\ &v_i \cdot v_i = 1 \quad \forall i \in V \\ &v_i \in \mathbb{R}^n \end{aligned}$$

To give a coloring of G, we will solve this SDP and then apply a randomized rounding strategy to color the vertices. The goal in rounding is to avoid using too many colors and at the same time avoid introducing monochromatic edges. We will use the following notion of a *semicoloring*.

Definition 14 (k-Semicoloring) *A k-semicoloring of G is a k-coloring of the vertices such that at most $\frac{n}{4}$ edges have endpoints of the same color.*

Note that any k-semicoloring will directly result in a k-coloring of the rest of the graph after removing all the monochromatic edges, so we obtain a k-coloring of at least $\frac{n}{2}$ vertices.

Lemma 15 *If we can semicolor a graph G with k colors, then we can color G with $k \log n$ colors.*

Proof Each round we use k colors and remove at least half the vertices. There are $\log n$ rounds, so we use at most $k \log n$ colors. ■

Now we are ready to present our approximation algorithm:

1. Solve the SDP on G.
2. Set $t = 2 + \log_3 \Delta$ and pick t random vectors $\{r_1, r_2, \cdots, r_t\}$ where each entry r_i is drawn from the standard normal distribution $\mathcal{N}(0, 1)$.
3. These t vectors define 2^t different regions based on the sign of the dot products with each of the t vectors. We color each of these regions with a different color.

Lemma 16 *Random hyperplane rounding with t hyperplanes produces a semicoloring using $4\ \Delta^{\log_3 2}$ colors with probability at least $1/2$.*

Proof The number of colors we are using is $2^t = 4 \cdot 2^{\log_3 \Delta} = 4\Delta^{\log_3 2}$. Using the trivial upper bound of n on Δ, we see that this is at most $4n^{0.631}$ colors.

We want to show that our coloring procedure produces a semicoloring on G, so we need to show that the number of monochromatic edges is at most $\frac{n}{4}$ with probability at least $1/2$.

$$\begin{aligned}
\Pr[\text{edge } (i,j) \text{ is monochromatic}] &= (1 - \frac{\arccos(v_i \cdot v_j)}{\pi})^t \\
&= (1 - \frac{\arccos(-0.5)}{\pi})^t \\
&= (1 - \frac{1}{\pi}\frac{2\pi}{3})^t \\
&= (\frac{1}{3})^t \\
&= \frac{1}{9\Delta}.
\end{aligned}$$

Let $m = |E|$. Then $m \leq \frac{n\Delta}{2}$. Let X be a random variable denoting the number of monochromatic edges in our coloring. Then:

$$E[X] = \sum_{(i,j)\in E} \Pr[\text{edge } (i,j) \text{ is monochromatic}] \leq \frac{n\Delta}{2} \cdot \frac{1}{9\Delta} = \frac{n}{18}.$$

By Markov's inequality, we have:

$$\Pr\left[X \geq \frac{n}{4}\right] \ \leq \ \frac{E[X]}{n/4} \ \leq \ \frac{2}{9}.$$

Then Lemma 16 holds with probability at least $\frac{7}{9}$. ■

Therefore this algorithm results in a $O(n^{0.631} \log n) = \tilde{O}(n^{0.631})$-approximation which is even worst than the first simple approximation algorithm, but it provides us with a way to combine the above algorithm with the combinatorial algorithm to get a $\tilde{\mathcal{O}}(n^{0.387})$-approximation algorithm.

4.3 $\tilde{O}(n^{0.387})$ Colors

Let θ be some parameter that we define later. The following algorithm merges the two approximation algorithms we presented so far.

1. If G has a vertex v with degree $d(v) \geq \theta$, we use three new colors to color the vertex v and its neighbours $\delta(v)$. We repeat this step until no vertices with degree at least θ remain.
2. Once G has $\Delta < \theta$, we use the second algorithm to color the rest of the graph with 4 $\theta^{\log_3 2} \log n$ colors.

Step 1 will use at most $\frac{3n}{\theta}$ colors. Setting $\frac{n}{\theta} = \theta^{\log_3 2}$, we have $n = \theta^{\log_3 6}$. Then we define θ as $\theta = n^{\log_6 3} \approx n^{0.6131}$. In total, this approximation algorithm uses $\tilde{O}(\frac{n}{\theta}) = \tilde{O}(n^{0.387})$ colors.

4.4 $\tilde{O}\left(n^{1/4}\right)$ Colors via Large Independent Sets

We now show how to color a 3-colorable graph with $\tilde{O}\left(n^{1/4}\right)$ colors in polynomial time. To do this, we use the fact that in a legal coloring, a set of vertices of the same color form an independent set. We use this fact to construct the following general algorithm:

1. Find an independent set I.
2. Color the vertices in I with a new color.
3. Remove I.
4. Repeat all the steps.

If in each iteration of Step 3 of the above algorithm, at least a γ-fraction of vertices are removed, then after k iterations, at most $(1-\gamma)^k n$ vertices remain. If we remove vertices until there is at least one vertex, then using $\log(1-x) \leq -x$ for $|x| > 1$, we derive k_{max}, the maximum number of iterations, as follows:

$$(1-\gamma)^{k_{max}} n \geq 1 \quad \Rightarrow \quad k_{max} \leq \frac{1}{\gamma} \log n.$$

Thus, if we have an algorithm that colors a γ-fraction of the graph at each iteration, then the algorithm will use $O(\frac{1}{\gamma} \ln n)$ colors.

As mentioned previously, we would like to map all the vertices onto vectors in a 2-dimensional plane, so that any two vectors corresponding to the two endpoints of an edge are 120 degrees apart. There are three such vectors, each corresponding to a single color, which are uniquely determined up to a rotation. Since such a solution corresponds to an optimal integral solution and can therefore not solve such an SDP, we relaxed the dimension of vectors onto which the vertices are mapped. However, after the relaxation, there can be more than three vectors satisfying the constraints, and it is no longer straightforward to map these vectors to colors. We still know that two vectors v_i and v_j representing an edge are far away from each other, i.e. $v_i \cdot v_j = -1/2$ for every $(i,j) \in E$. To benefit from such a structural guarantee, we randomly sample an n-dimensional vector r and consider vectors that are close to it. Intuitively,

vectors that are close to r should also be close to each other and therefore form an independent set. More formally, let

$$r = (r_1, \ldots, r_n) \text{ such that } r_i \in \mathcal{N}(0,1), \forall i \in \{1, \ldots, n\},$$

and define

$$S(\epsilon) = \{i \in V : v_i \cdot r \geq \epsilon\},$$

where ϵ will be determined later. Note that $S(\epsilon)$ might not be an independent set. We also define

$$S'(\epsilon) = \{i \in S(\epsilon) : i \text{ has no neighbors in } S(\epsilon)\},$$

which is an independent set. (Observe that a "smarter" way to define $S'(\epsilon)$ would be to find a maximal matc hing in $S(\epsilon)$ and delete the vertices defining the matching. However, it would make our analysis more complicated.) Next, our goal is to estimate the size of $S'(\epsilon)$.

First we recall some basic properties about the normal distribution $\mathcal{N}(0,1)$ such as its probability density and cumulative distribution functions:

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}},$$

$$\Phi(x) = \int_{-\infty}^{x} p(s)\mathrm{d}s \quad \Rightarrow \quad \bar{\Phi}(x) = 1 - \Phi(x) = \int_{x}^{\infty} p(s)\mathrm{d}s.$$

Observe that $v_i \cdot r$, for some vertex $i \in V$, does not depend on the vector v_i since the distribution of r is spherically symmetric. Therefore, we can rotate v_i so that v_i is any vector and without loss of generality, we can assume that $v_i = (1, 0, \ldots, 0)$, and thus $v_i \cdot r = r_1$. Then, $v_i \cdot r$ is distributed as $\mathcal{N}(0,1)$. This implies that:

$$\Pr[i \in S(\epsilon)] = \bar{\Phi}(\epsilon) \quad \Rightarrow \quad E[\ |S(\epsilon)|\ ] = n\,\bar{\Phi}(\epsilon). \tag{8}$$

Now, we prove the following lemma, which gives an upper bound on the probability that a vertex i "fails", i.e. is not included in $S'(\epsilon)$, the independent set picked by our algorithm. In other words, we wish to compute the probability that vertex i does not belong to $S'(\epsilon)$ given that it belongs to $S(\epsilon)$.

Lemma 17

$$Pr[i \notin S'(\epsilon) \mid i \in S(\epsilon)] \leq \Delta\bar{\Phi}\left(\sqrt{3}\epsilon\right). \tag{9}$$

Proof We have

$$\Pr[i \notin S'(\epsilon) \mid i \in S(\epsilon)] = \Pr[\exists (i,j) \in E : v_j \cdot r \geq \epsilon \mid v_i \cdot r \geq \epsilon]. \tag{10}$$

From $v_i \cdot v_j = -1/2$ and $v_i \cdot v_i = v_j \cdot v_j = 1$, we conclude that v_j can be written in the following way

$$v_j = -\frac{1}{2}v_i + \frac{\sqrt{3}}{2}u, \text{ such that } v_i \cdot u = 0 \text{ and } u \cdot u = 1.$$

The last equation implies

$$u = \frac{2}{\sqrt{3}}\left(v_j + \frac{1}{2}v_i\right).$$

If $v_i \cdot r \geq \epsilon$ and $v_j \cdot r \geq \epsilon$, then

$$u \cdot r = \frac{2}{\sqrt{3}}\left(v_j \cdot r + \frac{1}{2}v_i \cdot r\right) \geq \frac{2}{\sqrt{3}}\left(\epsilon + \frac{1}{2}\epsilon\right) = \sqrt{3}\epsilon.$$

Thus, we see that *if* $i \notin S'(\epsilon)$ *and* $i \in S(\epsilon)$ (i.e. if both $v_i \dot{r}$ and $v_j \cdot r$ are at least ϵ), then it must be the case that the projection of r onto u is at least $\sqrt{3}\epsilon$. However, note that if $r \cdot u \geq \sqrt{3}\epsilon$, then this does necessarily imply that both $r \cdot v_i$ and $r \cdot v_j$ are at least ϵ.

Since $u \cdot v_i = 0$, $r \cdot u$ and $r \cdot v_i$ are independently distributed. Thus, we have:

$$\begin{aligned} \Pr\left[v_j \cdot r \geq \epsilon | v_i \cdot r \geq \epsilon\right] &\leq \Pr\left[u \cdot r \geq \sqrt{3}\epsilon | v_i \cdot r \geq \epsilon\right] \\ &= \Pr\left[u \cdot r \geq \sqrt{3}\epsilon\right] \\ &= \bar{\Phi}(\sqrt{3}\epsilon). \end{aligned}$$

To conclude the proof, we use a union bound over the neighbors of i. Since there are at most Δ of them, we have:

$$\Pr\left[\exists (i,j) \in E : v_j \cdot r \geq \epsilon \mid v_i \cdot r \geq \epsilon\right] \leq \sum_{j:(i,j)\in E} \Pr\left[v_j \cdot r \geq \epsilon \mid v_i \cdot r \geq \epsilon\right] \leq \Delta \bar{\Phi}\left(\sqrt{3}\epsilon\right),$$

which implies the lemma.■

So, how should we choose ϵ? Intuitively, a smaller value of ϵ should result in the smaller number of edges in $S(\epsilon)$, since two adjacent vertices should be far from each other. On the other hand, we would like $S(\epsilon)$ to be large. We will set ϵ so that

$$\Delta \bar{\Phi}\left(\sqrt{3}\epsilon\right) \leq 1/2. \tag{11}$$

This will upper bound the probability that a vertex in $S(\epsilon)$ does not belong to $S'(\epsilon)$. Note that, along with (8) and Lemma 17, it would further imply

$$E[|S'(\epsilon)|] \geq \frac{E[|S(\epsilon)|]}{2} = n\frac{\bar{\Phi}(\epsilon)}{2}. \tag{12}$$

Before we state the main result of this section, we state the following lemma without a proof.

Lemma 18 *For* $x > 0$*,*

$$\frac{x}{1+x^2} p(x) \leq \bar{\Phi}(x) \leq \frac{1}{x} p(x).$$

Now we can state the following theorem which lower bounds the size of $S'(\epsilon)$.

Theorem 19

$$E[|S'(\epsilon)|] \geq \Omega\left(n\, \Delta^{-\frac{1}{3}} (\ln \Delta)^{-\frac{1}{2}}\right).$$

Proof First, we derive ϵ so that (11) holds. By Lemma 18 we have

$$\bar{\Phi}\left(\sqrt{3}\epsilon\right) \leq \frac{1}{\sqrt{3}\epsilon} \frac{1}{\sqrt{2\pi}} e^{-\frac{3\epsilon^2}{2}}. \tag{13}$$

From (11) and (13) we conclude it is sufficient to set $\epsilon = \sqrt{\frac{2}{3}\ln \Delta}$.

To estimate the size of $S'(\epsilon)$, we must lower bound the value of $\Phi(\epsilon)$. We obtain the following bound by applying Lemma 18 and using inequality $\frac{x}{1+x^2} \geq \frac{1}{2x}$, for $x \geq 1$, as follows

$$\begin{aligned} \bar{\Phi}(\epsilon) &\geq p(\epsilon) \frac{\epsilon}{1+\epsilon^2} \\ &= \frac{1}{\sqrt{2\pi}} e^{-\frac{\epsilon^2}{2}} \frac{\epsilon}{1+\epsilon^2} \\ &\geq \frac{1}{\sqrt{2\pi}} e^{-\frac{\ln \Delta}{3}} \frac{1}{2\epsilon} \\ &\geq \Delta^{-\frac{1}{3}} (\ln \Delta)^{-\frac{1}{2}}. \end{aligned}$$

Applying the inequality in Equation (12) concludes the proof.■

Theorem 19 says that in each iteration of the algorithm, we are able to find an independent set of size at least $\tilde{O}\left(n\,\Delta^{-1/3}\right)$. This implies that we can color a given 3-colorable graph with $O\left(\Delta^{1/3}\log n\right) = \tilde{O}\left(\Delta^{1/3}\right)$ colors. We can summarize the given approach, which we call Independent-Set-Coloring, as follows:

1. Solve SDP.
2. Pick a random vector r.
3. Pick a set $S(\epsilon)$ which is ϵ-close to r.
4. Let $S'(\epsilon) \subseteq S(\epsilon)$ be vertices with degree zero in $S(\epsilon)$.
5. Remove the vertices in the independent set $S'(\epsilon)$ from the graph.
6. Repeat from Step 2 while the graph is non-empty.

Note that there is no need to resolve SDP in Step 1 in each iteration, since a solution for the initial graph is valid for any subgraph.

Finally, we combine the algorithm from Section 4.1 with the algorithm Independent-Set-Coloring.

1. If G has a vertex v with degree $d(v) \geq n^{3/4}$, we use three new colors to color the vertex v and its neighbours $\delta(v)$. We repeat this step until no vertices with degree at least $n^{3/4}$ remain.
2. Once G has $\Delta < n^{3/4}$, we use the algorithm Independent-Set-Coloring to color the rest of the graph with at most $\tilde{O}((n^{3/4})^{1/3}) = \tilde{O}(n^{1/4})$ colors.

Observe that Step 1. will execute at most $n/n^{3/4} = n^{1/4}$ times, using at most $n^{1/4}$ colors. Thus, the total number of colors used is at most $\tilde{O}(n^{1/4})$.