6.891 Approximation Algorithms	March 3, 2000
Lecture 5	
Lecturer: Michel Goemans	*Scribe: Nitin Thaper*

1 Semidefinite Programming

In the last lecture, we saw a $\frac{1}{2}$-approximation algorithm for the MAX CUT problem. We can't do any better because the total weight of the edges isn't a tight upper bound on the optimal solution.

We now turn to *convex* programming relaxations of various problems; in particular, to something called *semidefinite programming.*

First we define a positive semidefinite (psd) matrix X; we sometimes write $X \succeq 0$ to denote that X is psd.

Definition 1 *A matrix $X \in \Re^{n\times n}$ is positive semidefinite (psd) iff*
$\forall a \in \Re^n, a^T X a \geq 0.$

If $X \in \Re^{n\times n}$ is a symmetric matrix, then the following are equivalent:

1. X is psd;
2. X has non-negative eigenvalues;
3. $X = V^T V$ for some $V \in \Re^{m\times n}$, where $m \leq n$.

A semidefinite program (SDP) can be formulated as

$$
\begin{array}{lll}
\text{Max or Min} & \sum c_{ij} x_{ij} & \\
\text{subject to:} & & \\
 & \sum_{i,j} a_{ijk} x_{ij} = b_k & \forall k \\
 & X = (x_{ij}) \succeq 0 & \text{and } X \text{ is symmetric}
\end{array}
$$

Note that the last constraint, $X \succeq 0$, can be viewed as an infinite set of linear constraints. This is what prevents a SDP from being cast into an equivalent LP.

SDP's have the useful property that they can be solved in polynomial time using

- the ellipsoid method
- modifications of interior-point methods that are used to solve LP's

to within an additive error of ϵ. The running time depends only on $\log(1/\epsilon)$. This additive error of ϵ is necessary because sometimes the solutions to SDP's may be irrational numbers. SDPs also need to meet some additional specialized conditions (e.g., having an interior point in both the primal and the dual) to be solvable in this way, but all the SDPs we consider will be well-behaved, and so we will ignore these additional conditions.

SDP is equivalent to *vector programming* (VP) which can be formally stated as

$$\begin{array}{lll} \text{Max or Min} & \sum c_{ij}(\vec{v_i} \cdot \vec{v_j}) & \\ \text{subject to:} & & \\ & \sum_{i,j} a_{ijk}(\vec{v_i} \cdot \vec{v_j}) = b_k & \forall k \\ & \vec{v_i} \in \Re^n & \forall i \end{array}$$

This follows since X is psd and X is symmetric iff $X = V^T V$ i.e., iff $x_{ij} = \vec{v_i} \cdot \vec{v_j}$ where

$$V = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \vec{v_1} & \vec{v_2} & \cdots & \vec{v_3} \\ \vdots & \vdots & & \vdots \end{pmatrix}$$

(That is, the $\vec{v_i}$ are the column vectors of V). So if we have a feasible solution to SDP, then we have a solution to VP with the same value and vice versa. Note that it's straightforward to derive the SDP solution from the VP solution but the reverse process is a bit more involved. It involves Cholesky factorization of psd matrix X to derive the vectors v_i. Moreover, the vectors so derived are not necessarily unique.

1.1 MAX CUT using Semidefinite Programming

We now consider applying semidefinite programming to the MAX CUT problem. Let us consider the following formulation of the MAX CUT problem which we denote by (MC).

$$\begin{array}{llll} & \text{Max} & \frac{1}{2}\sum_{i<j} w_{ij}(1 - y_i \cdot y_j) & \\ (MC) & & y_i \in \{-1, +1\} & \forall i \end{array}$$

We claim that if we can solve (MC), then we can solve the MAX CUT problem.

Claim 1 *The formulation (MC) models MAX CUT.*

Proof: Consider the cut given by $S = \{i \in V | y_i = -1\}$. We have

$$\frac{1}{2}\sum_{i<j} w_{ij}(1 - y_i \cdot y_j) = \frac{1}{2}\sum_{i<j: y_i = y_j} w_{ij}(1 - y_i \cdot y_j) + \frac{1}{2}\sum_{i<j: y_i \neq y_j} w_{ij}(1 - y_i \cdot y_j)$$

This is because $y_i \in \{-1,+1\}$ for all i. So for a given i and j, either $y_i = y_j$ or $y_i \neq y_j$. If $y_i = y_j$, then $1 - y_i \cdot y_j = 0$ and if $y_i \neq y_j$, then $1 - y_i \cdot y_j = 2$. So in the above expression, the first term becomes zero. So we have

$$\begin{aligned}
\frac{1}{2}\sum_{i<j} w_{ij}(1 - y_i \cdot y_j) &= \frac{1}{2}\sum_{i<j: y_i \neq y_j} w_{ij}(1 - y_i \cdot y_j) \\
&= \frac{1}{2}\sum_{i<j: y_i \neq y_j} w_{ij} \cdot 2 \\
&= \sum_{i<j: i\in S, j\notin S \text{ or } i\notin S, j \in S} w_{ij}
\end{aligned}$$

□

Let us now consider a vector programming relaxation (denoted (VP)) of (MC).

$$(VP) \qquad \begin{aligned}
Z_{VP} \quad = \quad \text{Max} \quad & \frac{1}{2}\sum_{i<j} w_{ij}(1 - \vec{v_i} \cdot \vec{v_j}) & \\
& \vec{v_i} \cdot \vec{v_i} = 1 & \forall i \\
& \vec{v_i} \in \Re^n & \forall i
\end{aligned}$$

To see that (VP) is indeed a relaxation of (MC), we can view the y_i's in (MC) as 1-dimensional vectors and so anything that is feasible for (MC) is feasible for (VP). Also note that this implies $Z_{VP} \geq OPT$.

The equivalent semidefinite programming formulation (denoted (SDP)) is given below.

$$(SDP) \qquad \begin{aligned}
Z_{SDP} \quad = \quad \text{Max} \quad & \frac{1}{2}\sum_{i<j} w_{ij}(1 - Y_{ij}) & \\
& Y_{ii} = 1 & \forall i \\
& Y \succeq 0 &
\end{aligned}$$

Let us look at an example to see how well Z_{VP} (Z_{SDP}) approximates OPT.

Consider the graph C_5 (the cycle graph on 5 vertices). Assume all edges of the graph have weight 1. Then it's easy to see by inspection that S is an optimum cut and $OPT = 4$ (see Figure 1).

It can be shown that the optimal VP solution for this graph lies in $\Re^2$. The optimal vectors are shown graphically in Figure 2. Note that $\angle v_i v_{i+1} = \frac{4\pi}{5}$.

In this case,

$$Z_{VP} = 5\left(\frac{1 - \cos(\frac{4\pi}{5})}{2}\right) = \frac{25 + 5\sqrt{5}}{8} = 4.52$$

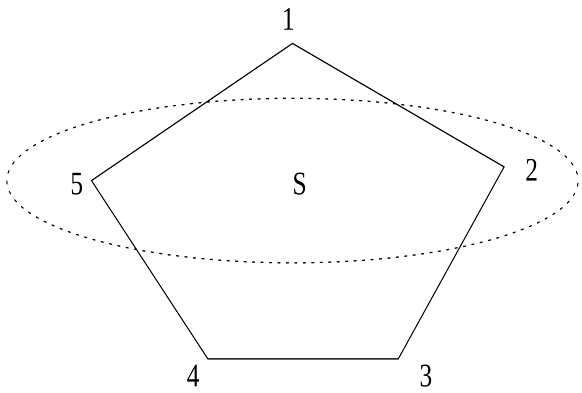

Figure 1: MAX CUT for C_5

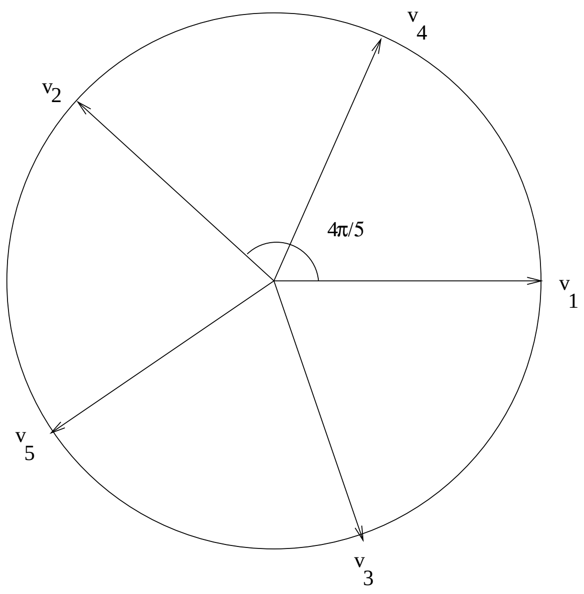

Figure 2: Optimal solution vectors for C_5

Therefore,

$$\frac{OPT}{Z_{VP}} = 0.884$$

This sets an upper bound on how well we can perform.

We can solve (SDP) (and hence (VP)) in polynomial time, but not (MC). So how do we convert a solution of (VP) to a solution of (MC) ? To do this, we would like to apply randomized rounding in some way.

Consider the following algorithm:

VectorRound

Solve vector programming problem (VP) and get vectors $\vec{v^*}$
Choose a random vector $\vec{r}$ uniformly from the unit n-sphere
$S \leftarrow \emptyset$
for $i \leftarrow 1$ to n
  if $\vec{v_i^*} \cdot \vec{r} \geq 0$
    $S \leftarrow S \cup \{i\}$

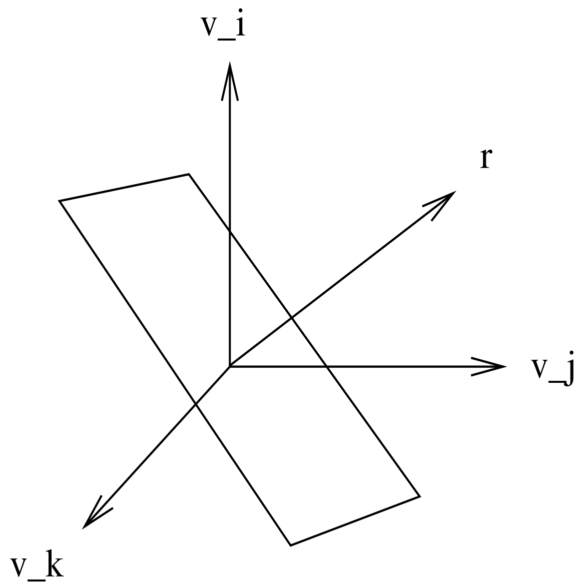

Figure 3: A random hyperplane example

The vector $\vec{r}$ is a normal to some hyperplane. So everything that has a non-negative dot product with $\vec{r}$ will be on one side of the hyperplane and everything that has a negative dot product with $\vec{r}$ will be on the other side (see Figure 3). To get vector $\vec{r}$, choose $\vec{r} = (r_1, r_2, \cdots, r_n)$, such that $r_i \in \mathcal{N}(0,1)$ where $\mathcal{N}$ is a normal distribution. (The normal distribution $\mathcal{N}(0,1)$ can be simulated using uniform distribution on $[0,1]$.)

Theorem 2 *VectorRound is a* 0.878*-approximation algorithm.*

To prove this theorem we require a couple of facts and a couple of lemmas which we give below.

Fact 1 $\frac{\vec{r}}{||\vec{r}||}$ *(i.e., normalization of* $\vec{r}$*) is uniformly distributed over a unit sphere.*

Fact 2 *The projection of* $\vec{r}$ *onto two lines* l_1 *and* l_2 *are independent and normally distributed iff* l_1 *and* l_2 *are orthogonal.*

Corollary 3 *Let* $\vec{r'}$ *be the projection of* $\vec{r}$ *onto a plane.* $\frac{\vec{r'}}{||\vec{r'}||}$ *is uniformly distributed on a unit circle on the plane (see Figure 4).*

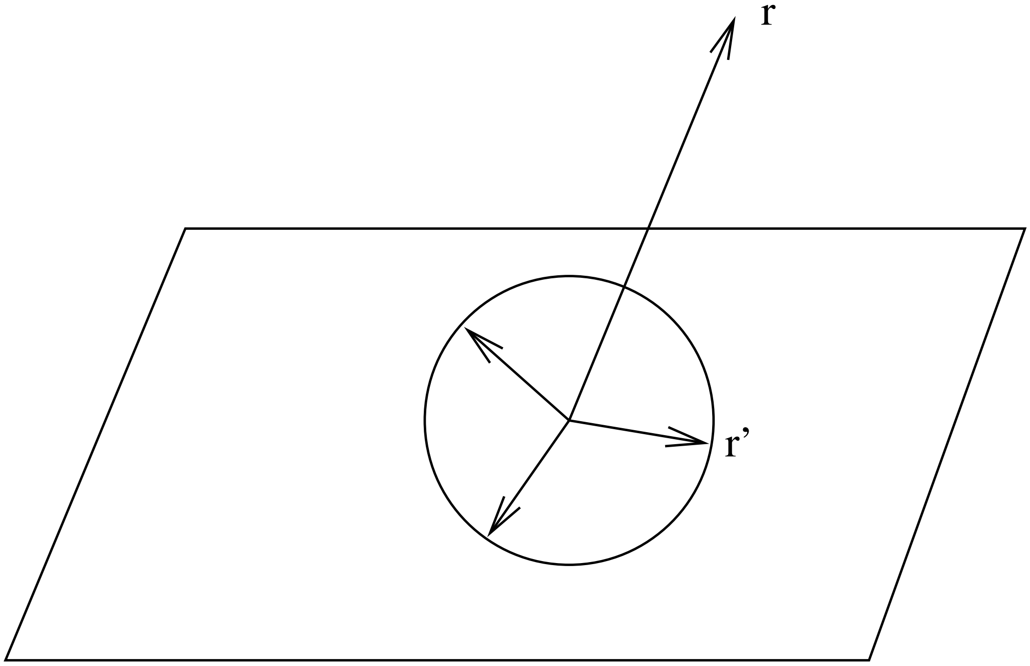

Figure 4: Projection of r to r'

Lemma 4 $\Pr[i \in S, j \notin S \;\; or \;\; i \notin S, j \in S] = \frac{1}{\pi}\arccos(\vec{v_i^*} \cdot \vec{v_j^*})$.

Proof:

Let $\vec{r'}$ be the projection of $\vec{r}$ onto the plane defined by $\vec{v_i^*}$ and $\vec{v_j^*}$.

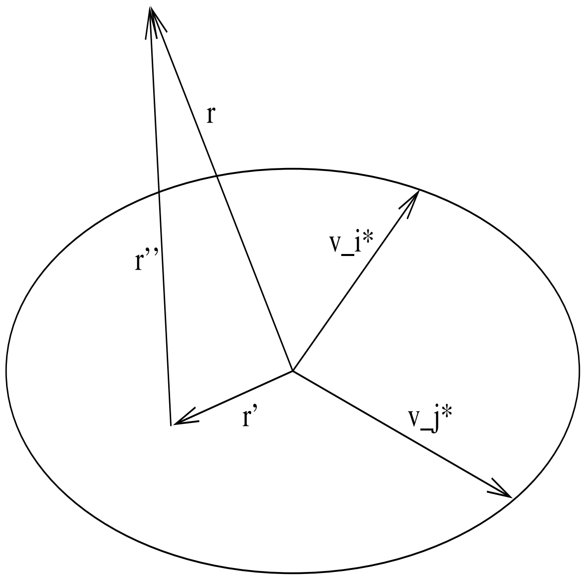

Figure 5: Projection of r into plane defined by v_i^* and v_j^*

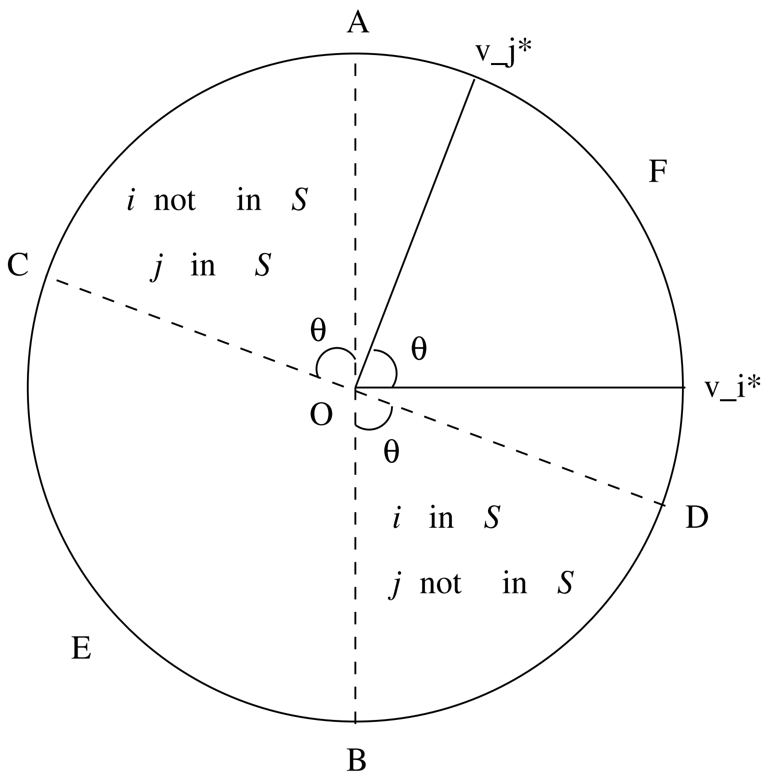

Figure 6: Determining the probability

If $\vec{r} = \vec{r'} + \vec{r''}$ (Figure 5), then

$$\begin{aligned} \vec{v_i^*} \cdot \vec{r} &= \vec{v_i^*} \cdot (\vec{r'} + \vec{r''}) \\ &= \vec{v_i^*} \cdot \vec{r'} \end{aligned}$$

The second equality follows because $\vec{r''}$ is orthogonal to $\vec{v_i^*}$. Similarly, $\vec{v_j^*} \cdot \vec{r} = \vec{v_j^*} \cdot \vec{r'}$.

There are a total of 2π possible orientations of $\vec{r'}$. If $\vec{r'}$ lies on the semi-circular plane AFB (see Figure 6) then $\vec{v_i^*} \cdot \vec{r'} \geq 0$ and so $i \in S$. If $\vec{r'}$ lies on the semi-circular plane AEB, then $i \notin S$. Likewise, if $\vec{r'}$ lies on the semi-circular plane CFD, then $j \in S$ and if $\vec{r'}$ lies on the semi-circular plane CED, then $j \notin S$. Let θ be the angle between the vectors $\vec{v_i^*}$ and $\vec{v_j^*}$. So by construction, $\widehat{AOC} = \widehat{BOD} = \theta$. Note that in the sector AOC, $i \notin S$ and $j \in S$ and in the sector BOD, $i \in S$ and $j \notin S$. Therefore, 2θ of the orientations out of a total of 2π orientaions for $\vec{r'}$ cause $i \in S, j \notin S$ or $i \notin S, j \in S$. Therefore the required probability is $\frac{2\theta}{2\pi}$. We have $\vec{v_i^*} \cdot \vec{v_j^*} = \|\vec{v_i^*}\| \cdot \|\vec{v_j^*}\| \cos\theta$. Therefore, $\theta = \arccos(\vec{v_i^*} \cdot \vec{v_j^*})$, since the $\vec{v_i^*}$'s are unit vectors. Hence the result. $\square$

Lemma 5

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

Proof: Using Mathematica! $\square$

So now we can prove the theorem.

Theorem 6 *(Goemans, W '95) VectorRound is a* 0.878*-approximation algorithm.*

Proof: Consider the random variables

$$X_{ij} = \begin{cases} 1 & \text{if } i \in S, j \notin S \text{ or } i \notin S, j \in S \\ 0 & \text{otherwise} \end{cases}$$

and

$$W = \sum_{i<j} w_{ij} X_{ij}.$$

Then

$$\begin{aligned} E[W] &= \sum_{i<j} w_{ij} \cdot \Pr[i \in S, j \notin S \text{ or } i \notin S, j \in S] \\ \text{(1)} \qquad &= \sum_{i<j} w_{ij} \frac{1}{\pi} \arccos(\vec{v_i^*} \cdot \vec{v_j^*}) \\ \text{(2)} \qquad &\geq 0.878 \cdot \frac{1}{2} \sum_{i<j} w_{ij} (1 - \vec{v_i^*} \cdot \vec{v_j^*}) \\ &= 0.878 \cdot Z_{SDP} \\ &\geq 0.878 \cdot OPT, \end{aligned}$$

where (1) follows by Lemma 4 and (2) follows by Lemma 5. $\square$

Corollary 7 *For any graph with non-negative weights,*

$$\frac{OPT}{Z_{SDP}} \geq 0.878.$$

Can we do better than this ? We saw earlier that for the 5-cycle, C_5,

$$\frac{OPT}{Z_{SDP}} = \frac{32}{25 + 5\sqrt{5}} \approx 0.884.$$

which gives us an upper bound on how well any SDP for this problem can perform.

Can we do a tighter analysis of VectorRound and reach that bound ?

Theorem 8 *(Karloff '95) There exist graphs G for which,*

1. $OPT = Z_{SDP}$
2. $\frac{E[W]}{OPT} = \frac{E[W]}{Z_{SDP}}$ *is arbitrarily close to 0.878*

This implies that as long as we obtain the cut by choosing a random hyperplane, the preceding analysis is tight.

So far, no better approximation algorithm is known for this problem. However, some very recent work shows that the worst case of the algorithm is quite confined: in many cases, we can get an approximation algorithm with a performance guarantee better than 0.878.

Theorem 9 *(Zwick, 10/7/98) If for a graph G*

$$\frac{Z_{SDP}}{\sum_{i<j} w_{ij}} < .84 - \epsilon$$

then there is a $(.878 + f(\epsilon))$-approximation algorithm, where f is an increasing function.

Compare this with previously known work:

Theorem 10 *(Goemans, W '95) If for a graph G*

$$\frac{Z_{SDP}}{\sum_{i<j} w_{ij}} > .84 + \epsilon$$

then VectorRound is a $(.878+g(\epsilon))$-approximation algorithm, where g is an increasing function.

It turns out that there is a limit on how well we can do in any case.

Theorem 11 *(Håstad '97) If $\exists$ an α-approximation algorithm for MAX CUT, $\alpha > \frac{16}{17} \approx 0.941$, then $P = NP$.*

Theorem 12 *(Håstad '97) If $\exists$ an α-approximation algorithm for MAX CUT', $\alpha > \frac{11}{12} \approx 0.916$, then $P = NP$.*

where MAX CUT$'$ is a variant of MAX CUT in which we have two kinds of edges, say red and blue, and the objective is to maximize the weight of red edges *in* the cut and blue edges *not in* the cut. It turns out that VectorRound can be modified to solve this problem with the same performance guarantee. So for this problem, VectorRound is much closer to the theoretical limit.

Research Question: Can you get a 0.878-approximation algorithm without solving an SDP?

1.2 Graph Coloring

Next we will show that semidefinite programming can be applied to the graph coloring problem. In particular, we will show the following result:

Theorem 13 *(Karger, Motwani, Sudan '94) There is a polytime algorithm to color a 3-colorable graph with $\tilde{O}(n^{1/4})$ colors.*

The previous best algorithms used $\tilde{O}(n^{3/8})$ colors (Blum '94) and $O(\sqrt{n})$ colors (Wigderson '83).

Definition 2 *A function $f(n) = \tilde{O}(g(n))$ when the following is valid:*

$$\exists \quad n_0, c_1, c_2 \quad s.t. \quad \forall \quad n \geq n_0 \qquad f(n) \leq c_1 g(n) \log^{c_2} n$$

The following facts are known about coloring graphs.

- We can color 2-colorable (aka bipartite) graphs in polytime.
- We can color G with $\Delta + 1$ colors ($\Delta =$ max degree of G) in polytime.

Proof: Color greedily (color with color not used by neighbors yet). $\square$

The following coloring algorithm for 3-colorable graphs was also previously known.

```
Color1

    While ∃v ∈ G  s.t.  deg(v) ≥ √n
        Color v with color #1
        Color neighbors of v in polytime with 2 new colors
        Remove v and its neighbors from graph
    Color remaining G with √n new colors
```

Theorem 14 *(Widgerson '83) Color1 colors 3-colorable graphs in polytime with $O(\sqrt{n})$ colors.*

Proof: We can execute the while loop at most $\frac{n}{\sqrt{n}}$ times, since we remove at least $\sqrt{n}$ vertices from the graph every time. Hence we use $1 + 2\frac{n}{\sqrt{n}}$ colors in the while loop. The last step takes $\sqrt{n}$ colors by the fact above (since the maximum degree is $\sqrt{n} - 1$), so the total numbers of colors needed is $3\sqrt{n} + 1$. □

We now think about applying semidefinite programming to the problem of coloring 3-colorable graphs. Consider the following vector program:

$$\begin{array}{llll} & \text{Min} & \lambda & \\ \text{subject to:} & & & \\ & & v_i \cdot v_j \leq \lambda & \forall \quad (i,j) \in E \\ & & v_i \cdot v_i = 1 & \forall \quad i \\ & & v_i \in \Re^n & \end{array}$$

Claim 15 *For a 3-colorable graph $\lambda \leq -\frac{1}{2}$.*

Proof: Consider an equilateral triangle, and associate the vectors for the three different colors with the three different vertices of the triangle. Note that the angle between any two vectors of the same color is 0, while the angle between any two vectors of different color is $2\pi/3$. Then for v_i, v_j such that $(i,j) \in E$

$$\begin{aligned} v_i \cdot v_j &= \|v_i\| \, \|v_j\| \cos\theta \\ &= \cos\left(\frac{2\pi}{3}\right) \\ &= -\frac{1}{2}, \end{aligned}$$

So this solution is a feasible solution to the vector program with $\lambda = -1/2$. Thus in the optimal solution, $\lambda \leq -1/2$. □

Now we need a way to round the solution to this vector program into a valid coloring. As before we will consider randomized algorithms. It turns out that it is too much to expect that we will get an algorithm that colors the whole graph correctly with high probability. Instead, we will aim for an algorithm that colors mostly correctly. In particular, we want a *semicoloring*, which means that at most $\frac{n}{4}$ edges have the same colored endpoints. In such a solution at least $\frac{n}{2}$ of the vertices are colored "correctly" (any edge between these vertices has differently colored endpoints).

Note then if we can semicolor a graph with k colors, then we can color the graph with $k \log n$ colors: we obtain a semicoloring of the graph with k colors, and take the half of the graph colored correctly. We then semicolor the remaining half of the graph with k new colors, and take the half colored correctly, and so on. This takes $\log n$ iterations, after which the graph is colored correctly with $k \log n$ colors.

Consider now the following rounding algorithm.

KMS1

> Solve vector program, get v_i
> Pick $t = 2 + \log_3 \Delta$ random vectors $r_1, \ldots, r_t$
> Let $R_1 = \{i : r_1 \cdot v_i \geq 0, r_2 \cdot v_i \geq 0, \ldots, r_t \cdot v_i \geq 0\}$
> $R_2 = \{i : r_1 \cdot v_i < 0, r_2 \cdot v_i \geq 0, \ldots, r_t \cdot v_i \geq 0\}$
> $\vdots$
> $R_{2^t} = \{i : r_1 \cdot v_i < 0, r_2 \cdot v_i < 0, \ldots, r_t \cdot v_i < 0\}$
> Color vertices in R_i with color i

The algorithm creates t random hyperplanes and assigns a particular color to the vectors lying in each of the 2^t regions created by the intersecting hyperplanes.

Theorem 16 *(Karger, Motwani, Sudan '94) The algorithm KMS1 gives a semicoloring of $O(\Delta^{\log_3 2})$ colors with probability $\frac{1}{2}$.*

Proof: Since we used 2^t colors, this is $2^t = 4 \times 2^{\log_2 \Delta} = 4\Delta^{\log_3 2}$ colors. Now

$$\begin{aligned}
&\Pr[i \text{ and } j \text{ get the same color for edge } (i,j)] \\
&= \left(1 - \frac{1}{\pi}\arccos(v_i \cdot v_j)\right)^t \\
&\leq \left(1 - \frac{1}{\pi}\arccos(\lambda)\right)^t \\
&\leq \left(1 - \frac{1}{\pi}\arccos(-\frac{1}{2})\right)^t \\
&= \left(1 - \frac{1}{\pi}\frac{2\pi}{3}\right)^t \\
&= \left(\frac{1}{3}\right)^t \\
&\leq \frac{1}{9\Delta}.
\end{aligned}$$

This follows since for any particular random vector r_k, we know (from the analysis for MAX CUT) that the probability that $v_i \cdot r_k \geq 0$ and $v_j \cdot r_k \geq 0$ OR that $v_i \cdot r_k < 0$ and $v_j \cdot r_k < 0$ is $1 - \frac{1}{\pi}\arccos(v_i \cdot v_j)$. Thus the probability that v_i and v_j get the same color is the probability that this happens for each of the t vectors, which is $(1 - \frac{1}{\pi}\arccos(v_i \cdot v_j))^t$, since each of these events is independent.

Let m denote the number of edges in the graph. Note that $m \leq n\Delta/2$. Thus

$$E[\#\text{ bad edges}] \leq \frac{m}{9\Delta} \leq \frac{\frac{\Delta n}{2}}{9\Delta} \leq \frac{n}{8},$$

and therefore

$$\Pr[\text{ more than } \tfrac{n}{4} \text{ bad edges}] \leq \frac{1}{2}.$$

where the last result follows from an application of Markov's inequality. $\square$

If we just plug in n for Δ, this gives us an algorithm that colors with $\tilde{O}(n^{\log_3 2}) = \tilde{O}(n^{.631})$, which is worse than Widgerson's algorithm. But we can use Widgerson's technique to make things better:

Color2

> While $\exists v \in G$ s.t. $\deg(v) \geq \sigma$
>   Color v with color #1
>   2-color neighbors of v in polytime with 2 new colors
>   Remove v & neighbors
> Apply KMS1 to color remaining graph with $O(\sigma^{\log_3 2})$ colors

Let's analyze this algorithm. The While loop uses $O(\frac{n}{\sigma})$ colors, since we remove σ vertices from the graph each time. The final step uses $\tilde{O}(\sigma^{\log_3 2})$ colors. To balance these two parts, we set σ such that $\frac{n}{\sigma} = \sigma^{\log_3 2}$, which gives $\sigma = n^{0.613}$. This gives a coloring with $\tilde{O}(n^{0.387})$ colors.

But this algorithm is still worse than Blum (whose algorithm uses $\tilde{O}(n^{\frac{3}{8}})$ colors)! We consider next the following algorithm:

KMS2

> Solve vector program, get vectors v_i
> Pick $t = \tilde{O}(\Delta^{\frac{1}{3}})$ random vectors $r_1, \ldots, r_t$
> Assign vector v_i to random r_j that maximizes $v_i \cdot r_j$
> Color vectors assigned to r_j with color j

Theorem 17 *(Karger, Motwani, Sudan)*

$$\Pr[i \textit{ and } j \textit{ get same color for edge } (i,j)] = \tilde{O}(t^{-3}) = \tilde{O}(\Delta^{-1})$$

We omit the proof of this theorem. To see that this theorem leads to a better algorithm, note that if we use $t = \tilde{O}(\Delta^{\frac{1}{3}})$ vectors, for an appropriate choice of the right constants, we get that

$$\Pr[i \text{ and } j \text{ get same color for edge } (i,j)] \leq \frac{1}{9\Delta},$$

just as with the previous algorithm, and the previous analysis goes through, except now our algorithm uses $\tilde{O}(\Delta^{\frac{1}{3}})$ colors. If we now apply Widgerson's technique using this algorithm, we get an algorithm that colors the graph with $\tilde{O}(n^{\frac{1}{4}})$ colors.