25
Approximation Algorithms via SDPs

Just like the use of linear programming was a major advance in the design of approximation algorithms, specifically in the use of linear programs in the relax-and-round framework, another significant advantage was the use of semidefinite programs in the same framework. For instance, the approximation guaranteee for the Max-Cut problem was improved from 1/2 to 0.878 using this technique. Moreover, subsequent results have shown that any improvements to this approximation guarantee in polynomial-time would disprove the Unique Games Conjecture.

25.1 Positive Semidefinite Matrices

The main objects of interest in semidefinite programming, not surprisingly, are positive semidefinite matrices.

Definition 25.1 (Positive Semidefinite Matrices). Let $A \in \mathbb{R}^{n \times n}$ be a real-valued symmetric matrix and let $r = \operatorname{rank}(A)$. We say that A is *positive semidefinite (PSD)* if any of the following equivalent conditions hold:

a. $x^\intercal A x \geq 0$ for all $x \in \mathbb{R}^n$.

b. All of A's eigenvalues are nonnegative (with r of them being strictly positive), and hence $A = \sum_{i=1}^{r} \lambda_i v_i v_i^\intercal$ for $\lambda_1, \ldots, \lambda_r > 0$, and v_i's being orthonormal.

c. There exists a matrix $B \in \mathbb{R}^{n \times r}$ such that $A = BB^\intercal$.

d. There exist vectors $v_1, \ldots, v_n \in \mathbb{R}^r$ such that $A_{i,j} = \langle v_i, v_j \rangle$ for all i, j.

e. There exist jointly distributed (real-valued) random variables $X_1, \ldots, X_n$ such that $A_{i,j} = \mathbb{E}[X_i X_j]$.

f. All principal minors have nonnegative determinants.

A principal minor is a submatrix of A obtained by taking the columns and rows indexed by some subset $I \subseteq [n]$.

The different definitions may be useful in different contexts. As an example, we see that the condition in Definition 25.1(f) gives a short proof of the following claim.

Lemma 25.2. *Let $A \succeq 0$. If $A_{i,i} = 0$ then $A_{j,i} = A_{i,j} = 0$ for all j.*

We will write $A \succeq 0$ to denote that A is PSD; more generally, we write $A \succeq B$ if $A - B$ is PSD: this partial order on symmetric matrices is called the *Löwner order*.

Proof. Let $j \neq i$. The determinant of the submatrix indexed by $\{i, j\}$ is

$$A_{i,i}A_{j,j} - A_{i,j}A_{j,i}$$

is nonnegative, by assumption. Since $A_{i,j} = A_{j,i}$ by symmetry, and $A_{i,i} = 0$, we get $A_{i,j}^2 = A_{j,i}^2 \leq 0$ and we conclude $A_{i,j} = A_{j,i} = 0$. $\square$

Definition 25.3 (Frobenius Product)**.** Let $A, B \in \mathbb{R}^{n\times n}$. The Frobenius inner product $A \bullet B$, also written as $\langle A, B\rangle$ is defined as

$$\langle A, B\rangle := A \bullet B := \sum_{i,j} A_{i,j}B_{i,j} = \mathrm{Tr}(A^\intercal B).$$

We can think of this as being the usual vector inner product treating A and B as vectors of length $n \times n$. Note that by the cyclic property of the trace, $A \bullet xx^\intercal = \mathrm{Tr}(Axx^\intercal) = \mathrm{Tr}(x^\intercal Ax) = x^\intercal Ax$; we will use this fact to derive yet another of PSD matrices.

Lemma 25.4. *A is PSD if and only if $A \bullet X \geq 0$ for all $X \succeq 0$.*

Proof. Suppose $A \succeq 0$. Consider the spectral decomposition $X = \sum_i \lambda_i x_i x_i^\intercal$ where $\lambda_i \geq 0$ by Definition 25.1(b). Then

$$A \bullet X = \sum_i \lambda_i (A \bullet x_i x_i^\intercal) = \sum_i \lambda_i \, x_i^\intercal A x_i \geq 0.$$

On the other hand, if $A \not\succeq 0$, there exists v such that $v^\intercal Av < 0$, by 25.1(a). Let $X = vv^\intercal \succeq 0$. Then $A \bullet X = v^\intercal Av < 0$. $\square$

Finally, let us mention a useful fact (which can be proved, e.g., using the $x^\intercal Ax \geq 0$ characterization of PSD matrices):

Fact 25.5 (PSD cone). Given two matrices $A, B \succeq 0$, and scalars $\alpha, \beta > 0$ then $\alpha A + \beta B \succeq 0$. Hence the set of PSD matrices forms a convex cone in $\mathbb{R}^{n(n+1)/2}$.

Here $n(n+1)/2$ is the number of entries on or above the diagonal in an $n \times n$ matrix, and completely specifies a symmetric matrix.

25.2 *Semidefinite Programs*

Loosely, a semidefinite program (SDP) is the problem of optimizing a linear function over the intersection of a convex polyhedron K (given by finitely many linear constraints, say $Ax \geq b$) with the PSD cone $\mathcal{K}$. Let us give two useful packagings for semidefinite programs.

25.2.1 *As Linear Programs with a PSD Constraint*

Consider a linear program where the variables are indexed by pairs $i, j \in [n]$, i.e., a typical variable is $x_{i,j}$. Let X be the $n \times n$ dimensional matrix whose $(i, j)^{th}$ entry is $x_{i,j}$. As the objective and constraints are linear, we can write them as $C \bullet X$ and $A_k \bullet X \leq b_k$ for some (not necessarily PSD) matrices $C, A_1, \ldots, A_m$ and scalars $b_1, \ldots, b_m$. An SDP is an LP of this form with the additional constraint $X \succeq 0$:

$$\begin{aligned} \underset{X \in \mathbb{R}^{n\times n}}{\text{maximize}} \quad & C \bullet X \\ \text{subject to} \quad & A_k \bullet X \leq b_k, \qquad \forall k \in [m] \\ & X \succeq 0. \end{aligned}$$

Observe that if each of the matrices A_i and C are diagonal matrices, say with diagonals a_i and c, this SDP becomes the linear program

$$\max\{c^\intercal x \mid a_k^\intercal x \leq b_k, x \geq 0\},$$

where x denotes the diagonal of the PSD matrix X.

25.2.2 *As Vector Programs*

We can use Definition 25.1(d) to rewrite the above program as a "vector program": where the linear objective and the linear constraints are on inner products of vector variables:

$$\begin{aligned} \underset{v_1,\ldots,v_n \in \mathbb{R}^n}{\text{maximize}} \quad & \sum_{i,j} c_{ij} \langle v_i, v_j \rangle \\ \text{subject to} \quad & \sum_{i,j} a_{ij}^{(k)} \langle v_i, v_j \rangle \leq b_k, \qquad \forall k \in [m]. \end{aligned}$$

In particular, we optimize over vectors in n-dimensional space; we cannot restrict the dimension of these vectors, much like we cannot restrict the rank of the matrices X in the previous representation.

25.2.3 *Examples of SDPs*

Let A a symmetric $n \times n$ real matrix. Here is an SDP to compute the maximum eigenvalue of A:

$$\begin{aligned} \underset{X \in \mathbb{R}^{n\times n}}{\text{maximize}} \quad & A \bullet X \\ \text{subject to} \quad & I \bullet X = 1 \\ & X \succeq 0 \end{aligned} \tag{25.1}$$

Lemma 25.6. *SDP (25.1) computes the maximum eigenvalue of A.*

Proof. Let X maximize SDP (25.1) (this exists as the objective is continuous and the feasible set is compact). Consider the spectral decomposition $X = \sum_{i=1}^n \lambda_i x_i x_i^\intercal$ where $\lambda_i \geq 0$ and $\|x_i\|_2 = 1$. The trace constraint $I \bullet X = 1$ implies $\sum_i \lambda_i = 1$. Thus the objective value $A \bullet X = \sum_i \lambda_i x_i^\intercal A x_i$ is a convex combination of $x_i^\intercal A x_i$. Hence without loss of generality, we can put all the weight into one of these terms, in which case $X = yy^\intercal$ is a rank-one matrix with $\|y\|_2 = 1$. By the Courant-Fischer theorem, $OPT \leq \max_{\|y\|_2=1} y^\intercal A y = \lambda_{\max}$.

On the other hand, letting v be a unit eigenvector of A corresponding to $\lambda_{\max}$, we have that $OPT \geq A \bullet vv^\intercal = v^\intercal A v = \lambda_{\max}$. $\square$

Here is another SDP for the same problem:

$$\begin{aligned} \underset{t}{\text{minimize}} \quad & t \\ \text{subject to} \quad & tI - A \succeq 0. \end{aligned} \tag{25.2}$$

In fact, it turns out that this SDP is dual to the one in (25.1). Weak duality still holds for this case, but strong duality does not hold in general for SDPs. Indeed, there could be a duality gap for some cases, where both the primal and dual are finite, but the optimal solutions are not equal to each other. However, under some mild regularity conditions (e.g., the Slater conditions) we can show strong duality. More about SDP duality here.

Lemma 25.7. *SDP (25.2) computes the maximum eigenvalue of A.*

Proof. The matrix $tI - A$ has eigenvalues $t - \lambda_i$. And hence the constraint $tI - A \succeq 0$ is equivalent to the constraint $t - \lambda \geq 0$ for all its eigenvalues λ. In other words, $t \geq \lambda_{\max}$, and thus $OPT = \lambda_{\max}$. $\square$

25.3 *SDPs in Approximation Algorithms*

We now consider designing approximation algorithms using SDPs. Recall that given a matrix A, we can check if it is PSD in (strongly) polynomial time, by performing its eigendecomposition. Moreover, if A is not PSD, we can return a hyperplane separating A from the PSD cone. Thus using the ellipsoid method, we can approximate SDPs when OPT is appropriately bounded. Informally,

We know that there is an optimal LP solution where the numbers are singly exponential, and hence can be written using a polynomial number of bits. But this is not true in SDPs, in fact, OPT in an SDP may be as large (or small) as doubly exponential in the size of the SDP. (See Section 2.6 of the Matoušek and Gärtner.)

Theorem 25.8 (Informal Theorem)**.** *Assuming that the radius of the feasible set is at most* $\exp(\text{poly}(\langle SDP \rangle))$*, the ellipsoid algorithm can weakly solve SDP in time* $\text{poly}(\langle SDP \rangle, \log(1/\varepsilon))$ *up to an additive error of ε.*

For a formal statement, see Theorem 2.6.1 of Matoušek and Gärtner. However, we will ignore these technical issues in the remainder of this chapter, and instead suppose that we can solve our SDPs exactly.

25.4 *The* MaxCut *Problem and Hyperplane Rounding*

Given a graph $G = (V, E)$, the MaxCut problem asks us to find a partition of the vertices $(S, V \setminus S)$ maximizing the number of edges crossing the partition. This problem is **NP**-complete. In fact assuming **P** $\neq$ **NP**, a result of Johan Håstad shows that we cannot approximate MaxCut better than $17/16 - \varepsilon$ for any $\varepsilon > 0$.

25.4.1 *Greedy and Randomized Algorithms*

We begin by considering a greedy algorithm: process the vertices $v_1, \ldots, v_n$ in some order, and place each vertex v_i in the part of the bipartition that maximizes the number of edges cut so far (breaking ties arbitrarily).

Lemma 25.9. *The greedy algorithm cuts at least $|E|/2$-many edges.*

Proof. Let δ_i be the number of edges from vertex i to vertices $j < i$: then the greedy algorithm cuts at least $\sum_i \delta_i/2 = |E|/2$ edges. $\square$

This result shows two things: (a) every graph has a bipartition that cuts half the edges of the graph, so $\mathrm{Opt} \geq |E|/2$. Moreover, (b) that since $\mathrm{Opt} \leq |E|$ on any graph, this means that $\mathrm{Alg} \geq |E|/2 \geq \mathrm{Opt}/2$.

We cannot hope to prove a better result than Lemma 25.9 in terms of $|E|$, since the complete graph K_n has $\binom{n}{2} \approx n^2/2$ edges and any partition can cut at most $n^2/4$ of them.

Here's a simple randomized algorithm: place each vertex in either S or in $\bar{S}$ independently and uniformly at random. Since each edge is cut with probability 1/2, the expected number of cut edges is $|E|/2$. Moreover, by the probabilistic method $\mathrm{Opt} \geq |E|/2$.

25.4.2 *Relax-and-Round using LPs*

A natural direction would be to write an ILP formulation for MaxCut and to relax it: this approach does not give us anything beyond a factor of 1/2, say.

25.4.3 *A Semidefinite Relaxation*

We now see a well-known example of an SDP-based approximation algorithm due to Michel Goemans and David Williamson. Again, we will use the relax-and-round framework from the previous chapter. The difference is that we write a quadratic program to model the problem exactly, and then relax it to get an SDP.

Indeed, observe that the MaxCut problem can be written as the following quadratic program.

$$\begin{aligned} \underset{x_1,\dots,x_n \in \mathbb{R}}{\text{maximize}} \quad & \sum_{(i,j)\in E} \frac{(x_i - x_j)^2}{4} \\ \text{subject to} \quad & x_i^2 = 1 \qquad \forall i. \end{aligned} \tag{25.3}$$

Since each x_i is real-valued, and $x_i^2 = 1$, each variable must be assigned one of two labels $\{-1, +1\}$. Since each term in the objective contributes 1 for an edge connecting two vertices in different partitions, and 0 otherwise, this IP precisely captures MaxCut.

We now relax this program by replacing the variables x_i with vector variables $v_i \in \mathbb{R}^n$, where $\|v_i\|^2 = 1$.

$$\begin{aligned} \underset{v_1,\dots,v_n \in \mathbb{R}^n}{\text{maximize}} \quad & \sum_{(i,j)\in E} \frac{\|v_i - v_j\|^2}{4} \\ \text{subject to} \quad & \|v_i\|^2 = 1 \qquad \forall i. \end{aligned} \tag{25.4}$$

Noting that $\|v_i - v_j\|^2 = \|v_i\|^2 + \|v_j\|^2 - 2\langle v_i, v_j\rangle = 2 - 2\langle v_i, v_j\rangle$, we rewrite this vector program as

The SDP relaxation for the MaxCut problem was first introduced by Svata Poljak and Franz Rendl.

$$\begin{aligned} \underset{v_1,\dots,v_n \in \mathbb{R}^n}{\text{maximize}} \quad & \sum_{(i,j)\in E} \frac{1 - \langle v_i, v_j \rangle}{2} \\ \text{subject to} \quad & \langle v_i, v_i \rangle = 1 \qquad \forall i. \end{aligned} \tag{25.5}$$

This is a relaxation of the original quadratic program, because we can model any $\{-1, +1\}$-valued solution using vectors, say by a corresponding $\{-e_1, +e_1\}$-valued solution. Since this is a maximization problem, the SDP value is now at least the optimal value of the quadratic program.

25.4.4 *The Hyperplane Rounding Technique*

In order to round this vector solution $\{v_i\}$ to the MaxCut SDP into an integer scalar solution to MaxCut, we use the remarkably simple method of *hyperplane rounding*. The idea is this: a term in the SDP objective incurs a tiny cost close to zero when v_i, v_j are very close to each other, and almost unit cost when v_i, v_j point in nearly opposite directions. So we would like to map close vectors to the same value.

To do this, we randomly sample a hyperplane through the origin and partition the vectors according to the side on which they land. Formally, this corresponds to picking a vector $g \in \mathbb{R}^n$ according to the standard n-dimensional Gaussian distribution, and setting

$$S := \{i \mid \langle v_i, g \rangle \geq 0\}.$$

We now argue that this procedure gives us a good cut in expectation; this procedure can be repeated to get an algorithm that succeeds with high probability.

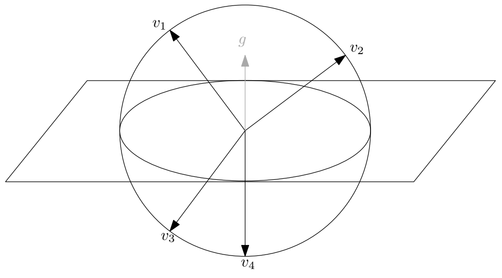

Figure 25.1: A geometric picture of Goemans-Williamson randomized rounding

Theorem 25.10. *The partition produced by the hyperplane rounding algorithm cuts at least $\alpha_{GW} \cdot$ SDP edges in expectation, where $\alpha_{GW} := 0.87856$.*

Proof. By linearity of expectation, it suffices to bound the probability of an edge (i, j) being cut. Let

$$\theta_{ij} := \cos^{-1}(\langle v_i, v_j \rangle)$$

be the angle between the unit vectors v_i and v_j. Now consider the 2-dimensional plane P containing v_i, v_j and the origin, and let $\tilde{g}$ be the projection of the Gaussian vector g onto this plane. Observe that the edge (i, j) is cut precisely when the hyperplane defined by g separates v_i, v_j. This is precisely when the vector perpendicular to $\tilde{g}$ in the plane P lands between v_i and v_j. As the projection onto a subspace of the standard Gaussian is again a standard Guassian (by spherical symmetry),

$$\Pr[(i, j) \text{ cut}] = \frac{2\theta_{ij}}{2\pi} = \frac{\theta_{ij}}{\pi}.$$

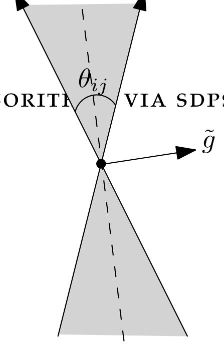

Figure 25.2: Angle between two vectors. We cut edge (i, j) when the vector perpendicular to $\tilde{g}$ lands in the grey area.

Since the SDP gets a contribution of

$$\frac{1 - \langle v_i, v_j \rangle}{2} = \frac{1 - \cos(\theta_{i,j})}{2}$$

for this edge, it suffices to show that

$$\frac{\theta}{\pi} \geq \alpha \frac{1 - \cos\theta}{2}.$$

Indeed, we can show (either by plotting, or analytically) that $\alpha = 0.87856\ldots$ suffices for the above inequality, and hence

$$\mathbb{E}[\text{\# edges cut}] = \sum_{(i,j)\in E} \theta_{ij}/\pi \geq \alpha \sum_{(i,j)\in E} \frac{1 - \cos(\theta_{ij})}{2} = \alpha \ \text{SDP}.$$

This proves the theorem. □

Corollary 25.11. *For any $\varepsilon > 0$, repeating the hyperplane rounding algorithm $O(1/\varepsilon \log 1/\delta)$ times and returning the best solution ensures that we output a cut of value at least $(.87856 - \varepsilon)$ Opt with probability $1 - \delta$.*

We leave this proof as an exerise in using Markov's inequality: note that we want to show that the algorithm returns something not too far *below* the expecation, which seems to go the wrong way, and hence requires a moment's thought.

The above algorithm is randomized and the result only holds in expectation. However, it is possible to derandomize this result to obtain a polynomial-time deterministic algorithm with the same approximation ratio.

25.4.5 *Subsequent Work and Connections*

Can we get a better approximation factor, perhaps using a more sophisticated SDP? An influential result of Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell says that a constant-better-than-α_{GW}-approximation would refute the Unique Games Conjecture.

Also, one can ask if similar rounding procedures exist for an linear-programming relaxation as opposed to the SDP relaxation here. Unfortunately the answer is again no: a result of Siu-On Chan, James Lee, Prasad Raghavendra, and David Steurer shows that no polynomial-sized LP relaxation of MaxCut can obtain a non-trivial approximation factor, that is, any polynomial sized LP of MaxCut has an integrality gap of 1/2.

25.5 *Coloring 3-Colorable Graphs*

Suppose we are given a graph $G = (V, E)$ and a promise that there is some 3-coloring of G. What is the minimum k such that we can find a k-coloring of G in polynomial time? It is well-known that 2-coloring a graph can be done in linear time, but 3-coloring a graph is **NP**-complete. Hence, even given a 3-colorable graph, it is **NP**-hard to color it using 3 colors. (In fact, a result of Venkat Guruswami and Sanjeev Khanna shows that it is **NP**-hard to color it using even 4 colors.) But what if we ask to color a 3-colorable graph using 5 colors? $O(\log n)$ colors? $O(n^\alpha)$ colors, for some fixed constant α? We will see an easy algorithm to achieve an $O(\sqrt{n})$-coloring, and then will use semidefinite programming to improve this to an $\widetilde{O}(n^{\log_6(2)})$ coloring. Before we describe these, let us recall the easy part of Brooks' theorem.

The harder part is to show that in fact Δ colors suffice unless the graph is either a complete graph, or an odd-length cycle.

Lemma 25.12. *Let Δ be the maximum degree of a graph G, then we can find a $(\Delta + 1)$-coloring of G in linear time.*

Proof. Pick any vertex v, recursively color the remaining graph, and then assign v a color not among the colors of its Δ neighbors. □

We will now describe an algorithm that colors a 3-colorable graph G with $O(\sqrt{n})$ colors, originally due to Avi Wigderson: while there exists a vertex with degree at least $\sqrt{n}$, color it using a fresh color. Moreover, its neighborhood must be 2-colorable, so use two fresh colors to do so. This takes care of $\sqrt{n}$ vertices using 3 colors. Remove these, and repeat. Finally, use Lemma 25.12 to color the remaining vertices using $\sqrt{n}$ colors. This proves the following result.

Lemma 25.13. *There is an algorithm to color a 3-colorable graph with $O(\sqrt{n})$ colors.*

25.5.1 An Algorithm using SDPs

Let's consider an algorithm that uses SDPs to color a 3-colorable graph with maximum degree Δ using $\widetilde{O}(\Delta^{\log_3 2}) \approx \widetilde{O}(\Delta^{0.63})$ colors. In general Δ could be as large as n, so this could be worse than the algorithm in Lemma 25.13, but we will be able to combine the ideas together to get a better result.

For some parameter $\lambda \in \mathbb{R}$, consider the following feasibility SDP (where we are not optimizing any objective):

$$\begin{aligned} \text{find} \quad & v_1, \dots, v_n \in \mathbb{R}^n & \\ \text{subject to} \quad & \langle v_i, v_j \rangle \le \lambda & \forall (i,j) \in E \\ & \langle v_i, v_i \rangle = 1 & \forall i \in V. \end{aligned} \tag{25.6}$$

Why is this SDP relevant to our problem? The goal is to have vectors clustered together in groups, such that each cluster represents a color. Intuitively, we want to have vectors of adjacent vertices to be far apart, so we want their inner product to be close to -1 (recall we are dealing with unit vectors, due to the last constraint) and vectors of the same color to be close together.

Lemma 25.14. *For 3-colorable graphs, SDP (25.6) is feasible with $\lambda = -1/2$.*

Proof. Consider the vector placement shown in the figure to the right.

If the graph is 3-colorable, we can assign all vertices with color 1 the red vector, all vertices with color 2 the blue vector and all vertices with color 3 the green vector. Now for every edge $(i, j) \in E$, we have that

$$\langle v_i, v_j \rangle = \cos\left(\frac{2\pi}{3}\right) = -1/2. \qquad \square$$

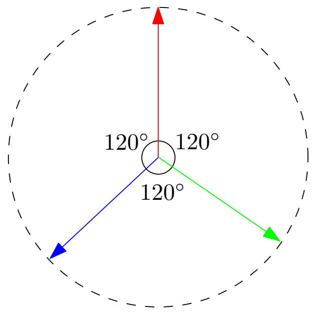

Figure 25.3: Optimal distribution of vectors for 3-coloring graph

At first sight, it may seem like we are done: if we solve the above SDP with $\lambda = -1/2$, don't all three vectors look like the figure above? No, that would only hold if all of them were to be co-planar. And in n-dimensions we can have an exponential number of cones of angle $\frac{2\pi}{3}$, like in the next figure, so we cannot cluster vectors as easily as in the above example.

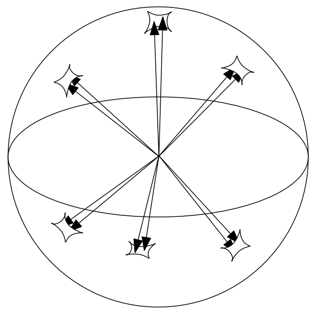
Figure 25.4: Dimensionality problem of $2\pi/3$ far vectors

To solve this issue, we apply a hyperplane rounding technique similar to that from the MaxCut algorithm. Indeed, for some parameter t we will pick later, pick t random hyperplanes. Formally, we pick $g_i \in \mathbb{R}^n$ from a standard n-dimensional Gaussian distribution, for $i \in [t]$. Each of these defines a normal hyperplane, and these split the $\mathbb{R}^n$ unit sphere into 2^t regions (except if two of them point in the same direction, which has zero probability). Now, each vectors $\{v_i\}$ that lie in the same region can be considered "close" to each other, and we can try to assign them a unique color. Formally, this means that if v_i and v_j are such that

$$\text{sign}(\langle v_i, g_k \rangle) = \text{sign}(\langle v_j, g_k \rangle)$$

for all $k \in [t]$, then i and j are given the same color. Each region is given a different color, of course.

However, this may color some neighbors with the same color, so we use the *method of alterations*: while there exists an edge between vertices of the same color, we uncolor both endpoints. When this uncoloring stops, we remove the still-colored vertices from the graph, and then repeat the same procedure on the remaining graph, until we color every vertex. Note that since we use t hyperplanes, we add at most 2^t new colors per iteration. The goal is to now show that (a) the number of interations is small, and (b) the value of 2^t is also small.

Lemma 25.15. *If half of the vertices are colored in a single iteration in expectation, then the expected number of iterations to color the whole graph is $O(\log n)$.*

Proof. Since the expected number of uncolored vertices is at most half, Markov's inequality says that more than $3/4$ of the vertices are uncolored in a single iteration, with probability at most $2/3$. In other words, at least $1/4$ of the vertices are colored with probability $1/3$. Hence, the number of iterations to color the whole graph is dominated by the number of flips of a coin of bias $1/3$ to get $\log_4 n$ heads. This is $4\log_4 n$, which proves the result. □

Lemma 25.16. *The expected number of vertices that remain uncolored after a single iteration is at most $n\Delta\,(1/3)^t$.*

Proof. Fix an edge ij: for a single random hyperplane, the probability that v_i, v_j are not separated by it is

$$\frac{\pi - \theta_{ij}}{\pi} \leq \frac{1}{3},$$

using that $\theta_{ij} \geq \frac{2\pi}{3}$ which follows from the constraint in the SDP. Now if i is uncolored because of j, then v_i, v_j have the same color, which happens when all t hyperplanes fail to separate the two. By independence, this happens with probability at most $(1/3)^t$. Finally,

$$\begin{aligned}\mathbb{E}[\text{remaining}] &= \sum_{i\in V} \Pr[i \text{ uncolored}] \\ &\leq \sum_{i\in V} \sum_{(i,j)\in E} \Pr[i \text{ uncolored because of } j]. \qquad (25.7)\end{aligned}$$

There are n vertices, and each vertex has degree at most Δ, which proves the result. □

Lemma 25.17. *There is an algorithm that colors a 3-colorable graph with maximum degree Δ with $O(\Delta^{\log_3 2} \cdot \log n)$ colors in expectation.*

Proof. Setting $t = \log_3(2\Delta)$ in Lemma 25.16, the expected number of uncolored vertices in any iteration is

$$n \cdot \Delta \cdot (1/3)^t \leq n/2. \qquad (25.8)$$

Now Lemma 25.15 says we perform $O(\log n)$ iterations in expectation. Since we use most $2^{\log_3(2\Delta)} = (2\Delta)^{\log_3 2}$ colors in each iteration, we get the result. □

25.5.2 *Improving the Algorithms Further*

The expected number of colors used by the above algorithm is $\widetilde{O}(n^{\log_3 2}) \approx \widetilde{O}(n^{0.63})$, which is worse than our initial $O(\sqrt{n})$ algorithm. However we can combine the ideas together to get a better result:

Theorem 25.18. *There is an algorithm that colors a 3-colorable graph with $\widetilde{O}(n^{\log_6(2)})$ colors.*

Proof. For some value σ, repeatedly remove vertices with degree greater than σ and color them and their neighbors with 3 new colors, as in Lemma 25.13. This requires at most $3n/\sigma$ colors overall, and leaves us with a graph having maximum degree σ. Now use Lemma 25.17 to color the remaining graph with $O(\sigma^{\log_3 2} \cdot \log n)$ colors. Picking σ to be $n^{\log_6 3}$ to balance these terms, we get a procedure that uses $\widetilde{O}(n^{\log_6 2}) \approx \widetilde{O}(n^{0.38})$ colors. □

25.5.3 *Final notes on coloring 3-colorable graphs*

This result us due to David Karger, Rajeev Motwani, and Madhu Sudan. They gave a better rounding algorithm that uses spherical caps instead of hyperplanes to achieve $\widetilde{O}(n^{1/4})$ colors. This result was then improved over a sequence of papers: the current best result by Ken-Ichi Kawarabayashi and Mikkel Thorup uses $O(n^{0.199})$ colors. It remains an outstanding open problem to either get a better algorithm, or to show hardness results, even under stronger complexity-theoretic hypotheses.