21
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 MAXIMUM 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.

21.1 Positive Semidefinite Matrices

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

Definition 21.1 (Positive Semidefinite Matrices)**.** Let $A \in \mathbb{R}^{n \times n}$ be a real-valued symmetric matrix and let $r = \text{rank}(A)$. 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 21.1(f) gives a short proof of the following claim.

Lemma 21.2. *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 $A_{i,j} = A_{j,i} = 0$. □

Definition 21.3 (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 21.4. *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 21.1(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 21.1(a). Let $X = vv^\intercal \succeq 0$. Then $A \bullet X = v^\intercal Av < 0$. □

Finally, let us mention a useful fact:

Fact 21.5 (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.

21.2 *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.

21.2.1 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 $X \succeq 0$:

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

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.

21.2.2 As Vector Programs

Consider a linear program where instead of linear constraints (respectively, objective) on variables, we have linear constraints (objective) on the inner products of vector variables, i.e. a program of the following form:

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

In particular, note that we optimize over vectors in n-dimensional space. By definition 4, we see that the two views are the same.

21.2.3 Examples of SDPs

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

$$\begin{aligned} \underset{X \in \mathbb{R}^{n \times n}}{\text{maximize}} \quad & A \bullet X \\ \text{subject to} \quad & I \bullet X = 1 \\ & X \succeq 0 \end{aligned} \tag{21.1}$$

Lemma 21.6. *SDP (21.1) computes the maximum eigenvalue of A.*

Proof. Let X maximize SDP 21.1 (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 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, $X = yy^\intercal$ is rank one with $\|y\|_2 = 1$. Thus, by Courant-Fischer, $OPT \leq \max_{\|y\|_2 = 1} y^\intercal A y = \lambda_{\max}$.

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

Here is another SDP for the same problem:

$$\begin{aligned} \underset{t}{\text{minimize}} \quad & t \\ \text{subject to} \quad & tI - A \succeq 0. \end{aligned} \tag{21.2}$$

In fact, it turns out that this SDP is dual to the one in (21.1). 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.

Lemma 21.7. *SDP (21.2) computes the maximum eigenvalue of A.*

Proof. The constraint $tI - A \succeq 0$ is equivalent to the constraint $t - \lambda \geq 0$ for all eigenvalues λ, i.e., $t \geq \lambda_{\max}$. Thus $OPT = \lambda_{\max}$. □

21.3 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. In fact, if A is not PSD, we can return a hyperplane separating A from the PSD cone. Thus using the ellipsoid method (see Lecture 21), we can weakly 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 21.8 (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* $\text{poly}(\langle SDP \rangle, \log(\frac{1}{\varepsilon}))$ *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 the lecture and instead suppose that we can solve our SDPs exactly. More here.

21.4 The Max-Cut Problem and Hyperplane Rounding

Given a graph $G = (V, E)$, the Max-Cut 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 $\mathbf{P} \neq \mathbf{NP}$, a result of Johan Håstad shows that we cannot approximate Max-Cut better than $17/16 - \varepsilon$ for any $\varepsilon > 0$.

We begin by considering the greedy algorithm.

Lemma 21.9. *The greedy algorithm cuts at least $|E|/2$-many edges.*

Proof. Consider the following procedure: process the vertices $v_1, \ldots, v_n$ in order and place each vertex in the partition that maximizes the number of edges cut so far. 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. □

In particular, we see that OPT is suitably bounded and we can hope to "solve" an SDP relaxation.

Corollary 21.10. *Let OPT be the optimal value of Max-Cut. Then $|E|/2 \leq OPT \leq |E|$.*

We now see a famous example of an SDP-based approximation algorithm due to Goemans and Williamson. Begin by noting that the Max-Cut problem can be written as the following integer program.

$$\begin{aligned} \underset{x_1,\ldots,x_n \in \mathbb{R}}{\text{maximize}} \quad & \sum_{(i,j)\in E} \frac{(x_i - x_j)^2}{4} \\ \text{subject to} \quad & x_i \in \{-1, +1\},\ \forall i. \end{aligned} \tag{21.3}$$

In this program, each element must be assigned one of two labels $\{-1, +1\}$. The objective value gained from an edge connecting to vertices in different partitions is 1 and is 0 otherwise, thus we see that this IP captures Max-Cut. We will relax this program by replacing the variables x_i with vector variables $v_i \in \mathbb{R}^n$.

$$\begin{aligned} \underset{v_1,\ldots,v_n \in \mathbb{R}^n}{\text{maximize}} \quad & \sum_{(i,j)\in E} \frac{\|v_i - v_j\|^2}{4} \\ \text{subject to} \quad & \|v_i\| = 1,\ \forall i. \end{aligned} \tag{21.4}$$

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

$$\begin{aligned} \underset{v_1,\ldots,v_n \in \mathbb{R}^n}{\text{maximize}} \quad & \sum_{(i,j)\in E} \frac{1 - \langle v_i, v_j\rangle}{2} \\ \text{subject to} \quad & \langle v_i, v_i\rangle = 1,\ \forall i. \end{aligned} \tag{21.5}$$

The SDP relaxation for the Max-Cut problem was first introduced by Svata Poljak and Franz Rendl. It is clear that this is a relaxation of the original integer program as given a $\{-1, +1\}$ valued solution, we can consider the corresponding $\{-e_1, +e_1\}$-valued solution where e_1 is the first standard basis vector.

Our goal is now to find a solution to Max-Cut given a solution to the Max-Cut SDP. To do so, we are going to use a method known as hyperplane rounding.

Assume we have a solution $\{v_i\}$ to the Max-Cut SDP. We want to find some partition of the vectors $\{v_i\}$ placing vectors that are close together in the same set and vectors that are far from each other in distinct sets. To do this, we will 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 Gaussian and setting $S = \{i | \langle v_i, g\rangle \geq 0\}$ (meaning v_i is on the "nonnegative" side of the hyperplane).

Alternatively, we could have considered the randomized algorithm placing each vertex in $S, \bar{S}$ uniformly independently at random. Then the expected number of cut edges is $|E|/2$ whence by the probabilistic method $OPT \geq |E|/2$. We see that both the greedy and randomized algorithms are $1/2$-approximations.

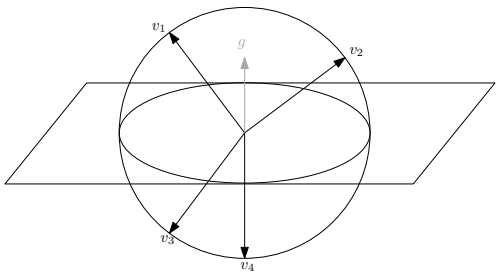

Figure 21.1: A geometric picture of Goemans-Williamson randomized rounding

We will argue that this procedure gives us a good cut in expectation then repeat the procedure to achieve an algorithm succeeding with high probability.

Theorem 21.11. *For any $\varepsilon > 0$, the Goemans-Williamson randomized rounding algorithm returns a partition cutting at least $\alpha_{GW} := (0.87856 - \varepsilon) \cdot SDPOpt$-many edges with high probability.*

Proof. Let's first compute the expected number of cut edges:

$$\mathbb{E}[\#\text{ edges cut}] = \sum_{(i,j)\in E} \Pr[(i,j) \text{ is cut}].$$

For some edge $(i,j) \in E$, let

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

be the angle between vectors v_i and v_j. We will work in the plane containing v_i, v_j and the origin. Let $\tilde{g}$ be the projection of g onto this plane. Note that the cut defined by g and $\tilde{g}$ agree on this plane. The probability of edge (i,j) being cut is the probability that the vector perpendicular to $\tilde{g}$ lands between v_i and v_j. As the projection onto a subspace of the standard Gaussian is a standard Guassian, this probability is exactly θ_{ij}/π. The following figure illustrates the proof. Notice that we cut edge (i,j) when the vector perpendicular to $\tilde{g}$ lands in the grey area, which accounts for $2\theta_{ij}/2\pi$-fraction or θ_{ij}/π-fraction of the directions.

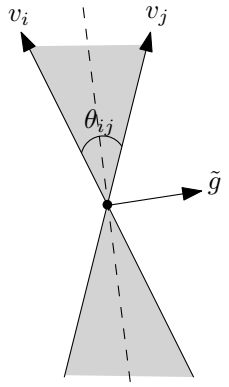

Figure 21.2: Angle between two vectors

Recall

$$SDPOpt := \sum_{(i,j)\in E} \frac{1 - \cos(\theta_{i,j})}{2}.$$

Observe that using the probability computed above we have

$$\mathbb{E}[\#\text{ edges cut}] = \sum_{(i,j)\in E} \theta_{ij}/\pi.$$

Thus supposing the existence of an α such that

$$\theta/\pi \geq \alpha \frac{1 - \cos(\theta)}{2}$$

for all $\theta \in [0, \pi]$, we have that

$$\mathbb{E}[\#\text{ edges cut}] \geq \alpha \cdot \text{SDPOpt},$$

where SDPOpt is the solution to the Max-Cut SDP. Such α exists and has value $\alpha_{GW} = 0.87856$. Since $\mathbb{E}[\#\text{ edges cut}] \geq \alpha_{GW} \cdot \text{SDPOpt}$, by Markov's inequality we obtain an $(\alpha_{GW} - \varepsilon)$-approximation to the Max-Cut-SDP for any constant ε. □

Finally noting that $SDPOpt \geq Opt$, we get the following result.

Corollary 21.12. *For any $\varepsilon > 0$, there is a $(.87856 - \varepsilon)$-factor approximation algorithm for Max-Cut*

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

As a follow-up to this analysis, one can ask some relevant questions. First, can we get a better approximation factor? A result of Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell says that a 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 Max-Cut can obtain a non trivial approximation factor, that is, any polynomial sized LP of Max-Cut has an integrality gap of $1/2$.

21.5 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 3-coloring a graph is **NP**-complete, but what if we want to color it with $O(n^\alpha)$, for some fixed constant α? We will see that is easy to achieve a $O(\sqrt{n})$ coloring and then will use semidefinite programming to improve this to a $\widetilde{O}(n^{\log_6(2)})$ coloring.

Lemma 21.13. *Let G be a 2-colorable graph, then we can find a 2-coloring of G in linear time.*

Proof. Being 2-colorable is equivalent to being bipartite. Thus, we can pick an arbitrary node and color it with color 1. Then, we color its neighbors with color 2, its neighbors' neighbors with color 1 and so on. We can do this using a depth-first search in the graph, so it runs in linear time. □

Lemma 21.14. *Let Δ be the maximum degree of a graph G, then we can find a $\Delta + 1$ coloring of G in linear time.*

Proof. We have the following iterative algorithm: pick some uncolored vertex and color it with a color different from all of its neighbors. Since every vertex has at most Δ colors, we never run out of colors to assign to a vertex. It is easy to see this can be done in linear time. □

We will now describe an algorithm that colors a 3-colorable graph G with $O(\sqrt{n})$ colors, originally due to Avi Wigderson: while the maximum degree of G is greater than $\sqrt{n}$, pick some vertex v with degree larger than $\sqrt{n}$. Color it with a new color. Consider the subgraph induced by the neighbors of v. Since G is 3-colorable and we have to color v with some color, the subgraph of v's neighbors is 2-colorable, so we can find a 2-coloring in linear time by Lemma 21.13 using two new colors. Now remove v and its neighbors from the graph. We removed at least $\sqrt{n}$ nodes and used at most 3 new colors, so after $\sqrt{n}$ rounds we will have used at most $3\sqrt{n}$ colors and the graph will have maximum degree $\sqrt{n}$. Finally, we color the remaining graph with at most $\sqrt{n}+1$ colors using Lemma 21.14. We used a total of $4\sqrt{n}+1$ colors, so this fits our original goal. We conclude,

Lemma 21.15. *There is an algorithm that colors a 3-colorable graph with with $O(\sqrt{n})$ colors.*

21.5.1 A better solution using SDPs

We will now use semidefinite programming to get an improved $\widetilde{O}(n^{\log_6(2)})$ coloring, using a result by David Karger, Rajeev Motwani, and Madhu Sudan. The core method for this result is an algorithm that colors a 3-colorable graph with maximum degree Δ using $\widetilde{O}(\Delta^{\log_3(2)})$ colors, which we will describe first.

For some parameter λ which we will pick later, consider the following SDP:

$$\begin{array}{lll} \text{find} & v_1, \ldots, v_n \in \mathbb{R}^n & \\ \text{subject to} & \langle v_i, v_j \rangle \leq \lambda & \forall (i,j) \in E \\ & \langle v_i, v_i \rangle = 1 & \forall i \in V \end{array} \tag{21.6}$$

Notice this is a feasibility SDP, we are not optimizing any objective. 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 21.16. *For 3-colorable graphs, SDP 21.6 is feasible with $\lambda = -1/2$.*

Proof. Consider vector placement shown in the following figure:

It is clear that if the graph is 3-colorable, we can have all vertices with color 1 align with the red vector, all vertices with color 2 align with the blue vector and all vertices with color 3 align with the green vector. For every edge $(i,j) \in E$, we have that $\langle v_i, v_j \rangle = \cos\left(\frac{2\pi}{3}\right) = -1/2$. □

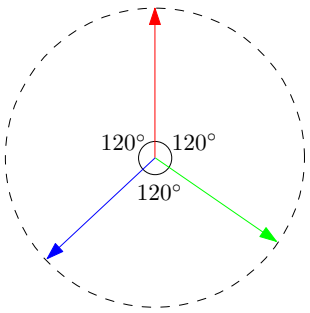

Figure 21.3: Optimal distribution of vectors for 3-coloring graph

It may seem like we are done, since if we solve the above SDP with $\lambda = -1/2$ we could expect it to look like the figure above. Unfortunately, in n-dimensions we can have an exponential number of cones of angle $\frac{2\pi}{3}$, like the figure below shows, so we cannot cluster vectors as easily as in the above example.

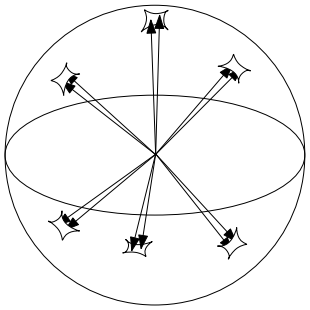

Figure 21.4: Dimensionality problem of $2\pi/3$ far vectors

To solve this issue, we are going to apply a technique similar to how we rounded Max-Cut SDP solutions, namely using random hyperplanes to separate vectors. Consider the following algorithm: for some parameter t we will pick later, pick t random hyperplanes. Formally, we pick $g_i \in \mathbb{R}^n$ from a standard Gaussian for $i \in [t]$. Observe that these split the $\mathbb{R}^n$ unit sphere into at most 2^t regions. Now, for every set of vectors on the same region induced by these random vectors, we 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 will have the same color, otherwise they will have different colors. This may color some adjacent vertices with the same color, so to fix this while there is any edge between vertices of the same color, uncolor both endpoints.

At the end of this procedure, we will have some colored vertices which we can remove 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 round. The hope with such an approach is that in each round we remove enough vertices from the graph such that the number of required iterations is small.

Lemma 21.17. *If, in expectation, half of the vertices are colored in a single round, then the expected number of rounds to color the whole graph is $O(\log n)$.*

Proof. Let T be the random variable which is the number of rounds needed to color the whole graph.

By Markov's inequality, at least $1/4$ of the vertices are colored in a single round with probability $1/3$. Let X_i be the random variable that is 1 if at least $1/4$ of the vertices are colored in the ith round and 0 otherwise. Thus $X_i = 1$ with probability at least $1/3$. Let T' be the random variable which is the number of rounds needed for $\sum X_i \geq \log_4(n)$, then $T \leq T'$.

Note $\mathbb{E}[T']$ is the expected number of rounds to get the first success, plus the expected number of rounds to the second success etc. Note also that the expected number of rounds to get a single success is a geometric random variable with mean at most 3. Combining, $\mathbb{E}[T] \leq \mathbb{E}[T'] \leq 3\log_4(n) = O(\log n)$. □

The previous claim means that if the expected fraction of vertices removed per iteration is $1/2$, then we can color the whole graph with

a blow up of $O(\log n)$. We compute the expected number of vertices remaining after a given iteration.

$$\begin{aligned}\mathbb{E}[\text{remaining}] &= \sum_{i \in V} \Pr[i \text{ uncolored}] \\ &\leq \sum_{i \in V} \sum_{(i,j) \in E} \Pr[i \text{ uncolored because of } j]. \end{aligned} \tag{21.7}$$

As we saw in the Max-Cut section, we know that the probability i has the same color as j is exactly the probability that the projection of g_k onto the plane containing v_i, v_j and the origin is in the dual cone to v_i and v_j for all $k \in [t]$. For a given hyperplane, the probability that v_i, v_j land on the same side is exactly $(\pi - \theta_{ij})/\pi$, which is at most $1/3$ (recall we built our SDP such that $\theta_{ij} \geq \frac{2\pi}{3}$ for edges (i,j)). So plugging this into the above,

Contrary to the Max-Cut case, we are looking at the case where none of the hyperplanes separate v_i and v_j, which is why we consider g_k instead of its perpendicular.

$$(\text{eq. } 21.7) \leq n \cdot \Delta \cdot (1/3)^t. \tag{21.8}$$

Since we want the expected fraction of vertices removed to be at least $1/2$, i.e. we want $\mathbb{E}[\text{remaining}]$ to be at most $1/2$. We compute the necessary value of t,

$$n \cdot \Delta \cdot (1/3)^t \leq n/2 \Rightarrow t \geq \log_3(2\Delta) \tag{21.9}$$

We will take $t = \log_3(2\Delta)$. Thus we use at most $2^{\log_3(2\Delta)} = (2\Delta)^{\log_3(2)}$ colors in each round. Combining this with the above claim, we get the following lemma.

Lemma 21.18. *There is an algorithm that colors a 3-colorable graph with maximum degree Δ with $\widetilde{O}(\Delta^{\log_3(2)})$ colors.*

This gives us an algorithm that uses at most $\widetilde{O}(n^{\log_3(2)}) \approx \widetilde{O}(n^{0.63})$, which is even worse than our initial $O(\sqrt{n})$ algorithm. However we can combine the ideas from that algorithm with the ideas of this new algorithm as follows.

Theorem 21.19. *There is an algorithm that colors a 3-colorable graph with with $\widetilde{O}(n^{\log_6(2)})$ colors.*

Proof. Let σ be a parameter we will set later. Remove all vertices with degree greater than σ and color them and their neighbors with 3 new colors, as in the $O(\sqrt{n})$ algorithm (Lemma 21.15). This requires at most $3n/\sigma$ new colors and the resulting graph has maximum degree σ. Now we can use Lemma 21.18 to color the remaining graph with $\sigma^{\log_3(2)}$ colors.

In total, this procedure uses $\widetilde{O}(\sigma^{\log_3(2)} + n/\sigma)$ colors. We pick $\sigma = n^{\log_6(3)}$ to balance terms and resulting in a procedure that uses at most $\widetilde{O}(n^{\log_6 2}) \approx \widetilde{O}(n^{0.38})$ colors. □

21.5.2 *Final notes on coloring 3-colorable graphs*

Until recently the best combinatorial algorithm to color 3-colorable graphs in polynomial was due to Avrim Blum and David Karger and uses at most $O(n^{3/8})$. This is even better than the $\widetilde{O}(n^{\log_6 2})$ we obtained, however, it is possible to improve the rounding procedure to obtain an algorithm that uses $\widetilde{O}(\Delta^{1/3}\sqrt{\ln \Delta})$ colors — in turn using balancing trick, we get an algorithm using $\widetilde{O}(n^{1/4})$ colors. Currently, the state of the art is $O(n^{0.199})$ colors by Ken-Ichi Kawarabayashi and Mikkel Thorup, and also shows a combinatorial algorithm that achieves $\widetilde{O}(n^{4/11})$.