E0 249: Approximation Algorithms February 14, 2022

Week 7
Lecture 2: Graph Coloring

Instructors: Arindam Khan, Anand Louis *Scribe: Shirish Gosavi*

1 Introduction

A *proper coloring* is an assignment of colors to the vertices of a graph so that no two adjacent vertices have the same color. A $k-coloring$ of a graph is a proper coloring involving a total of k colors. A graph that has a $k-coloring$ is said to be $k-colorable$. Formally, a graph $G = (V, E)$ is said to be k-colorable if there exists a "coloring" $f : V \to [k]$ such that $f(u) \neq f(v)\ \forall (u, v) \in E$.
The k-coloring problem asks whether G can be properly coloured using at most k colours. The 2-coloring problem is easy. Only bipartite graphs, including trees and forests, are 2-colorable. We can run BFS starting from any vertex u and use one color for the vertices on even levels and the second color for the vertices on odd levels. If this results in a valid coloring, then the given graph is 2-colorable, otherwise not. Thus, 2-coloring is in P.
2-coloring is easy, but 3-coloring is NP-hard. This can be shown using reduction from the 3-SAT problem. In this lecture, we will study algorithms based on semidefinite programming technique to color a given 3-colorable graph using as few colors as possible.
We will begin section **2** with a simple algorithm to color a 3-colorable graph using $O(n/\log n)$ colors. It will be followed by description and analysis of Wigderson's algorithm [Wig82], which provides coloring using $O(\sqrt{n})$ colors. In section **3** we will see a rounding algorithm based on semidefinite programming technique which gives coloring using $\tilde{O}(n^{0.387})$ colors. We will discuss a second rounding algorithm, also known as KMS algorithm, that gives $\tilde{O}(n^{1/4})$ algorithm in section **4**.
Specific references are mentioned at appropriate places. A couple of general references used are [Vaz13] and [WS11].

2 Wigderson's Algorithm

Any graph with n vertices can be colored using n colors trivially (use a different color for each vertex). It is possible to color a *3-colorable* graph using $O(n/\log n)$ colors using the following algorithm 1. The main idea is to partition the set of vertices and color each partition using a different set of 3 colors so that the coloring remains valid when the entire graph is considered. To be specific, the graph is partitioned into $(n/\log n)$ partitions with each partition having $(\log n)$ vertices. For each partition, a set of 3 colors is selected and using brute-force method the coloring can be decided in $O(3^{\log n})$ time. Therefore, the total number of colors required will be $3 \cdot (n/\log n)$, that is, $O(n/\log n)$.

Algorithm 1: Algorithm 1: Coloring using $O(n/\log n)$ colors

1. Partition the set of vertices into $(n/\log n)$ parts, each of $(\log n)$ size.
2. Color each part using a set of 3 new colors in $O(3^{\log n})$ time using brute-force method
3. Output the coloring assignment for the entire vertex set.

Wigderson's algorithm [Wig82] improves this bound to $O(\sqrt{n})$. It mainly uses the following facts.

1. Graph of maximum degree d can be colored using d+1 colors. When we reach a vertex v for coloring, at most d colors would be already used for coloring its neighbors. Therefore, if we have $(d+1)$ colors available at our disposal, one unused color out of the $(d+1)$ will always be available to color the vertex. Since this holds for each vertex in the graph, the graph can certainly be colored using $(d+1)$ colors.

2. If $G = (V, E)$ is k-colorable, then for any $v \in V$, the graph induced on $N(v)$ is (k-1)-colorable. This can be justified using contradiction. If $N(v)$ is not (k-1)-colorable, at least k colors are required to color $N(v)$. If k or more colors are used to color adjacent vertices of v, we need one new color different from all the colors used for $N(v)$ vertices to color v. It means the graph is not k-colorable contradicting our assumption. Therefore, if $G = (V, E)$ is k-colorable, then for any $v \in V$, the graph induced on $N(v)$ is (k-1)-colorable. If $k = 3$, that is, if the graph is 3-colorable, $N(v)$ for any vertex $v \in V$ will be 2-colorable and the coloring can be obtained in polynomial time.

The main idea in Wigderson's algorithm is that if an uncolored vertex $v \in V$ has "many" uncolored neighbors, then they can be 3-colored in polynomial time. The algorithm is described below.

Algorithm 2: Wigderson's algorithm

1. If graph has an uncolored vertex with $\geq t$ uncolored neighbors, then color it and its uncolored neighbors using 3 new colors.
   Repeat till there is no vertex with degree $\geq t$ left in the graph.
2. After step 1, the graph will not have any vertex with degree $\geq t$.
   Therefore, the remaining vertices can be colored using t colors.
   This can be done using brute-force method in polynomial time.
3. Output the coloring assignment for the entire vertex set.

Analysis

- Each iteration of step (1) colors at least (t+1) vertices. Therefore, there can be at most $n/(t+1)$ iterations. Once we reach a state when there is no vertex with degree $\geq t$, the maximum degree of any vertex in the uncolored graph has degree $< t$ and therefore t colors are enough to color the remaining graph.
- Total number of colors used $\leq 3\frac{n}{t+1} + t$
- Choosing $t = \lceil\sqrt{n}\rceil$, the algorithm will use $O(\sqrt{n})$ colors for coloring G.

Thus, Wigderson's algorithm achieves coloring a 3-colorable graph using $O(\sqrt{n})$ colors in polynomial time.

3 Semidefinite Programming Relaxation

We will see algorithms based on semidefinite programming technique that improve the bound further, that is, they give bound better than $O(\sqrt{n})$ of Wigderson's algorithm.
The semidefinite programming relaxation for graph coloring assigns a unit vector to each vertex of a graph $G = (V, E)$ such that certain separation properties are satisfied for vectors corresponding to each pair of adjacent vertices.
We can use the following vector program to decide the minimum number of colors required to color a graph:

$$\begin{aligned} &\min \lambda \\ &\text{subject to} \\ &\langle v_i, v_j \rangle \leq \lambda, \ \forall (i,j) \in E \\ &\|v_i\|^2 = 1, \ \forall i \in V \\ &v_i \in \mathbb{R}^n, \ \forall i \in V \end{aligned}$$

If a graph is known to be k-colorable, we can decide one unit vector corresponding to each color class. The graph coloring problem reduces to assigning one of the k unit vectors to each vertex such that no adjacent vertices get the same unit vector. For assignment of unit vectors to vertices, we can use the vector program given above. However, we need to decide the value of λ to be able to use the vector program. It is done as follows.

Claim: For any k unit vectors $u_1, ..., u_k \in \mathbb{R}^d$, $\underset{i,j \in [k]}{max} \langle u_i, u_j \rangle \geq -\frac{1}{k-1}$

Proof. We know that

$$\begin{aligned} &\|\sum_{i=1}^{k} u_i\|^2 \geq 0 \\ \implies &\sum_{i=1}^{k} \|u_i\|^2 + 2\sum_{i<j} \langle u_i, u_j \rangle \geq 0 \\ \implies &2\sum_{i<j} \langle u_i, u_j \rangle \geq -k \ (\because \|u_i\|^2 = 1 \ \forall i) \\ Now, &2\binom{k}{2} \underset{i,j}{max} \langle u_i, u_j \rangle \geq 2\sum_{i<j} \langle u_i, u_j \rangle \geq -k \\ &\underset{i,j}{max} \langle u_i, u_j \rangle \geq -k \cdot \frac{1}{k(k-1)} \\ &\underset{i,j}{max} \langle u_i, u_j \rangle \geq \frac{-1}{k-1} \end{aligned}$$

Hence proved.

□

From the claim given above, we note that λ has to be greater than or equal to $-\frac{1}{k-1}$. If $\lambda < -\frac{1}{k-1}$, we will not be able to find a feasible solution as at least one of the pairs of values will certainly violate the constraint $\langle v_i, v_j \rangle \leq \lambda$. It is also known that for any integer $k \leq (n+1)$ there exist k unit vectors in $\mathbb{R}^n$ such that their pairwise inner products are $-1/(k-1)$ [KMS98]. Therefore, we can choose λ equal to $-\frac{1}{k-1}$ and change the inequality in the constraint $\langle v_i, v_j \rangle \leq \lambda$ to equality. Thus, the vector program will reduce to finding the values of $v_i, \forall i \in \{V\}$ such that:

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

If a graph is 3-colorable, we can set k to 3 and the above vector program reduces to:

$$\langle v_i, v_j \rangle = -\frac{1}{2},\ \forall (i,j) \in E$$
$$\|v_i\|^2 = 1,\ \forall i \in V$$
$$v_i \in \mathbb{R}^n,\ \forall i \in V$$

Ideally, the solution of the above vector program must give only 3 distinct vectors corresponding to each color class and each vertex must get one of these 3 vectors assigned to it (that is, each v_i must get one of the three values). However, the solution of the vector program may not be ideal as we obtain the solution for the relaxed version.
Therefore, we need a way to round the solution into a valid coloring of the graph. That is, we cannot expect an algorithm to color the whole graph properly. Instead, we aim for an algorithm that colors the graph almost properly. If we have an algorithm that colors f fraction of the vertices properly using c colors, then we can use the following strategy to color the entire graph: Color a "large" fraction (that is f) of the vertices using a "few" colors (that is c colors), and recurse on the remaining graph.

Lemma 1. *If there is an algorithm to color at least f fraction of the vertices using c colors, it can be used to color the entire graph in $O(\frac{c}{f} \log\ n)$ colors*

Proof. Start with coloring the graph using c colors. Since it is guaranteed that f fraction of the vertices will be colored properly, remove those vertices and the edges incident on them. Take the remaining graph and use the algorithm to color it using a new set of c colors. This process can be repeated till all the vertices are colored.
The number of uncolored vertices after i iterations will be $\leq (1-f)^i n \leq e^{-fi} n$.
Therefore, there will be no uncolored vertices after $O(\frac{1}{f} \log\ n)$ iterations. □

In the next two sections we will see two rounding schemes, developed by Karger et al. [KMS98], for transforming optimal vector colorings into getting colorings for a fraction f of vertices that are colored properly using c colors.

4 Rounding Algorithm-1

In this section we will discuss rounding algorithm using hyperplane partitions [KMS98]. We will consider 3-colorable graphs. If we want to say that a graph is c-colorable, we need to find c independent sets. If we have c independent sets, we can use a different color to color all the vertices in an independent set and thus we can get a c-coloring of the graph.

Definition 2. *A hyperplane H is said to separate two vectors if they do not lie on the same side of the hyperplane. For any edge (i, j) ∈ E, the hyperplane is said to cut the edge if it separates the vectors v_i and v_j associated with the vertices i and j in a vector coloring of G.*

Lemma 3. *[GW95] Given two vectors at an angle of θ, the probability that they are separated by a random hyperplane is exactly θ/π.*

Vectors for "edges" are far apart. We can think of using GW-algorithm to cut the edges of a graph so that vertices collected together on each side of the hyperplane will form independent sets. For a 3-colorable graph, we have $\langle v_i, v_j \rangle \leq -1/2, \forall (i,j) \in E$. $\langle v_i, v_j \rangle \leq -1/2$ implies that the angle between them is at least $2\pi/3$. If we fix an edge (i,j), the probability that it will not get cut by a random hyperplane is

$$\Pr_{g \sim \mathcal{N}(0,1)^n}[\text{g doesn't separate } (i,\ j)] = \left(1 - \frac{2\pi/3}{\pi}\right) = \frac{1}{3} \tag{1}$$

As the probability of an edge not getting cut is large, there can be many edges in the supposedly independent set of vertices obtained by using a single hyperplane to cut the edges. We can, however, use many independent hyperplanes to cut the edges. If we use k hyperplanes, there will be 2^k regions created by the intersecting hyperplanes. We expect a large number of edges will get cut and vertices belonging to a particular region will most likely form an independent set. It is possible that the vertices belonging to a particular region may have edges between them (that is, some edges may not get cut and both the end points may get mapped to a single region). However, we can delete such edges and the vertices connected by such edges. Even after this correction, we expect that we will have many vertices in each region which can be colored using a single color.

The algorithm 3 is given without specifying the value of k. Its analysis tells us what value of k will give us the desired results.

Algorithm 3: Rounding algorithm -1 : Algorithm for obtaining independent sets in V

1. Sample $g_1, ..., g_k \sim \mathcal{N}(0,1)^n$ independently.
2. For each $x \in \{-1,1\}^k$, let $S_x \stackrel{def}{=} \{i \in V : \langle v_i, g_l \rangle$ has same sign as $x_l \ \forall l \in [k]\}$.
   Color each S_x with a different color.
3. For any $(i,j) \in E$, if the previous step assigned the same color to both the vertices i and j, then uncolor both of them.

Analysis
The probability that an edge does not get cut by a single hyperplane is $\frac{1}{3}$. Therefore, it does not get cut by any of the k hyperplanes is $\frac{1}{3^k}$. Thus,

$$\Pr[i,\ j \text{ assigned same color in step 2}] = \frac{1}{3^k} \tag{2}$$

If we choose $k = 2 + \log_3 \Delta$, where Δ is the maximum vertex degree, we have

$$\Pr[\ i,j \text{ assigned same color in step 2}] = \frac{1}{3^k} = \frac{1}{9\Delta} \tag{3}$$

If m denotes the number of edges in the graph, that is $m = |E|$, then $m \leq \frac{n\Delta}{2}$. Thus, the expected number of edges which have both endpoints colored the same is

$$\text{E[number of edges uncolored in step 3]} = \frac{m}{9\Delta} \leq \frac{n\Delta/2}{9\Delta} = \frac{n}{18} \tag{4}$$

Let X be a random variable denoting the number of edges which have both endpoints colored the same. By Markov's inequality, the probability that there are more than $n/4$ edges which have both endpoints colored the same is at most

$$Pr[X \geq n/4] \leq \frac{E[X]}{n/4} \leq \frac{n/18}{n/4} < \frac{1}{2} \tag{5}$$

Therefore, with probability at least $\frac{1}{2}$, the algorithm gives a proper coloring of at least $(n/4)$ vertices. Since we use a different color for each region and there are 2^k regions, we have

$$\text{Number of colors used} \leq 2^k = 2^{2+\log_3 \Delta} = 4\Delta^{\log_3 2} = O(\Delta^{\log_3 2}) \tag{6}$$

Thus, we can color $n/4$ vertices properly using $O(\Delta^{\log_3 2})$ colors with probability at least $\frac{1}{2}$. Since $\Delta < n$ and $\log_3 2 < 0.631$, the algorithm colors $\geq n/4$ vertices properly with probability $\geq \frac{1}{2}$ and using $O(n^{0.631})$ colors.

Algorithm 4: Algorithm (t) : Coloring algorithm using rounding algorithm-1

1. If the graph has an uncolored vertex with $\geq t$ uncolored neighbors, then color it and its uncolored neighbors using 3 colors.
2. Color remaining vertices using the rounding algorithm 1.

If this algorithm is used k times, the probability that it colors $n/4$ vertices properly is $1 - \frac{1}{2^k}$. Removing the nodes that have been colored and iterating the procedure on the remaining subgraph, the entire graph can be colored in $O(\log_2 n)$ iterations in expectation and using $O(n^{0.631} \log_2 n)$ colors.
We can combine the algorithm 3 with Wigderson's algorithm to get the final algorithm 4.
The total number of colors used will be $\leq 3\frac{n}{t+1} + O(t^{\log_3 2} \log n)$. The algorithm 4 has two steps. The first step uses $3n/(t+1)$ colors in total since we remove at least (t+1) vertices from the graph each in each iteration. The second step uses $O(t^{\log_3 2} \log n)$ colors. If we choose $t \approx n^{0.613}$, both the steps will use $O(n^{0.387})$ colors and therefore total number of colors used will be $\tilde{O}(n^{0.387})$.
Thus, we get an $\tilde{O}(n^{0.387})$ color algorithm using semidefinite programming which gives improvement over Wigderson's $O(\sqrt{n})$ color algorithm.

5 Rounding Algorithm-2

In this section we discuss another, a more powerful, rounding scheme proposed Karger et al. [KMS98] that improves the bound to $O\big(\Delta^{\frac{1}{3}} \text{polylog}(n)\big)$, that is, it colors the graph properly with $O\big(\Delta^{\frac{1}{3}} \text{polylog}(n)\big)$ colors. Just like for the previous rounding scheme, we describe a procedure with a parameter τ to construct independent sets and decide the value of τ in the analysis part.

Algorithm 5: Rounding algorithm-2 : Algorithm for obtaining independent sets in V

1. Sample g $\sim \mathcal{N}(0,1)^n$. Let $S_g \stackrel{def}{=} \{i \in V : \langle v_i, g\rangle \geq \tau\}$.
2. If the graph induced on S_g contains any edge, delete both the vertices. Output the remaining set of vertices (denote this set by S'_g).

The set S'_g returned by 5 is an independent set.

Let $\phi(x)$ be the cumulative distribution function of a standard normal variable, and let $\overline{\phi}(x) \stackrel{def}{=} 1 - \phi(x)$. For a fixed $i \in V$, we have $\Pr[i \in S_g] = \overline{\phi}(\tau)$.
The bound on the probability that a vertex gets deleted in step (2) of algorithm 5 is

$$\Pr[i \notin S'_g | i \in S_g] = \Pr[\exists j \in N(i) \text{ such that } \langle v_j, g\rangle \geq \tau | \langle v_i, g\rangle \geq \tau] \leq \sum_{j \in N(i)} \Pr[\langle v_j, g\rangle \geq \tau | \langle v_i, g\rangle \geq \tau] \quad (7)$$

Fix an edge $(i,j) \in E$. As G is 3-colorable, $\langle v_i, v_j\rangle = -\frac{1}{2}$. Therefore, $v_j = -\frac{1}{2}v_i + \frac{\sqrt{3}}{2}u$ for some unit vector $u \perp v_i$ (this value of v_j satisfies both constraints, $\langle v_i, v_j\rangle = -\frac{1}{2}$ and $\|v_j\| = 1$). Solving for u, we get $u = \frac{2}{\sqrt{3}}\big(\frac{1}{2}v_i + v_j\big)$. Since $u \perp v_i$, $\langle v_i, g\rangle$ and $\langle u, g\rangle$ are independent standard normal random variables.
$\langle v_i, g\rangle \geq \tau$ and $\langle v_j, g\rangle \geq \tau$ implies $\langle u, g\rangle = \frac{2}{\sqrt{3}}\big(\frac{1}{2}\langle v_i, g\rangle + \langle v_j, g\rangle\big) \geq \sqrt{3}\tau$. Therefore,

$$\Pr[\langle v_j, g\rangle \geq \tau | \langle v_i, g\rangle \geq \tau] \leq \Pr[\langle u, g\rangle \geq \sqrt{3}\tau | \langle v_i, g\rangle \geq \tau] = \Pr[\langle u, g\rangle \geq \sqrt{3}\tau] = \overline{\phi}(\sqrt{3}\tau) \quad (8)$$

Since the maximum degree of any vertex in graph is Δ, the size of neighborhood for any vertex is $\leq \Delta$. Therefore,

$$Pr[i \notin S'_g | i \in S_g] \leq \Delta \overline{\phi}(\sqrt{3}\tau) \tag{9}$$

If we choose τ such that $\overline{\phi}(\sqrt{3}\tau) \leq 1/(2\Delta)$, then

$$\Pr[i \in S'_g] \geq \frac{1}{2}\Pr[i \in S_g] \geq \frac{1}{2}\overline{\phi}(\tau) \tag{10}$$

To get lower bound for $\overline{\phi}(\tau)$, we use the following lemma.

Lemma 4. *For $x > 0$, $\frac{x}{1+x^2}p(x) \leq \overline{\phi}(x) \leq \frac{1}{x}p(x)$.*

Proof. Note that $p'(s) = -sp(s)$, so that $\left(\frac{1}{s}p(s)\right)' = \left(1 + \frac{1}{s^2}\right)p(s)$. To obtain the lower on $\overline{\phi}$, observe that

$$\begin{aligned}
\left(1 + \frac{1}{x^2}\right)\overline{\phi}(x) &= \int_x^\infty \left(1 + \frac{1}{x^2}\right) p(s)ds \\
&\geq \int_x^\infty \left(1 + \frac{1}{s^2}\right) p(s)ds \\
&= -\frac{1}{s}p(s)\Big]_x^\infty = \frac{1}{x}p(x)
\end{aligned}$$

Dividing both sides of the inequality by $1+\frac{1}{x^2}$ gives the lower bound on $\overline{\phi}(x)$. To obtain the upper bound, observe

$$\begin{aligned}
\overline{\phi}(x) &= \int_x^\infty p(s)d(s) \\
&\leq \int_x^\infty \left(1 + \frac{1}{s^2}\right) p(s)ds = \frac{1}{x}p(x)
\end{aligned}$$

as above. □

Setting $\tau = \sqrt{\frac{2}{3}ln\Delta}$, we have

$$\begin{aligned}
\overline{\phi}(\sqrt{3}\tau) &\leq \frac{1}{\sqrt{3}\tau}\frac{1}{\sqrt{2\pi}}e^{-\frac{3\tau^2}{2}} \ (\because \text{lemma 4 and normal distribution property}) \\
&\leq \frac{1}{\sqrt{2\ln\Delta}}\frac{1}{\sqrt{2\pi}}e^{-\ln\Delta} \\
&\leq \frac{1}{2\Delta} \\
\overline{\phi}(\tau) &\geq \frac{\tau}{1+\tau^2}\frac{1}{\sqrt{2\pi}}e^{-\frac{\tau^2}{2}} \ (\because \text{lemma 4 and normal distribution property}) \\
&\geq \frac{1}{2\tau}\frac{1}{\sqrt{2\pi}}e^{-\frac{\ln\Delta}{3}} \\
&= \Omega\left(\Delta^{-\frac{1}{3}}(\ln\Delta)^{-\frac{1}{2}}\right)
\end{aligned}$$

Therefore, by linearity of expectation, $\mathrm{E}[|S'_g|] = \Omega\big(n\Delta^{-\frac{1}{3}}(\ln\Delta)^{-\frac{1}{2}}\big)$.
Therefore, we can color the graph using $O\big(\Delta^{\frac{1}{3}}\mathrm{polylog}(n)\big)$ colors.
The algorithm 6 on the next page lists the steps involved.
Choosing $t = n^{3/4}$,

$$\text{Total number of colors used} \leq 3\frac{n}{t+1} + O\big(t^{\frac{1}{3}}\mathrm{polylog}(n)\big) = \tilde{O}(n^{1/4}) \tag{11}$$

Current best-known bounds is $\tilde{O}(n^{0.19996})$ colors for coloring a 3-colorable graph [KT17].

Algorithm 6: Algorithm (t) : Coloring algorithm using rounding algorithm-2

1. If graph has uncolored vertex with $\geq t$ uncolored neighbors, then color it and its uncolored neighbors using 3 colors.
2. Color remaining vertices using the rounding algorithm 2.

References

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, nov 1995.

[KMS98] David R. Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *CoRR*, cs.DS/9812008, 1998.

[KT17] Kenichi Kawarabayashi and Mikkel Thorup. Coloring 3-colorable graphs with less than n1/5 colors. *Journal of the ACM*, 64(1), March 2017.

[Vaz13] V.V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, 2013.

[Wig82] Avi Wigderson. A new approximate graph coloring algorithm. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *STOC*, pages 325–329. ACM, 1982.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*, page 469–484. Cambridge University Press, 2011.