Coloring 3-Colorable Graphs Notes

Lily Bhattacharjee

May 22, 2019

1 Introduction

Graph coloring is an optimization problem that involves – in one of its simplest cases – vertex coloring: assigning the vertices in a particular graph to different colors such that no two adjacent vertices are the same color. A graph is considered n-colorable when this can be done with a set of n colors, but in this section we will focus on arbitrary 3-colorable graphs $G = (V, E)$ where there are $|V|$ total vertices and $O(\sqrt{|V|})$ colors are used. We will see that a semidefinite programming algorithm that achieves this can be done in $\tilde{O}(n^{0.387})$ time ($n = |V|$).

2 The meaning of $\tilde{O}$

Definition: Similarly to O-notation, $\tilde{O}$ is a way of writing upper bounds; some function $g(n) = \tilde{O}(f(n))$ if $g(n)$ can be written in the following form: $g(n) = O(f(n)\log^c(n))$ where $n \geq$ some constant n_0, constant $c \geq 0$.

2.1 A $\sqrt{n}$-coloring algorithm

Graph coloring is a difficult approximation problem, but some special cases e.g. 2-coloring, (max degree + 1)-coloring can be solved in polynomial time. Looking closer at 2-coloring, we can give an algorithm in which – given some graph G – we start at an arbitrary vertex v, assigning it Color A. Hence, all of v's neighbors are forced to be assigned Color B. For each of these 1st-level connections, we repeat the process, selecting the opposite color (Color A), continuing until either the entire graph is colored or we run into a bad coloring i.e. 2 vertices that share an edge have been assigned the same color. If there is any such bad coloring, the graph is not 2-colorable because all of our coloring decisions were forced. Hence, we see that our algorithm is linear in terms of the number of vertices in G.

Likewise, considering (max degree + 1)-coloring, we provide an algorithm in which we select arbitrary vertices v repeatedly and assign unique colors to each of $v \bigcup N(v)$ ($N(v)$ represents the neighborhood of vertices immediately adjacent to v). Because we always have a maximum of (max degree + 1) assignments to make considering any v individually and exactly that many colors available, we will never encounter a conflict in which we run out of unique colors and create a bad coloring.

Knowing that 2-coloring and (max degree + 1)-coloring can be done in polynomial time, we can come up with an algorithm for coloring a 3-colorable graph $G = (V, E)$ with $O(\sqrt{n})$ colors:

while $v \in V$ s.t. $\deg(v) \geq n$:
  pick 3 new colors: A, B, C
  color v with A
  // the below is possible because G is known to be 3-colorable
  use the 2-coloring algorithm described above to color $N(v)$ with B, C
  remove the colored vertices from the graph
use the (max degree + 1)-coloring algorithm described above to color the rest of the vertices with $\sqrt{n}$ colors maximum

2.2 Proof of max-$4\sqrt{n}$ colorability

Claim: The algorithm for $O(\sqrt{n})$ coloring as described above can color any 3-colorable graph $G = (V, E)$ with max $4\sqrt{n}$ colors.

Proof: According to the pseudocode above, whenever we find v s.t. $\deg(v) \geq \sqrt{n}$, we pick 3 new colors. Because we remove $v \bigcup N(v)$ at the end of the iteration, the loop will run max $\frac{n}{\sqrt{n}}$ times. This means using max $\frac{3n}{\sqrt{n}} = 3\sqrt{n}$ colors throughout the while loop overall. We know that the final step, which uses the (max degree + 1)-coloring algorithm, will use $\sqrt{n}$ new colors total, so the number of unique colors used in the algorithm is upper-bounded by $3\sqrt{n} + \sqrt{n} = 4\sqrt{n}$ colors total.

3 A semidefinite programming algorithm

We'd like to come up with an algorithm that will more strictly upper-bound the number of colors used to appropriately color a 3-colorable graph. The below is a vector program with a vector v_i corresponding to every vertex $i \in V$:

$$\begin{gathered}\text{minimize } \lambda \\ \text{subject to } v_i \cdot v_j \leq \lambda, \forall (i, j) \in E, \\ v_i \cdot v_i = 1, \forall i \in V, \\ v_i \in \Re^n, \forall i \in V \text{ (we consider vectors in the real space only)}\end{gathered}$$

3.1 Solving with $\lambda \leq -\frac{1}{2}$

Claim: For an arbitrary 3-colorable graph, we can solve the vector program described above with $\lambda = -\frac{1}{2}$.

Proof: An example solution to this program is described by an equilateral triangle in which the unit vectors v_i corresponding to each vertex are matched with three different colors. The angle between vectors of the same color is 0 (only one vector per color in this solution), and the angle between vectors of different colors is $\frac{2\pi}{3}$ (3 vectors splitting the central 2π angle). Hence, as we have already satisfied condition (2) and (3) of the vector program, we show that (1) also holds:

$$v_i \cdot v_j = ||v_i||||v_j||\cos(\tfrac{2\pi}{3}) = -\tfrac{1}{2}$$

Because there exists a solution s.t. $\lambda = -\frac{1}{2}$, for the optimal solution (which may or may not be this one), $\lambda \leq -\frac{1}{2}$.

3.2 Solving with $v_i \cdot v_j = -\frac{1}{2}, \forall (i, j) \in E$

Claim: For an arbitrary 3-colorable graph, we can solve the vector program described above with $v_i \cdot v_j = -\frac{1}{2}, \forall (i, j) \in E$.

Proof: To prove the claim, we first define the term *semicoloring*.

Definition: A *semicoloring* is a vertex coloring of a graph G ($n = |V|$) where $\leq \frac{n}{4}$ edges have vertex endpoints with the same color (bad colorings), indicating the $\leq \frac{n}{2}$ vertices are correctly colored (edge endpoints have different colors).

If we can come up with an algorithm that creates a k-semicoloring, we can color the whole graph with $k \log n$ colors. Initially, we semicolor with k colors and only consider correctly-colored vertices, which should be at minimum $\frac{n}{2}$ according to the definition above. Hence, the number of incorrectly-colored vertices is upper-bounded at $\frac{n}{2}$. We take k new colors and perform a k-semicoloring on the remaining vertices, which will leave $\leq \frac{n}{4}$ incorrectly-colored vertices, and so on. Following this pattern, there will be $\log n$ iterations until G is correctly colored. If k new colors are selected each time, this upper-bounds the total number of colors used at $k \log n$.

Everything we just described is based on the assumption that a randomized algorithm for generating a semicoloring exists. We solve the vector program described above, selecting $t = 2 + \log_3 \Delta$ random vectors $r_1, ..., r_t$, with $\Delta = \max \deg(G)$. The t random vectors will create 2^t different regions for the vectors v_i: (1) $r_j \cdot v_i \geq 0$, (2) $r_j \cdot v_i < 0$, $\forall j \in [1, t]$. The vectors in each region are assigned different colors.

3.3 Probability of a $4\Delta^{\log_3 2}$ semicoloring

Claim: The coloring algorithm semicolors $4\Delta^{\log_3 2}$ colors with probability ≥ 0.5.

Proof: Our algorithm created 2^t different regions and assigned each a different color. Because $t = 2+\log_3 \Delta$, $2^t = 2^{2+\log_3 \Delta} = 4 * 2^{\log_3 \Delta} = 4\Delta^{\log_3^2}$ colors. We must now show that the probability that this occurs is ≥ 0.5.

There are a few possibilities:

- For some edge (i, j), endpoints i, j are assigned different colors. Because these vertices have been correctly colored, there is no need to consider them for re-coloring in the next iteration.
- For some edge (i, j), endpoints i, j are assigned the same color. This is equivalent to the probability that i, j fall into the same region.

  $P(1 \text{ random hyperplane separates } i, j) = \frac{1}{\pi} \arccos(v_i \cdot v_j)$
  $P(\text{t independent hyperplanes separate } i, j) = (\frac{1}{\pi} \arccos(v_i \cdot v_j))^t$
  $P(\text{t independent hyperplanes do not separate } i, j) = (1 - \frac{1}{\pi} \arccos(v_i \cdot v_j))^t$

  $P(i, j \text{ are assigned the same color}) = (1 - \frac{1}{\pi} \arccos(v_i \cdot v_j))^t \leq (1 - \frac{1}{\pi} \arccos(\lambda))^t$, following from the vector program definition

  $(1 - \frac{1}{\pi} \arccos(\lambda))^t \leq (1 - \frac{1}{\pi} \arccos(-\frac{1}{2}))^t$

  $(1 - \frac{1}{\pi} \arccos(-\frac{1}{2}))^t = (1 - \frac{1}{\pi} * \frac{2\pi}{3})^t = (\frac{1}{3})^t \leq \frac{1}{9\Delta}$

  Hence, $P(i, j \text{ are assigned the same color}) \leq \frac{1}{9\Delta}$.

If $m = |E|$, $m \leq \frac{n\Delta}{2}$ (recall $n = |V|$, so if each vertex is of max degree, the number of edges will equal $\frac{n\Delta}{2}$). From above, the number of edges with same-colored vertices $\leq \frac{m}{9\Delta} \leq \frac{n}{18}$. Create a random variable $X =$ number of edges with same-colored endpoints. By Markov's inequality:

$$P(X \geq \frac{n}{4}) \leq \frac{E[X]}{n/4} \leq \frac{n/18}{n/4} = \frac{2}{9} \leq \frac{1}{2}$$

We know n upper-bounds max degree Δ, so this algorithm semicolors with $O(n^{\log_3 2}) = \tilde{O}(n^{\log_3 2})$ colors. This isn't as good as our starting algorithm $(O(n^{1/2})) - \log_3 2 \approx 0.631 \geq 0.5$, but we can improve using some of the ideas we've already explored.

Assume some parameter σ. Our new algorithm is the following:

```
while v ∈ V s.t. deg(v) ≥ σ:
    pick 3 new colors: A, B, C
    color v with A
    // the below is possible because G is known to be 3-colorable
    use the 2-coloring algorithm to color N(v) with B, C
    remove the colored vertices from the graph
use the semicoloring algorithm described above to color the rest of the vertices with O(σ^(log_3 2)) colors maximum
```

3.4 Probability of a $O(n^{0.387})$ semicoloring

Claim: The algorithm we just described semicolors with ≥ 0.5 probability any arbitrary 3-colorable graph with $O(n^{0.387})$ colors.

Proof: Our loop removes $\geq \sigma$ vertices in every iteration, so we use max $\frac{3n}{\sigma}$ colors overall (n total iterations, 3 new colors used in each one). If we set σ s.t. $\frac{n}{\sigma} = \sigma^{\log_3 2}$ or $\sigma = n^{\log_6 3} \approx n^{0.613}$, we can balance the number of colors used in both parts of the algorithm. Dividing our exponent by 2, we then have the algorithm overall using $O(n^{0.387})$ colors.