

Lecture 4

Lecturer: Ronitt Rubinfeld

Scribe: Daph Guo

Topics covered: Analysis of the randomized $(\Delta + 1)$ -coloring algorithm presented in the last lecture, brief preview of the link between sublinear and distributed algorithms (next lecture).

1 Recall: Coloring algorithm

To recap, our goal is to solve the following problem (see previous notes for more details):

Definition 1 $(\Delta + 1)$ -coloring problem: *Given* an undirected graph $G(V, E)$ with (bounded but not necessarily constant) maximum degree Δ , *return* an assignment of each vertex $v \in V$ to a color $c_v \in \{1, \dots, \Delta + 1\}$ so that no edge is monochromatic; that is, for all edges $\{u, v\} \in E$, $c_u \neq c_v$.

For this particular instance of coloring, $O(n\Delta)$ runtime can be achieved with a simple greedy algorithm: for each vertex, check all of its neighbors and assign it a color that is unused by any of them. By the pigeonhole principle, this is always possible.

The question is whether we can use randomization to do better in the case where $\Delta \in \tilde{\Omega}(\sqrt{n})$. And the answer is yes!

Algorithm 1 Randomized $\tilde{O}(n\sqrt{n})$ -time $(\Delta + 1)$ -coloring

- 1: Initialize $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_{\Delta+1} = \emptyset$. We maintain $\mathcal{C}_k :=$ the set of vertices that have been assigned color k .
 - 2: Let $v_1, \dots, v_n$ be the vertices of G sampled in a uniformly random order.
 - 3: **For** $i = 1, \dots, n$:
 - A. Uniformly sample a color $c \in \{1, \dots, \Delta + 1\}$.
 - B. Try to color v_i with c : for every $u \in \mathcal{C}_c$, check whether $\{u, v_i\}$ is an edge in G . (In adjacency matrix format, this is an $O(1)$ -time query.)
 - C. **If** no vertex in $\mathcal{C}_c$ is a neighbor of v_i , we can color v_i with c : update $\mathcal{C}_c \leftarrow \mathcal{C}_c \cup \{v_i\}$ and **continue** to the next vertex to be colored. **Else, goto** step **A** and sample a color again.
-

Intuitively, this algorithm takes a “color-based” approach, where we choose a color and loop through the vertices that have been assigned that color to check for edges. This could be thought of as a dual to the “vertex-based” approach of the greedy algorithm, which loops through *neighbors* to check for *color usage*. This approach ends up being more efficient in the high-degree many-potential-color case we are considering, where vertex neighborhoods are large, but not *that* many vertices are expected to be assigned to the same color.

Now, considering this procedure, one might have some worries:

- **We could sample a bad color, then resample the same bad color again.** It is true that sampling with replacement could force our algorithm to do more work in some cases. However, we will see that it makes the analysis easier to assume the recursive case is exactly identical! The same analysis will show that this “oversight” is OK :).
- **We seem to have a guaranteed correct output, but not a guaranteed runtime.** True! This is a different type of randomized algorithm than we’ve looked at before. This one could run for arbitrary amounts of time with no result. The analysis will justify that the *expected running time* is reasonable rather than that the output value is close to the true value.
- **If the color sets $\mathcal{C}_k$ become large, it will take more time to look through them.** It could be that *some* sets reach $O(n)$ elements, but since the sets partition the vertices, the average size *must* be n/Δ . This will factor into our analysis.

- Eventually, a particular vertex's neighbors could take up Δ colors, so that the probability of sampling a usable color becomes $\frac{1}{\Delta+1}$. Bounding the probabilities will show that this does not happen too often!

2 Coloring analysis

We now prove a bound on the expected runtime of our algorithm.

2.1 Preliminaries

Definition 2 Define a set of random variables X_i for $i \in \{1, \dots, n\}$, where X_i is the number of queries required to color v_i .

The values of the X_i depend on which and how many colors are sampled during the corresponding iteration of the loop.

Suppose that a particular v_i samples k colors before success; without loss of generality, denote this sequence of colors as $\{1, \dots, k\}$. Then, we have that

$$X_i = \sum_{j=1}^k |\mathcal{C}_j^{(i)}|,$$

where $\mathcal{C}_j^{(i)}$ is the state of the set $\mathcal{C}_j$ when the algorithm is processing vertex v_i . This value is technically unbounded, but it is unlikely for it to be very large.

We only ever add vertices to these sets, so their sizes change monotonically. We can therefore disregard these size changes by redefining $\mathcal{C}_j$ as the state of the set at the end of the algorithm ($i = n$), which lets us upper bound the value above as

$$X_i \leq \sum_{j=1}^k |\mathcal{C}_j|.$$

(**Scribe note:** There was an... *extended* discussion in lecture on whether this summation violated our previous declaration that colors would be chosen without replacement. I will not be transcribing it beyond this note, since it turned out to be a non-issue. If you are not confused, disregard this parenthetical.)

(Colors will be chosen without replacement for the entirety of this analysis. The sequence of colors $\{1, \dots, k\}$ considered here is completely arbitrary and can contain duplicates, since this summation represents the colors used at one *particular step*. We will see that our analysis never requires claiming anything about this sum. k does not have to be less than $\Delta + 1$ and can in fact be any positive integer.)

We run through the loop for all v_i in some order and do a constant amount of work in initialization and around our edge queries, so our total expected runtime is

$$\text{Exp}[\text{runtime}] = \text{Exp}\left[\sum_{i=1}^n X_i \cdot O(1)\right] = O(1) \cdot \sum_{i=1}^n \text{Exp}[X_i]$$

by linearity of expectation.

Note that we are dealing with two sources of randomness in the algorithm: the vertex permutation order, and the sequence of sampled colors. For the first half of our analysis, we will fix the former and assume there is a canonical vertex ordering. After we analyze runtime under that restriction, we will reincorporate the random ordering at the end.

2.2 Fixed vertex ordering

Definition 3 The preceding degree of a vertex $\deg^<(v_i)$ is the number of neighbors of v_i that appear before v_i in the vertex ordering.

Notice that this is exactly equal to the number of neighbors that have already been colored by the time the algorithm processes v_i , which depends on the ordering. For now, we can assume these are fixed values; we will analyze them as random variables in the next section.

Claim 4 The expected number of edge queries for processing vertex i can be bounded as:

$$\text{Exp}[X_i] \leq \frac{n}{\Delta + 1 - \deg^<(v_i)}.$$

Proof By the definition of expected value,

$$\begin{aligned} \text{Exp}[X_i] &= \sum_{c=1}^{\Delta+1} \Pr[\text{first sampled color for } v_i \text{ is } c] \cdot \text{Exp}[X_i \mid \text{first sampled color for } v_i \text{ was } c] \\ &= \sum_{c=1}^{\Delta+1} \frac{1}{\Delta + 1} \cdot \text{Exp}[X_i \mid \text{first sampled color for } v_i \text{ was } c], \end{aligned}$$

where the first factor follows from uniform sampling.

The second factor depends on whether c was a “good” color for v_i (i.e. whether none of v_i ’s neighbors have been colored with c). If good, we only have to query at most $|\mathcal{C}_c|$ vertices to validate that it is good. If bad, we still have to do all of those queries, but we also have to sample a color again and start from scratch. Thus the expected number of queries in the bad case is $|\mathcal{C}_c| + \text{Exp}[X_i]$.

(The simplicity of this relation is why we want to assume colors are selected with replacement! Being a bit amnesiac about the colors means that nothing has changed when we recurse, and we can represent that recurrence by the expected value. It remains to justify mathematically that this assumption does not blow up our expected runtime.)

A particular color c is bad for v_i if and only if c is one of the colors used by the previously-processed neighbors of v_i , of which there can only be $\deg^<(v_i)$ out of $\Delta + 1$ total colors. Thus,

$$\Pr[c \text{ is bad for } v_i] \leq \frac{\deg^<(v_i)}{\Delta + 1}.$$

For ease of notation, we define $B(v_i)$ as the set of bad colors for v_i . Now, the full expression for the expected number of queries per vertex is:

$$\begin{aligned} \text{Exp}[X_i] &= \frac{1}{\Delta + 1} \left(\sum_{c=1}^{\Delta+1} |\mathcal{C}_c| + \sum_{c \in B(v_i)} \text{Exp}[X_i] \right) \\ &\leq \frac{n}{\Delta + 1} + \frac{1}{\Delta + 1} \deg^<(v_i) \cdot \text{Exp}[X_i]. \end{aligned}$$

The bound on the first term was obtained by noting that the color sets partition the vertices, so their sum is at most n . As for the second term: at most $\deg^<(v_i)$ neighbors have been processed at this point, so the number of ruled-out colors $|B(v_i)| \leq \deg^<(v_i)$.

Solving this inequality for $\text{Exp}[X_i]$ gives the desired result. ■

Since the denominator ranges from one to $\Delta + 1$, this value is always less than n . We cannot naively provide a better upper bound on it, though, which is troubling.

It is easy to see that we have no problem finding colors without needing many queries for the first few vertices we look at when not many colors can have been ruled out: their preceding degrees are small and the expected number of queries is close to n/Δ . Later in the game, however, we have to process all the vertices in the set of each color candidate to confirm whether it is usable, each of which is $O(n/\Delta)$ in expectation. This is represented in the expression as $\deg^<(v_i)$ growing closer to $\Delta + 1$ and thus the expected number of queries growing to $O(n)$. We want to show that the algorithm does not have to do this “too many” times.

How do we do this? We bring back randomization of vertex ordering!

2.3 Randomized vertex ordering

Claim 5 *Under a uniformly random ordering of vertices Π , the relative rank of v_i in the set of neighbors of v_i plus v_i itself is a uniform random variable in the range $\{1, \dots, \deg(v_i) + 1\}$, that is, it is equally likely to have any position in the ordering.*

Proof (sketch) This essentially follows from symmetry. Determining v_i ’s position among its neighbors in a random ordering of all vertices is equivalent to randomly ordering all the neighbors and placing v_i in a uniformly random place between them. ■

It follows that a particular vertex’s preceding degree $\deg^<(v_i)$ in a uniformly chosen Π is uniformly random in $\{0, \dots, \deg(v_i)\}$.

We condition on Π , then plug in our fixed order analysis from before:

$$\begin{aligned} \text{Exp}[\text{runtime}] &= \text{Exp}_{\Pi}[\text{Exp}[\text{runtime} \mid \Pi]] \\ &= O(1) \cdot \text{Exp}_{\Pi}\left[\sum_{i=1}^n X_i \mid \Pi\right] \\ &= O(1) \cdot \sum_{i=1}^n \text{Exp}_{\Pi}[X_i \mid \Pi] \\ &\leq O(1) \cdot \sum_{i=1}^n \text{Exp}_{\Pi}\left[\frac{n}{\Delta + 1 - \deg^<(v_i)}\right]. \end{aligned}$$

Now, we focus on bounding the expectation expression inside the sum. From **Claim 5**, the probability of $\deg^<(v_i)$ taking on any particular value d is $\frac{1}{\deg(v_i)+1}$, so:

$$* = \text{Exp}_{\Pi}\left[\frac{n}{\Delta + 1 - \deg^<(v_i)}\right] = \sum_{d=0}^{\deg(v_i)} \frac{1}{\deg(v_i) + 1} \cdot \frac{n}{\Delta + 1 - d} \quad (1)$$

$$\leq \sum_{d=0}^{\Delta} \frac{1}{\Delta + 1} \cdot \frac{n}{\Delta + 1 - d}. \quad (2)$$

This step is not obvious! Replacing $\deg(v_i)$ with its maximum possible value Δ means equation (2) sums a larger number of smaller terms than equation (1), and the inequality does not automatically follow.

To justify this bound, consider the sequence

$$A = \left\{ \frac{n}{\Delta + 1}, \frac{n}{\Delta + 1 - 1}, \frac{n}{\Delta + 1 - 2}, \dots, \frac{n}{\Delta + 1 - \Delta} \right\}.$$

Equation (1) is the average value of the first $\deg(v_i)$ terms in A , and equation (2) is the average value of all terms in A . Note that A is monotonically increasing, since the denominator always decreases. Thus, the average value of any prefix of A is at most the average value of all of A , and our bound on $*$ holds!

Continuing with our analysis,

$$\begin{aligned}
* &\leq \frac{n}{\Delta+1} \sum_{d=0}^{\Delta} \cdot \frac{1}{\Delta+1-d} \\
&= \frac{n}{\Delta+1} \sum_{d'=1}^{\Delta+1} \cdot \frac{1}{d'} \\
&= \frac{n}{\Delta+1} (\log(\Delta+1) + 1).
\end{aligned}$$

We plug this into our runtime expression to get

$$\begin{aligned}
\text{Exp}[\text{runtime}] &\leq O(1) \cdot \sum_{i=1}^n \frac{n}{\Delta+1} (\log(\Delta+1) + 1) \\
&= O\left(\frac{n^2}{\Delta+1} \log(\Delta)\right).
\end{aligned}$$

Clearly this is $\tilde{O}(n\sqrt{n})$ for $\Delta \in \tilde{\Omega}(\sqrt{n})$, as desired. Woohoo! ■

This algorithm and analysis is a relatively new result, and this is the first time it has been presented in this class.

3 A link between distributed and sublinear

We've seen a few standalone examples of sublinear algorithms, but how do we better characterize sublinear as a field of study?

A famous reduction from a particular kind of distributed algorithm to sublinear algorithms gives us a more structured way to construct sublinear results from very fast distributed algorithms. In fact, the development of this result in 2007 motivated increased research into distributed algorithms, for the purpose of using them to create super-fast sublinear algorithms.

We will discuss this in more detail next lecture.