# *Lecture 23: Randomized Algorithms*

*Anup Rao*

*March 5, 2018*

We discuss a few randomized algorithms.

So far in this course we have been learning the basic concepts of probability and counting. We have seen some examples of how these concepts can be used for applications like load balancing, polling, building a classifier and so on. In the remaining part of the course, we explore how these ideas can help with the design of randomized algorithms.

## *Graph Coloring*

Suppose you are given an undirected graph and want to color each of the vertices red, blue or green, in such a way that the number of edges that gets two distinct colors is maximized. There is a greedy algorithm to do this, but here we give a simple algorithm:

**Input:** Undirected graph $G$ with $m$ edges
**Result:** A coloring of the vertices with 3 colors
Color every vertex with a uniformly random color.

For every edge $e$, let $X_e$ denote the random variable that is 1 if the edge gets two distinct colors, and 0 otherwise. Then the number of edges colored with distinct colors is $\sum_e X_e$.

For each edge $e$, the probability that the two colors of its vertices are the same is exactly $1/3$. Thus $\mathbb{E}[X_e] = 0 \cdot 1/3 + 1 \cdot 2/3 = 2/3$. By linearity of expectation, this means that $\mathbb{E}[\sum_e X_e] = \sum_e \mathbb{E}[X_e] = 2m/3$. Thus the algorithm colors $2m/3$ edges with distinct colors in expectation.

By Markov's inequality, the probabilty that this algorithm finds a coloring that miscolors $(1.1)(m/3)$ edges is at most $1/(1.1)$. So, repeating the algorithm $t$ times, the probability that the algorithm fails to find a coloring that miscolors at most $(1.1)m/3$ edges is at most $1/(1.1)^t$, which can be made very small.

## *Dominating Set*

We are given an undirected graph on $n$ vertices with the guarantee that every vertex in the graph has degree at least $\Delta$. We want to find a *dominating set*: a small set of vertices $S$ such that every other ver-

tex in the graph is either contained in $S$ or is a neighbor of $S$. For a parameter $p$, consider the following algorithm:

**Input:** Undirected graph $G$
**Result:** A dominating set
Include every vertex of the graph in the set $X$ with probability $p$.
Let $Y$ be the set of all vertices that are excluded from $X$ and are not a neighbor of $X$. Output $X \cup Y$.

Clearly, the output of the algorithm is always a dominating set. Now let us calculate the size of the set in the output. Since every vertex is included in $X$ with probability $p$, by linearity of expectation, $\mathbb{E}[|X|] = pn$.

For any vertex $v$ let $Y_v$ be 1 if $v$ is in $Y$, and 0 else. Since $v$ has degree at least $\Delta$, we have that $\Pr[Y_v = 1] \leq (1-p)^{1+\Delta}$. Thus $\mathbb{E}[|Y|] = \sum_v \mathbb{E}[Y_v] \leq (1-p)^{1+\Delta} n$.

So $\mathbb{E}[|X \cup Y|] \leq pn + (1-p)^{1+\Delta} n$. Since $(1-x) \leq e^{-x}$, we get that the expected size is at most $pn + e^{-p(1+\Delta)} n$. Set $p = \ln(1+\Delta)/(1+\Delta)$. Then the expected size of the output is $n(1+\ln(1+\Delta))/(1+\Delta)$.

As in the previous section, we can use Markov's inequality to get an algorithm that finds a dominating set of size close to $n(1+\ln(1+\Delta))/(1+\Delta)$.

### *Min-Cut*

We are given an undirected graph and want to partition the vertices into two non-empty sets $A, B$, such that the number of edges that cross from $A$ to $B$ is minimized. Consider the following simple algorithm:

**Input:** Undirected graph $G$
**Result:** A partition $A, B$
Repeatedly do the following as long as the graph has more than 2 vertices: pick a uniformly edge that connects two distinct vertices and merge them. Output the partition that corresponds to the two vertices that are left at the end.

We shall show that this algorithm find the minimum cut with non-negligible probability. Suppose the min-cut has $k$ edges. Then the algorithm finds this min-cut if and only if none of these $k$ edges are picked to do a merge by the algorithm.

Observe that every vertex must have at least $k$ neighbors, or the vertex by itself would give a smaller cut. This means that the graph has at least $nk/2$ edges. The probability that we pick one of the $k$

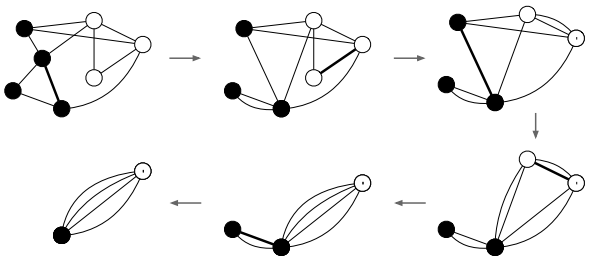

Figure 1: An execution of the randomized algorithm for Min-cut. This execution finds the cut indicated by black vertices.

edges of the min-cut for the merge is at most $k/(nk/2) \leq 2/n$. Assuming that one of these edges is not picked, then again we must have that every vertex in the new graph has degree at least $k$, or we would get a smaller min-cut in the original graph. Continuing in this way, we get that the probability that the $k$ edges of the min-cut are never picked is at least

$$\begin{aligned}
&\left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{4}\right)\left(1-\frac{2}{3}\right)\\
&=\left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right)\\
&=\frac{2}{n(n-1)}.
\end{aligned}$$

This is a small probability, but imagine we just repeat the above algorithm $t$ times and then output the best cut that we find. Then the probability that every run of the algorithm does not find the min-cut is at most $\left(1-\frac{2}{n(n-1)}\right)^t \leq e^{-\frac{2t}{n(n-1)}}$. If we set $t \gg n(n-1)$, this probability is extremely small.