CSE548, AMS542: Analysis of Algorithms, Spring 2015 **Date:** Apr 12

Homework #3

(Due: Apr 28)

Task 1. [90 Points] A Recursive Randomized Min-Cut Algorithm

Consider the randomized min-cut algorithm we saw in the class that returns a min-cut with probability $\geq 1 - \frac{1}{e}$. Given a connected undirected multigraph with n vertices, the strategy is to run the following algorithm $\frac{n^2}{2}$ times and return the smallest cut identified by those runs. Each run uses an algorithm that starts with the original n-vertex graph and performs a sequence of $n-2$ edge contractions. Each contraction is performed on an edge chosen uniformly at random from the current set of edges. A contraction step contracts the two endpoints of the given edge into a single vertex and removes all edges between them, but retains all other edges (and thus leading to a multigraph). After $n-2$ contraction steps only 2 vertices remain, and all edges between those two vertices are returned as a potential min-cut.

(*a*) [**10 Points**] Argue that each contraction step can be implemented to run in $\mathcal{O}(n)$ time, and thus the randomized min-cut algorithm described above takes $\mathcal{O}\left(n^4 \log n\right)$ time to return a min-cut with high probability.

```
RECURSIVE-RANDOMIZED-MIN-CUT( $G$, $a$, $b$, $c$ )
(Input is an undirected multigraph $G$ with $n$ vertices, two integer constants $a > 0$ and $c > 0$, and one real-valued
constant $b > 1$. Output is a cut of $G$.)

1. if $n \leq c$ then
2.     $C \leftarrow$ a min-cut of $G$ found using brute force (exhaustive) search
3. else
4.     $G' \leftarrow$ multigraph obtained by applying $n - \lceil \frac{n}{b} \rceil$ random contraction steps on $G$
5.     for $i \leftarrow 1$ to $a$ do
6.         $C' \leftarrow$ RECURSIVE-RANDOMIZED-MIN-CUT( $G'$, $a$, $b$, $c$ )
7.         if $i = 1$ or $|C'| < |C|$ then $C \leftarrow C'$
8. return $C$
```

Figure 1: A recursive randomized min-cut algorithm.

Now consider the recursive algorithm given in Figure 1.

(*b*) [**10 Points**] Let $T(n)$ be the running time of the algorithm on a multigraph with n vertices. Write a recurrence relation describing $T(n)$ and solve it.

(*c*) [**10 Points**] Let $P(n)$ be the probability that the algorithm returns a min-cut when run on a multigraph with n vertices. Write a recurrence relation describing $P(n)$.

(*d*) [**25 Points**] Let $P_\alpha(n)$ be the value of $P(n)$ when $a = b^2 = \alpha$, where $\alpha > 1$ is an integer. Solve the recurrence for $P_2(n)$.

(*e*) [**15 Points**] Show that $\lim_{n\to\infty} \frac{P_{\alpha+1}(n)}{P_\alpha(n)} = 1 - \frac{1}{\alpha^2}$ for any integer $\alpha > 1$.

(*f*) [**5 Points**] Use your result from part (*e*) to show that $\lim_{n\to\infty} \frac{P_\alpha(n)}{P_2(n)} = \frac{\alpha}{2(\alpha-1)}$ holds for any integer $\alpha > 1$.

(*g*) [**15 Points**] How would you use the algorithm in Figure 1 to obtain a min-cut of an n-node multigraph w.h.p. in n, assuming $a = b^2$? What is the running time of the resulting algorithm?

Task 2. [40 Points] Load Balancing

Consider a set of $n \gg 0$ of identical processors each identified with a unique integer between 0 and $n-1$. The processors execute jobs in rounds. In every round each processor receives at most 1 job to execute, and all jobs arrive at the same time right after the start of the round. Let us assume for simplicity that execution of every job takes exactly the same (very large) amount of time. Now for load balancing the processors use the following approach. At the start of every round $i \geq 0$, processor ($i \mod n$) chooses an integer r_i uniformly at random from $[0, n-1]$, and distributes it to all other processors. Then every processor j sends its job (if any) to processor $((j + r_i) \mod n)$ for execution instead of executing the job itself. We will prove high probability bounds on the maximum number of jobs executed by a processor in n such rounds. Clearly, a processor may end up executing n jobs in the worst case.

Let m be the total number of jobs received by all processors in n rounds. Then prove the following two statements.

(*a*) [**20 Points**] If $m \geq n \ln n$, then w.h.p. in n no processor executes more than $\frac{5m}{n}$ jobs.

(*b*) [**20 Points**] If $m < n \ln n$, then w.h.p. in n no processor executes more than $\frac{m}{n} + 4 \ln n$ jobs.

Task 3. [50 Points] NAND Flash

I have n data items stored on a NAND flash drive, and I have a program that accesses each data item at most once every time it is run. I will have to run the program n times, and count the total number of times each data item is accessed by the program across all runs. I intend to use an array $C[1..n]$ of counters (initialized to 0's) for this purpose. This counter array is also stored on the flash drive (too large for the RAM).

I have heard that flash drives have limited *write endurance*. Repeated writes to a data block degrades the oxide layer isolating its gates, and after a certain number of writes the block becomes *bad* or unsuitable for data storage. Though the device's own erase algorithm tries to mitigate the problem, I have decided to be a bit more careful.

My plan is to keep multiple copies of the counter array, and try to distribute the writes to the copies as evenly as possible. The hope is to reduce the number of writes to each block by dividing them among multiple blocks. Every time the program is run it will choose one copy of the counter array, and perform all writes on that copy only. After the n runs of the program, I will add up the counts in all copies to get the final count for each data item. Observe that since the program may access different sets of data items in different runs and I am directing all writes of a run to a single copy of the counter array, distributing the writes to every counter evenly among different copies may not even be possible. After my many failed attempts to find a deterministic algorithm that always gives good (not necessarily perfect) solutions, a friend of mine advised me to choose a copy of the counter array independently and uniformly at random in every run.

In this task we consider a simplified scenario. Suppose we have only two copies C_1 and C_2 of the counter array, and every run of the program chooses one of those two copies independently and uniformly at random. For each $i \in [1, n]$. let $w_i = C_1[i] + C_2[i]$ and $\delta_i = C_1[i] - C_2[i]$. Also let $\Delta = \max_{1 \leq i \leq n} |\delta_i|$. Observe that the lower the value of Δ is, the better (i.e., more balanced) the distribution of writes (between C_1 and C_2) is.

Now answer the following questions. In parts (a)–(b) below we consider any given index $i \in [1, n]$.

(a) [**10 Points**] Prove that expected value of $C_1[i]$ is $\frac{w_i}{2}$.

(b) [**20 Points**] Prove that $|\delta_i| < 4\sqrt{n \ln n}$ holds w.h.p. in n, provided $w_i > 4\sqrt{n \ln n}$.

(c) [**20 Points**] Prove that $\Delta < 4\sqrt{n \ln n}$ holds w.h.p. in n.