CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2023**

Homework 2

Due: Monday, October 10, 2023

Problem 1. We reexamine the balls-and-bins experiment in this question, focusing on aspects other than the maximum load. Suppose we are throwing m balls into n bins where each ball chooses one of the bins independently and uniformly at random.

(*a*) Prove that it is sufficient and necessary for m to be $\Theta(n \log n)$ so that with constant probability, every bin has at least one ball inside it. **(12.5 points)**

(*b*) Find the sufficient and necessary asymptotic value for m to be, so that with constant probability, at least one bin has two or more balls inside it. **(12.5 points)**

Problem 2. For any $\varepsilon \in (0, 1/4)$, a $(1 \pm \varepsilon)$-**cut sparsifier** of an undirected unweighted graph $G = (V, E)$ is a *weighted* spanning subgraph $H = (V, E_H)$ of G with weights $w : E_H \to \mathbb{R}^+$ such that for *every* cut $S \subseteq V$,

$$(1 - \varepsilon) \cdot |\delta_G(S)| \leqslant w(\delta_H(S)) \leqslant (1 + \varepsilon) \cdot |\delta_G(S)|;$$

here, $\delta_G(S)$ is the set of edges in the cut S in G and $w(\delta_H(S))$ denotes the *weight* of the cut S in H. In other words, the weight of every cut in H is a $(1 \pm \varepsilon)$-approximation of the size of the same cut in G.

We are generally interested in constructing cut sparsifiers that are *sparse*, i.e., has few edges (otherwise, we could have taken $H = G$ with weight 1 everywhere). In this question, we see a simple (but not that efficient) way of constructing a cut sparsifier using random sampling.

Let λ denote the minimum cut value of G. Suppose we sample each edge in G with probability

$$p := \frac{100 \log n}{\varepsilon^2 \cdot \lambda},$$

to obtain the graph H and set the weights of all sampled edges to be $1/p$. Prove that with high probability, H is a $(1 \pm \varepsilon)$-cut sparsifier of G with

$$O(\frac{m \log n}{\varepsilon^2 \cdot \lambda})$$

many edges. Note that for "large" values of λ, this approach indeed sparsifies the graph. **(25 points)**

Note: There is a fundamental graph structural result that is crucial for solving this problem: in any graph $G = (V, E)$ with minimum cut λ, and for any $\alpha \geqslant 1$, the number of cuts with size at most $\alpha \cdot \lambda$ is $O(n^{2\alpha})$. This is a generalization of the bound we proved earlier in the course that the number of minimum cuts is $O(n^2)$ (the aforementioned result extends this from exact minimum cuts to approximate ones). You can prove this generalization along the same lines of the proof for exact minimum cuts, but for this question, you can just directly use this fact without a proof.

Problem 3. In this question, we prove two other simple results from *random graph theory* (for a slightly different family of random graphs than the ones studied in the lecture). Let $G = (V, E)$ be a graph on n vertices obtained by adding an edge between each pair of vertices independently and with probability

$$p := \frac{100 \log n}{n}.$$

(a) Prove that with high probability G is connected. **(10 points)**

(b) Prove that with high probability the *chromatic number* of G is $\Omega(\frac{\log n}{\log \log n})$. Recall that the chromatic number is the minimum number of colors we can assign to the vertices so that no edge receives the same color on both its endpoints. **(15 points)**

Hint: Try to upper bound the size of the largest *independent set* in G first and then deterministically relate that to the chromatic number.

Problem 4. In this question, we design another simple algorithm for MSTs with runtime better than the classical algorithms (although not as good as the advanced ones we studied). Recall the following two facts:

- Each round of Boruvka's algorithm takes $O(m)$ time and reduces the number of vertices by at least a half.
- Prim's algorithm can be implemented in $O(m + n \log n)$ time using Fibonacci heaps.

Combine these two algorithms in a careful way to obtain an $O(m \log \log n)$ time algorithm for MSTs.

(25 points)

Problem 5 (**Extra Credit**)**.** Design a *deterministic* $O(m)$ time algorithm for finding MST of a given *planar* graph. A planar graph is a graph that can be drawn on a surface so that no two edges cross each other.

(+10 points)

Hint: This is a pretty simple question. Think about (or search) how many edges are in a planar graph? And, what happens if you contract an edge in a planar graph?