6.856 — Randomized Algorithms

David Karger

Handout #10, Mar. 16th, 2005 — Homework 6, Due 3/30

1. MR 12.12. Consider the following alternative parallel algorithm for maximal independent set. In a phase, a (non maximal) independent set S is output, and all of its neighbors are deleted. To find S, randomly permute the vertices, then mark each vertex that precedes all of its neighbors in the permutation. It may help to use the following equivalent formulation: assign to each vertex a uniformly distributed random weight from the range $\{1, ..., n^4\}$. Mark all vertices, then in parallel unmark the larger-weight endpoint of each edge. The set of vertices that remain marked is S. Continue until the graph is empty.

   (a) Argue that the set of vertices output over all phases is a maximal independent set.

   (b) Assuming that v has degree d during a phase, what is the probability that vertex v stays marked?

   (c) Show that this approach yields an **RNC** algorithm for maximal independent set.

2. Basic Sampling Tricks

   (a) Amplification for sampling. Suppose you have an estimation algorithm that will find a $(1 \pm \epsilon)$ approximation to the correct value with probability 3/4. Show that you can reduce the failure probability exponentially fast from 1/4 to any desired δ by performing some number k of estimation experiments and taking the median value returned. Give the smallest upper bound you can on k as a function of δ. This shows the $\log(1/\delta)$ term in $\mu_{\epsilon,\delta}$ is natural.

   (b) Error bound for sampling. Suppose you are able to sample from some probability distribution whose standard deviation is less than its mean. Give an (ϵ, δ)-FPRAS for estimating the mean of this distribution (to within $1 \pm \epsilon$ with probability $1 - \delta$) with a number of samples polynomial in $1/\epsilon$ and $\log 1/\delta$. **Hint:** Consider the sum of n independent samples from the distribution and determine its mean and variance. Bound the probability that this sum deviates greatly from its mean. Now use the previous part. This shows the $1/\epsilon^2$ term in $\mu_{\epsilon,\delta}$ is natural.

3. **This problem should be done without collaboration** Suppose you are given a directed graph with n vertices and m unit-length edges. Consider the problem of estimating the number of vertices within distance d of each vertex. Give a fully polynomial (ϵ, δ) approximation scheme that solves this problem simultaneously for all vertices for any fixed d. Your running time should be $O((m+n)/\epsilon^2 \log 1/\delta))$ to within polylogarithmic factors.

4. Based on MR 11.2. In class we gave an FPRAS for DNF counting that evaluated the entire formula about m times. Here we develop a faster algorithm. Consider the following variant of the DNF counting algorithm from class. For the t-th trial, pick a satisfying assignment a uniformly at random from the *disjoint* union of satisfying assignments, just as described in class. But now, instead of checking whether a is the "first" copy of itself, try the following. Let N be the number of assignments in the disjoint union. Let c_a be the number of clauses that a satisfies. Define $X_t = 1/c_a$.

   (a) Prove that $N \cdot E[X_t]$ is the number of satisfying assignments to the DNF formula

   (b) Prove that $O(m\mu_{\epsilon\delta})$ trials (and computation of the resulting $\sum X_t$) suffice to DNF-count to within $(1 \pm \epsilon)$ with probability $1 - \delta$. **Hint:** use the Chernoff bound generalization from MR 4.7.

   (c) Once a has been chosen in the previous subproblem, give an algorithm for quickly estimating c_a to within $(1 \pm \epsilon)$. Argue that this is sufficient to give us the $(1 \pm \epsilon)$ approximation for DNF-counting.

   (d) Using the new scheme from the previous subproblem, analyze the expected number of clauses you need to evaluate in the course of the algorithm. Assuming all clauses are the same size, what is the actual running time of the scheme in terms of basic operations?

   (e) (Optional) Justify the assumption that all clauses are the same size to within a logarithmic factor, thus extending the runtime analysis to arbitrary formulae.

5. **Optional**. Suppose you wish to implement a counter that supports an "increment" operation and a "query current value" operation. To handle values up to n exactly you need $\log n$ bits. Suppose that you are willing to tolerate relative error ϵ in your queries. Design a scheme that uses $O(\log \log n)$ bits. **Hint: increment probabilistically.**