Advanced Algorithm Analysis

Project #3

Introduction: Randomized Algorithms.

In this unit of the course we've been talking about algorithms which don't give the correct answer every time. "What good is an algorithm that is sometimes wrong," you ask? *No good at all,* unless you understand some key things about the probability distribution of the algorithm's answers. For example, we discussed the Miller-Rabin primality test. Part of this test involves choosing a random integer A and doing some math with it, then outputting either "prime" or "composite." Thanks to the hard work of some brilliant mathematicians over the years, we know the following facts:

- The test *always* says "prime" when the input is prime, no matter what A you chose (the false negative rate is 0).
- When the input is composite, the test says "prime" for some choices of A. However, this false positive happens for no more than 25% of the possible values of A. That is, a single run of the algorithm has a false positive rate no greater than 25%.

Due to these two facts, it's easy to use the test to create an algorithm to check if a number n is prime: if the test ever returns "composite," you're done: the number is composite. If it returns "prime," run it again a few times. If you get "prime" K times in a row, then you know that the probability that the number is composite is never more than $1/4^K$, which is a very very small number even for a moderate K.

Note that not just any algorithm which makes randomized choices is a good algorithm. For example, if BOTH the false positive and false negative rates are greater than 0, you need to be more careful. If you don't know a bound on either of these rates, or if the rates are 1/2 or worse, then you can't use the algorithm to make any guarantees with any confidence.

Also, the problem is a bit trickier if the algorithm is an optimization algorithm like Karger's algorithm. One feature which makes Karger's algorithm good is that it is guaranteed *never* to underestimate the size of the min-cut. That way, you can always have a simple policy like "after you've run the algorithm K times, output the smallest value you've seen over all the runs." If it ever were to underestimate the answer, that policy wouldn't work and the algorithm wouldn't give you any guaranteed results.

This project: Your assignment

In collaboration with your work group and with the professor, write 2 problem statements ("deliverables"). Everybody in a work group should complete the same 2 deliverables; this will give provide you with flexibility but also closer working partners as you go through the semester. Each of the deliverables should address a well-posed scientific question and explore some aspect of randomized algorithms. There are lots and lots of options for these, be creative! Here are a couple to get you thinking:

- Implement the Miller-Rabin test. Do some research or perform some experiments to try to find composite numbers that the M-R test lies about! Then empirically compute the error rate on those numbers. Can you come anywhere close to 1/4?
- Implement Karger's Algorithm for finding the min-cut of a graph, do some experiments to find out how many times you typically need to run it before it finds the right answer!
- Do some experiments with stochastic gradient descent (SGD) for a problem like least-squares regression.
- Do a broader literature review of randomized algorithms and come back with some conclusions!

While the project is active, your team will give regular weekly updates in class and I may occasionally ask for demos of the project. When the project is finished, each person in your group will submit a written report which summarizes your findings, and we will supplement the weekly work group update with brief verbal reports from each person.