Advanced Algorithms Course. Lecture Notes. Part 7

Repeat Until Success

An important "algorithm" is to repeat a random experiment until success: Suppose that we have a $0,1$-valued random variable that attains value 1 with probability p. We observe this variable many times independently, until result 1 appears for the first time. What is the expected number of iterations needed? Intuitively one would think $1/p$, but intuition is often misleading, therefore we'd better derive this result by calculation. Although this is still a basic exercise, a strict formal treatment would already be a bit laborious: Our probability space is the Cartesian product of infinitely many copies of a probability space with two events. However we may skip some technicalities and think in a semi-formal way. Let E_i be the event that the ith iteration is successful. Then $Pr(E_i) = (1-p)^{i-1}p$. Note that the first $i-1$ iterations have failed, and the probabilities can be multiplied, because the trials are independent. Hence our expected value is

$$\sum_{i=1}^{\infty} Pr(E_i) \cdot i = \sum_{i=1}^{\infty} (1-p)^{i-1} pi.$$

Now some standard algebra (that we omit here) confirms the result $1/p$.

Global Minimum Cut Revisited

In a graph $G = (V, E)$ with n nodes and m edges we wish to find a global min-cut (A, B), that is, a partitioning $V = A \cup B$ such that the number of cut edges (those edges between A and B) is minimized. Motivations include the assessment of reliability of networks, finding clusters in graphs, and efficient hierarchical computation of distances in graphs.

We can easily reduce the problem to Minimum Cut, by trying all possible pairs of sources and sinks $s, t \in V$. But since flow and cut algorithms are somewhat sophisticated, it may be pleasant to see an extremely simple randomized algorithm that solves the Global Min-Cut problem as well. However this comes with a price: Success is no longer guaranteed. We will get a correct solution "only" with high probability.

For simplicity we discuss only the basic randomized algorithm for Global Min-Cut, although faster algorithms are known. In the following we have to allow graphs with parallel (multiple) edges. The algorithm works as follows. In every step, choose an edge $e = (u, v)$ at random and contract it. Contraction means: shrink e, identify u, v (merge them into a new vertex), and delete all edges that have been parallel to e (they would be loops at the new vertex). Iterate this step until two nodes remain. This two-node graph represents a cut, in the obvious sense. The whole procedure is repeated a certain number of times from scratch, and finally we output the smallest cut found in this way.

It may seem that this algorithm has nothing to do with the problem. It just repeatedly contracts random edges. However, the intuition is that a small cut has a chance not to be affected by these random contractions, thus being preserved in the end. Still, the analysis which has to confirm this intuition is not so obvious. It uses a clever combination of several elementary tools from probability theory.

Consider any global min-cut (A, B). Let F denote the set of its cut edges, and $k := |F|$. After j steps of the algorithm, clearly the contracted graph has $n - j$ nodes. Moreover, every node has degree at least k, since otherwise the node and its complement set would already form a global min-cut smaller than k, a contradiction. Hence at least $k(n - j)/2$ edges still exist after j steps. Therefore, the probability that unfortunately some of the k edges in F is contracted in the next step is at most $2/(n - j)$. That means, our specific cut (A, B) is returned with probability at least

$$\prod_{j=0}^{n-3}(1 - 2/(n - j)) = \prod_{j=0}^{n-3}((n - j - 2)/(n - j)) = 2/n(n - 1)$$

after the contraction procedure. (However, think carefully: Why is it correct to multiply the probabilities, although the events "step j avoids to select an edge from F" are certainly not independent?) This is a small probability, but we repeat this $O(m)$-time contraction procedure sufficiently often: Each run fails with probability $1-2/n(n-1)$, but a simple calculation shows that some of $O(n^2)$ runs succeeds, subject to a small constant failure probability. We can make this failure probability arbitrarily small by increasing the hidden constant factor in $O(n^2)$. (Note the superficial similarity to approximation schemes.)

About Randomized Algorithms in General

Randomized algorithms should not be confused with average-case analysis. All randomness is with the algorithms, while nothing is assumed about the probability distribution of inputs. The analysis results (expected time, probability of a correct solution, etc.) hold for every fixed instance, not only averaged on all instances.

The Global Minimum Cut algorithm is an example of a **Monte Carlo algorithm**. These are algorithms that run in polynomial time in the worst case, but whose result can be wrong with some small probability. However this failure probability can be reduced exponentially by repeated runs. The latter technique is called **amplification**. By way of contrast, a **Las Vegas algorithm** gives always the correct result, but only an expected time bound can be shown, and the running time can be much higher in the worst case.