CSCI3390-Lecture 19: Probabilistic Algorithms

A probabilistic algorithm is one that is allowed to make random choices–that is, flip coins, and take different execution paths depending on whether the outcome of the coin flip is heads or tails..

Can using randomness make hard problems easier? In a sense, the answer is 'yes', as we'll show with an example below. In a deeper sense, this is an open question: If there is a polynomial-time probabilistic algorithm for a problem, can it be 'derandomized' to a polynomial-time deterministic algorithm? (The technical description of this question is 'Is $\mathbf{P} = \mathbf{BPP}$'? The most common conjecture is 'yes'.)

1 One-sided and two-sided error.

As usual, we will stick to decision problems. The answer that a probabilistic algorithm gives might be erroneous. If the algorithm always answers 'No' when the correct answer is 'No', but sometimes answers 'No' when the correct answer is 'Yes', then we say that there is a *one-sided error*. Let p be the probability that on a given 'Yes' instance, the algorithm answers 'No.' As long as $p < 1$, then we can *amplify* the probability of correctness by running the algorithm repeatedly on the same input, until we either get a 'Yes' answer or can be sure that the probability of error is negligibly small. For example, if $p = 3/4$, then on a single run on a 'Yes' instance, the algorithm is more likely than not to give the incorrect answer 'No'. But if we run the algorithm 50 times on the same instance, the probability of getting 'No' 50 times in a row is

$$(3/4)^{50} < 10^{-6},$$

that is, less than one in a million. If we get 'No' 200 times in a row, then the probability is so small that we can be as confident as possible that this answer

is correct—the probability of an error is far less than the probability that some hardware malfunction changed the answer.

Two-sided error occurs when there is a nonzero probability of both a false negative (as above) and a false positive. If both of these probabilities are less than some value $c < 1/2$, , then we can similarly perform the error amplification, although the calculation is more complicated. In practice, you are more likely to see algorithms with one-sided error.

2 A probabilistic algorithm for 3-SAT

We'll exhibit a one-sided probabilistic algorithm for 3-SAT whose running time for a formula with n variables is $O(1.4^n)$. Since the brute-force algorithm takes time proportional to 2^n, and since

$$\lim_{n\to\infty} \frac{1.4^n}{2^n} = 0,$$

approaching 0 exponentially fast, this algorithm is a marked improvement, even though it still requires exponential time. We will give a somewhat less careful analysis that still gives a running time of $O(1.8^n)$, still a marked improvement.

Here is the core of the algorithm: Given a propositional formula ϕ in 3-CNF, choose a random assignment α to the variables in ϕ. Then, repeat the following S times (where the value of S is to be determined later): If α is a satisfying assignment for ϕ, return True. Otherwise, find a clause

$$\ell_1 \vee \ell_2 \vee \ell_3$$

satisfied by α. Pick one of the three literals ℓ_i at random, and flip the value of the associated variable, giving a new assignment α', and set α to α'.

The algorithm makes $n + \log_2 3 \cdot S$ coin flips and has running time $p(n) \cdot S$, where p is a polynomial—this is because we can check if an assignment is satisfying, and find an unsatisfied clause if one exists, in time polynomial in the size of the formula. (Observe that for a 3-CNF formula, the size of the formula is polynomial in the number of variables.) The error is one-sided: We never say that a formula is satisfiable if it is not, but we there is a strong chance that we will not find a satisfying assignment.

Let us estimate the probability of this kind of error. Suppose ϕ is satisfiable. Then there must be at least one satisfying assignment β. If α is another assignment, let us denote by $d(\alpha, \beta)$ the number of variables at which α and β differ.

This is the so-called *Hamming distance* between the two n-bit strings α and β. Each time we choose one of the three literals at random, we get a new assignment α'. The idea is that we would like α' to be a better approximation to β than α is. Since α is wrong on at least one of these three literals, we have at least a 1/3 probability of guessing such a literal, and thus

$$\text{Prob}(d(\alpha', \beta) \leq d(\alpha, \beta)) \geq 1/3.$$

The probability that we get S good guesses in a row is therefore at lest 3^{-S}. Now suppose α is our initial guess, which we got by flipping n coins. At least half of the n-bit strings are within Hamming distance $n/2$ of β, so the probability that α is no more than $n/2$ is at least 1/2. Therefore, if we set $S = n/2$, we have a probability of at least

$$q = \frac{1}{2 \cdot 3^{\frac{n}{2}}}$$

of finding the satisfying assignment. Of course this is a very crude estimate: there may be more than one satisfying assignment; there may be more than one literal in the selected clause that is wrong; and we really don't need a string of all correct guesses to hit β. But this is a lower bound for the probability.

Of course this is a very tiny probability of success. But let's repeat it T times, and see how big T must be to make the probability of success at least one half. The probability of T successive failures is no more than

$$\begin{aligned} (1-q)^T &\approx e^{-qT} \\ &= \exp(\frac{-T}{2 \cdot 3^{\frac{n}{2}}}). \end{aligned}$$

Setting this equal to 1/2 gives

$$T = 2 \ln 2 \cdot (\sqrt{3})^n \approx 2 \ln 2 \cdot 1.73^n.$$

So if we want an error probability of less than one millionth, we set T to 20 times this amount. Since each repetition takes time $p(n)S = n \cdot p(n)/2$, the overall running time is $O(1.8^n)$. A more careful analysis gives the smaller bound mentioned above.