1 The Probabilistic Method

The "probabilistic method" refers to an approach to showing something exists by setting up a probabilistic experiment and showing that the experiment has a nonzero chance of producing what you want. Or showing that something of size at least k exists by setting up a probabilistic experiment and showing that the expected value under this experiment of the size of the thing you get is at least k.

Here is an example.

Theorem 1 *Suppose F is a k-CNF formula with less than 2^k clauses, in which every clause has size exactly k (and you are not allowed to repeat variables inside a clause). Then F must be satisfiable.*

Proof: Consider a random assignment x The probability it fails to satisfy any given clause is $1/2^k$. So the probability it fails to satisfy f is at most (# clauses)$/2^k < 1$. This means a satisfying assignment must exist. ■

Here is another example.

Theorem 2 *Suppose G is an undirected graph with m edges. Then G must contain a cut (a partitioning of the vertices into two sets A and B) of size at least $m/2$ (at least $m/2$ edges cross between A and B).*

Proof: Just put the vertices at random into A or B. For any given edge, the probability it crosses the cut is exactly $1/2$ (with probability $1/4$ both endpoints are in A and with probability $1/4$ both endpoints are in B). So the expected number of edges crossing the cut is exactly $m/2$. ■

2 DeMillo-Lipton-Schwartz-Zippel

Now we're going to turn to randomized algorithms. Most problems that have polynomial-time randomized algorithms also have known polynomial-time deterministic algorithms, but there are a few hold-outs.

Here is one: it is like an algebraic version of the SAT problem.

Say p is an n-variable polynomial, given in some form that's easy to evaluate, like

$$p(x) = (x_1 + 3x_2 - x_3)(3x_1 + x_4 - 1)...,$$

over a field F with $|F| > 2d$, where d is the degree of the polynomial. (Note, $x_1x_2x_3^2$ has degree 4).

We want to know if p is identically 0 or if on the other hand there exists an assignment x (where each $x_i \in F$) for which $p(x) \neq 0$.

There is no known polynomial-time deterministic algorithm for this problem. But, one thing we *can* do is plug in random values and see if we get 0. The Demillo-Lipton-Schwartz-Zippel theorem says that if p is not identically 0, then this is unlikely to happen just by chance. Note that this also gives us a way to test if two polynomials p and q are identical. We can plug in a random x and see if $p(x) - q(x) = 0$. Here is the theorem we will prove:

Theorem 3 (DeMillo-Lipton-Schwartz-Zippel) *Say $p(x_1, ..., x_n)$ is a degree-d polynomial over some field F, and not identically 0. For any finite $S \subseteq F$, if we pick $r_1, ..., r_n$ at random in S,* $\Pr[p(r_1, ..., r_n) = 0] \leq d/|S|$.

Proof: We'll prove this by induction on n (the number of variables). For the case $n = 1$, we have at most d roots, so we satisfy the condition.

For the general case, there could be a large number of roots (e.g., over the reals there are an infinite number of roots of $x_1 x_2$). However, we can prove the theorem as follows. Say the maximum degree of x_n is i. So, we have

$$p(x) = x_n^i p_i(x_1, ..., x_{n-1}) + x_n^{i-1} p_{i-1}(x_1, ..., x_{n-1}) + ...,$$

where p_i is not identically 0 and has degree at most $d - i$.

Now, pick $x_1, ..., x_{n-1}$ at random. The probability we've zeroed out p_i is some value $\alpha \leq (d-i)/|S|$ by induction. Assuming this doesn't happen, we have a degree-i polynomial in 1 variable, so when we pick x_n randomly, the chance we get 0 is at most $i/|S|$. Overall, the failure probability is at most $\alpha + (1 - \alpha)i/|S| \leq d/|S| - i/|S| + (1 - \alpha)i/|S| < d/|S|$. ■

3 Perfect Matchings in General Graphs

Now, here is a nice application of the above result. Say we have an undirected graph and want to find a perfect matching. We saw how to do this for *bipartite* graphs using network flow. But what about for general graphs? Here is a neat randomized algorithm that works for general graphs. This will solve the decision problem (does G have a perfect matching) and then one can use that to actually find the matching. Note: there exist alternative deterministic algorithms, but this is the simplest polynomial-time algorithm I know of.

First, given a graph G, think of it as a directed graph G' where each (undirected) edge in G is replaced by two directed edges, one in each direction. G has a perfect matching iff G' has a cycle cover (a collection of disjoint cycles that cover all the vertices) in which all cycles have even length.

Now, given G, we create what's called the "Tutte matrix" M. For every edge (i, j), $i < j$, we put variable x_{ij} in entry ij (above the diagonal) and $-x_{ij}$ in entry ji (below the diagonal). The rest are 0. Can think of this like G', but when we replace an undirected edge by two directed ones, we are giving one a positive sign and one a negative sign.

Now, consider the determinant of this matrix $det(M)$. This is a polynomial of degree at most n, and the claim is it is identically 0 iff G has no perfect matchings. In particular, (1) every permutation that gives us a term of this polynomial is a directed cycle cover (every vertex has exactly one out-edge since there is one variable per row, and every vertex has exactly one in-edge since there is one variable per column), (2) two directed cycle covers correspond to the same term if and only if they have the same edges (but perhaps going around some cycles in opposite directions), and (3) if you sum over all cycle covers corresponding to the same term you will get zero if and only if there is an odd-length cycle in the cover. [Go through an example explaining why – also uses the fact that the inverse of a permutation has the same sign]. Given this fact, we can tell if G has a perfect matching by checking if this polynomial is identically 0. We do this by just plugging in random values for the variables and then calculating the determinant (which we can do efficiently).