# Classes with Randomness.

CS254
Chris Pollett
Nov. 1, 2006.

# Outline

- Randomized Algorithms
- Randomized Complexity classes

# Randomized Algorithms

- We now begin to investigate the power of the Turing Machines which have the ability to flip coins.
- To begin we consider some randomized algorithms for some common computational problems.

# Determinants

- Recall the determinant of a matrix A is given by

$$\det A = \sum_{\pi} \sigma(\pi) \prod_{i=1}^{n} A_{i,\pi(i)}$$

- Here $\pi$ is a permutation of 1,..,n and $\sigma(\pi)$ is 1 if $\pi$ is a product of an even number of transpositions and -1 otherwise.
- For example, if A is the 2x2 matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ its determinant is ad-bc.
- Determinants are useful for many things: Computing volumes, inverting matrices, etc.
- To compute a determinant of an n x n matrix one typically uses Gaussian elimination to convert the matrix into an upper triangular form. The determinant then becomes the product of the diagonals.
- So the determinant can be computed in polynomial time.

# Determinants and Matchings

- Given a bipartite graph $G=(V \cup U, E)$ we can compute the determinant of its adjacency matrix $A^G$.
- For this matrix $a_{ij}$ entry is $x_{ij}$ (a symbolic variable) if there is an edge from the *i*th element of V to the *j*th element of U. It is 0 otherwise.
- The only nonzero terms in the determinant correspond to perfect matchings in G.
- Since all of the elements in V and U appear at once, the terms we get in this symbolic matrix don't cancel.
- So G has a matching iff this symbolic determinant is non-zero.
- We would thus like to be able to figure out if a symbolic determinant is identically zero.
- The idea we'll use is to fill in random numbers for the variables and see if we get zero.

# Lemma

Let $\pi(x_1,.., x_m)$ be a polynomial, not identically zero, in m variables each of degree at most d. Let $M > 0$ be an integer.

Then the number of m-tuples $(x_1,.., x_m) \in \{0, 1\ldots, M-1\}^m$ such that $\pi(x_1,.., x_m) = 0$ is at most $mdM^{m-1}$.

**Proof.** By induction on m. When m=1, the lemma says that no polynomial of degree $\leq$d can have more than d roots which is just the Fundamental Theorem of Algebra. By induction suppose the result is true for m-1 variables. Suppose $\pi(x_1,.., x_m)$ is an m variable polynomial and it evaluates to 0 at some point in our domain. The highest degree coefficient of $x_m$ is then either 0 or not. This coefficient is a polynomial in just $x_1,.., x_{m-1}$ . If it is zero by our hypothesis, this can happen for at most one of $(m-1)dM^{m-2}$ places. Since in our domain we have M choices for $x_m$, the total number of places where the lead term is zero is at most $(m-1)dM^{m-1}$. The remaining terms in $\pi$ define a degree $\leq$d polynomial in $x_m$ so can have at most d roots for each combination of $x_1,.., x_{m-1}$. This gives at most $dM^{m-1}$ more roots. Adding these two estimates gives the result.

# A Perfect Matching Algorithm

1. Choose m random integers $i_1, \ldots, i_m$ between 0 and M=2m.
2. Compute the determinant A, det $A^G(i_1, \ldots, i_m)$ by Gaussian elimination.
3. If it is not 0 then reply G has a perfect matching
4. Otherwise reply it does not have a perfect matching.

- Notice this algorithm might give a false negative with probability less than half.
- By repeating the experiment multiple times we can reduce the probability to as small as we want.
- We already had an algorithm for matching; nevertheless, the above solves the more general problem of checking when a symbolic determinant is 0, for which no deterministic p-time algorithm is known.

# Random Walks for SAT

- Randomized algorithms can also be used in the context of SAT. Consider:

1. Start with any truth assignment T, and repeat the following r times:
   - If there is no unsatisfied clause output “Satisfiable”, halt.
   - Otherwise, take any unsatisfied clause; pick any of its literals at random and flip its value
2. After r repetitions reply “formula is probably unsatisfiable”

- The above algorithm is called a “random walk” algorithm -- changing a variables value can be viewed as taking a step on the boolean hypercube of truth assignments.
- If we choose r big enough this algorithm is likely to succeed in finding a truth assignment if there is one.
- By the coupon collector problem if $r = 2^n n$ we can expect to have tried all possible truth assignments.
- This is probably a big overestimate. What is a reasonable r however? To be continued next day…