2MMD30: Graphs and Algorithms	Lecture 10	Date: 16/3/2016
Probabilistic Algorithms		Instructor: Nikhil Bansal

In this chapter, we will look at some algorithms based on probability.

1 The Drunk Random Walk

The drunk random walk is the following. We start at the origin at time $t = 0$. At each time $t+1$, we move randomly either one step to the right or one step to the left with probability $1/2$. Formally, let X_t denote the (random) position at time t. Then we have

$$X_{t+1} = X_t + 1 \text{ with prob. } 1/2 \qquad \text{or} \qquad X_t - 1 \text{ with prob. } 1/2$$

It is useful to know that after T steps, the expected distance of this walk from the origin is about $T^{1/2}$ (up to constants). That is, $E[|X_T|] \approx T^{1/2}$. There are several ways to see this. One way is to actually calculate the precise probabilities of being at position k after T steps. (Note that for this to happen there must be exactly $(T+k)/2$ steps to the right and $(T-k)/2$ steps to the left, so the probability is $2^{-T}\binom{T}{(T+k)/2}$, and then one can try to approximate these.

A more conceptual way to see this is the following: Consider the random variable $Y_t = X_t^2$. Let us consider how the expectation of Y_t evolves over time. Then,

$$E[Y_t] = E[X_t^2] = 1/2(X_{t-1}+1)^2 + 1/2(X_{t-1}-1)^2 = X_{t-1}^2 + 1 = Y_{t-1} + 1.$$

Thus, at each step, Y_t is expected to be 1 more than what it was at the previous time step. As $Y_0 = 0$ at the beginning, the above implies that $E[Y_t] = E[X_t^2] = t$.

From this one can read off several things. For example, the probability that one is more than $10\sqrt{T}$ steps away after time T is at most $1/100$. This follows by applying Markov's inequality. As

$$Pr[|X_t| \geq 10\sqrt{t}] = Pr[Y_t \geq 100t] = Pr[Y_t \geq 100E[Y_t] \leq 1/100$$

Let us note the following concrete fact which will be useful for the application later. Consider the random walk on points $0, \ldots, n$. If the walk is at position $i \neq \{0, n\}$ at time t, it randomly moves to $i+1$ or $i-1$ with probability $1/2$ at time $t+1$. If it is at 0, it moves to 1. We are interested in the following: If we start some position i, i.e. $X_0 = i$, then what is the expected time until the walk terminates at n.

Theorem 1 *Let T_i denote the expected number of time steps for a walk starting at i to terminate at n. Then $T_i = n^2 - i^2$.*

Proof: Note that T_i satisfies

$$T_i = ((1/2)T_{i-1} + (1/2)T_{i+1}) + 1$$

$$T_0 = 1 + T_1$$

$$T_n = 0$$

Now show that $T_i = n^2 - i^2$ satisfies these conditions. Or you can also determine T_i systematically, by considering the variables $D_i = T_i - T_{i-1}$ and expressing the above equations in terms of D_i. □

2 2-SAT

We now describe a simple randomized algorithm to find a satisfying assignment to a 2-SAT formula, if one exists. In particular, we wil give an algorithm that runs in time $O(n^2)$ and finds an assignment with probability at least $1/2$, if one exists. Note that we can run this algorithm several times and reduce our chance of not finding a solution to exponentially small.

The algorithm is the following:

1. Pick an arbitrary assignment to the variables x_i. If our initial choice satisifes the formula, we are done.

2. Otherwise, repeat the following for $2n^2$ steps: Pick an arbitrary unsatisfied clause. Choose one of the (at most) two variables occurring in it, and flip its value. If this is a satisfying assignment, return this assignment and exit.

Theorem 2 *If the formula is satisfiable, then with probability at least* $1/2$*, the algorithm above finds a satisfying assignment.*

Proof: Since we assume that the formula is satisfiable, there is at least one satisfying assignment $Y = (y_1, y2, ..., y_n)$. Let X_0 be the initial arbitrary assignment, and let X_t be the assignment after t steps. We will track the random variable D_t, which gives the overlap between the (random) assignment X_t and the satisfying assignment Y, i.e. the number of entries in which X_t and Y are the same.

Note that D_t takes its values in $\{0, \ldots, n\}$. Clearly, if $D_t = n$, then $X_t = Y$, and the algorithm stops. Our plan will be to show that no matter what X_0 is and no matter what Y is and no matter which unsatisfied clauses the algorithm picks during its execution, D_t reaches n with probability at least $1/2$ by time $2n^2$. Note that the algorithm might even stop earlier if it finds some different satisfying assignment, but we will not even try to exploit this.

The point is that whenever the algorithm flips a variable in some unsatisfied clause, D_t does a random walk which is at least as good as the drunk random walk as far as reaching n is concerned. For example, suppose that unsatisfied clause was (x_1, x_2), so both x_1 and x_2 were set to FALSE. In Y at least one of them must be set to T. Suppose the assignment under Y to x_1, x_2 was (T, F). So, if we flip either x_1 or x_2 randomly, with probability at least $1/2$, the overlap between X_{t+1} and Y_t increases by 1 (i.e. if we flip x_1), or decreases by 1 (if we flip x_2). The same holds for (F, T). In fact if the assignment was (T, T) under Y, our random walk does even better, since the overlap always increases by 1, no matter whether we flip x_1 or x_2. □

3 3-Coloring dense 3-colorable graphs

We saw that coloring 3-colorable graphs using few colors is a big open problem. Currently, we only know how to color such graphs using about $n^{0.2}$ colors. Interestingly, if the graph is dense, i.e. the minimum vertex degree is at least δn for some $\delta > 0$, then we can actually give a polynomial time algorithm to 3-color the graph. Here is the precise theorem.

Theorem 3 *If* G *is* 3*-colorable and the minimum degree is at least* δn*, then one can find a* 3*-coloring in time* $n^{O(1/\delta)}$*.*

Proof: The idea will be to use the 2-SAT algorithm above in a clever way.

First we find a small dominating set of size $O((\log n)/\delta)$. To see this, recall that the minimum degree if d, then a random subset of $(n/d)\log n$ vertices is a dominating set with probability at least $1/2$. Call this set S.

Second, we can try all possible 3-colorings of vertices of S. There are most $3^{|S|} = n^{O(1/\delta)}$ such possibilities to try. Now, we don't know how S is colored in the correct 3-coloring of G, but one of these colorings of S that we try will be the right one. So the idea would be that for each possible coloring of S, we try to extend it to a valid coloring of G. Clearly, for one of these colorings we will succeed.

So, it remains to show how to extend the coloring from S to G. The crucial observation is that since S is a dominating set, once we fix some coloring of S, each vertex $v \in V \setminus S$ has at most two choices of colors left for it. In fact, we can assume that there are exactly two choices for each vertex. Indeed, if some v has 0 choices we already know that there is no way to extend this coloring. If there is exactly one choice of color for v, we can fix this choice for v and continue this preprocessing.

Now, we claim that we can model the above problem of extending to a valid coloring of G as a 2-SAT instance. For each vertex v, we create a variable x_v and we create a local mapping where $x_v = T$ if v is assigned its first choice of color, or $x_v = F$ if v is assigned its second choice. Given such a mapping for each vertex, we can model the constraint that the endpoints of each edge should get different colors by a valid collection of clauses of size 2. For example, if (u, w) is an edge and u has the color choices red and green and w has choices red and yellow. Suppose we define convention, $x_u = T$ if u is colored red and F otherwise. For w, suppose we choose $x_w = F$ if w is red, and T otherwise. Then we need to forbid the assignment (red,red) to this edge, so we would put the clause $(\overline{x}_v \vee x_w)$. This will precisely forbid the (red,red) assignment to u and v is any satisfying solution to the 2-SAT instance. We do this for each edge (note that you add up to at most clauses for an edge). □

4 Polynomial Identity Testing

Our next example is a very fundamental problem, where randomization plays a very important role. In fact, we do not know of any deterministic algorithm for this problem, and designing such an algorithm would have profound consequences in computer science (in particular, it would allow us to prove super-polynomial size circuit lower bounds).

The problem is the following: Given an n-variate polyonmial $p(x_1, \ldots, x_n)$ with integer coefficients, determine if this polynomial is identically 0. Note that we are not asking whether this polynomial has some roots (i.e. where it evaluates to 0). But that it is identically 0. For example $p(x_1, x_2) = (x_1 + x_2)^2 - x_1^2 - x_2^2 - 2x_1x_2$ is identically 0.

First, we need to address the question how this polynomial is given to us as input. This is done in the form of a polynomial size arithmetic circuit, consisting of +-gates and $-$ gates and $\times$ gates, and where you can multiply an input by any integer.

Now, the most obvious strategy would be expand out this polynomial into monomials and see if everything cancels out. However, this problem is that this could be an exponential time algorithm as you might get exponentially many terms. For example, if you expand out $(x_1 + x_2)(x_3 + x_4)\cdots(x_{2n-1} + x_{2n})$, we get 2^n terms, even though this expression can be evaluated by simple circuit using $O(n)$ additions and multiplications.

Another natural strategy could be to evaluate it a random point, and check whether the answer is 0 or not. Indeed, if the polynomial is identically 0 the answer will always be 0 no matter where we evaluate it. On the other hand, if it was non-zero polynomial, we would expect it to be evalute to a non-zero number at least with some reasonable chance.

This intuition is indeed correct. For example, in the univariate case (i.e. $n = 1$), we know that a non-zero polynomial $P(x)$ of degree d can be zero at at most d values (as it has at most d roots). This motivates the following algorithm in general. For each $i = 1, \ldots, n$, assign $x_i := r_i$ where r_i is chosen randomly. Except that we need to define what a random point means. Another issue is that we need to make sure that evaluation can be done in polynomial time. In particular, we need to make sure that the intermediate numbers involved themselves do not become of exponential bit length. Fortunately, there is an easy fix for both these issues. We can just work in the field F_p (i.e. numbers modulo p for some prime p). Now pick x_i randomly just means we pick from uniformly from $\{0, \ldots, p-1\}$. Also, every time we multiply or add two numbers we can take modulo p, and ensures that all intermediate numbers say in $\{0, \ldots, p-1\}$.

So, the algorithm is the following: We independently and uniformly pick r_i at random $\in \mathbb{F}_p$ for $i = 1, \ldots, n$. If $P(r_1, \ldots, r_n) = 0$ return *yes*, otherwise return *no*. To this end, we have the following very useful result, which will also tell us what p to choose.

Theorem 4 (Schwartz–Zippel) *Let $P(x_1, \ldots, x_n)$ be a polynomial, but not the zero-polynomial. If $r_1, \ldots, r_n$ are uniformly random selected from $\mathbb{F}_p$, then:*

$$Pr[P(r_1, \ldots, r_n) = 0] \leq \frac{\deg(P)}{|\mathbb{F}_p|}.$$

Here the degree of P is the maximum degree among all its monomials, where the degree of the monomial $\prod_i x_i^{d_i}$ is defined as $d_1 + \ldots + d_n$.

Proof: We will proof this theorem using induction on n, the number of variables. For $n = 1$ this is trivial, as P has at most $\deg(P)$ roots and r_1 can have $|\mathbb{F}_p|$ values:

$$Pr[P(r_1) = 0] \leq \frac{\deg(P)}{|\mathbb{F}_p|}.$$

Let $d = \deg(P)$. For $n > 1$, we can extract one variable from P and write:

$$P(x_1, \ldots, x_n) = x_1^d P_0(x_2, \ldots, x_n) + x_1^{d-1} P_1(x_2, \ldots, x_n) + \ldots + x_1^0 P_d(x_2, \ldots, x_n),$$

with P_i being a polynomial of degree at most i. Let i be the smallest value such that $P_i(x_2, \ldots, x_n) \neq 0$ is satisfied. Such i exists, as otherwise P would be the zero polynomial. By induction hypothesis we have that:

$$Pr[P_i(r_2, \ldots, r_n) = 0] \leq \frac{i}{|\mathbb{F}_p|}.$$

Now fix any values for $r_2, \ldots, r_n$ and consider the univariate polynomial $P'(x)$ given by $P'(x) = P(x, r_2, \ldots, r_n)$. If $P_i(r_2, \ldots, r_n) \neq 0$, then P' is a non-zero polynomial of degree at most $d - i$, because for all $j < i$ its terms $x^{d-j} P_j(r_2, \ldots, r_n)$ are zero. Hence:

$$Pr_{r_1}[P(r_1, \ldots, r_n) = 0 \mid P_i(r_2, \ldots, r_n) \neq 0] \leq \frac{d-i}{|\mathbb{F}_p|}$$

and thus we have that

$$\begin{aligned} Pr[P(r_1,\ldots,r_n)=0] &= Pr[P(r_1,\ldots,r_n)=0 \mid P_i(r_2,\ldots,r_n)=0]\cdot Pr[P_i(r_2,\ldots,r_n)=0] \\ &\quad + Pr[P(r_1,\ldots,r_n)=0 \mid P_i(r_2,\ldots,r_n)\neq 0]\cdot Pr[P_i(r_2,\ldots,r_n)\neq 0] \\ &\leq Pr[P_i(r_2,\ldots,r_n)=0] + Pr[P(r_1,\ldots,r_n)=0 \mid P_i(r_2,\ldots,r_n)\neq 0] \\ &\leq \frac{i}{|\mathbb{F}_p|}+\frac{d-i}{|\mathbb{F}_p|} \\ &= \frac{d}{|\mathbb{F}_p|}. \end{aligned}$$

This concludes the induction and the proof. □

So if we choose $p = 2d$, there the probability that a non-zero degree d polynomial evaluates to 0 is at most 1/2.

4.1 Application: Perfect Bipartite matching

Let G be a bipartite graph. We want to check whether $G = (V, E)$ has a perfect matching. The best known deterministic algorithm has a running time of $O(|E|\sqrt{|V|})$. Hence if the graph is dense, $|E| \in \Omega(|V|^2)$, this algorithm is $O(|V|^{2.5})$. However a faster randomized algorithm exists that always answers *no* if there is no perfect matching and incorrectly answers *no* if there is a perfect matching with a probability less than $\frac{1}{2}$. For this we need a fast algorithm to compute the determinant of a matrix.

Theorem 5 (Fast matrix multiplication) *Let A and B be $n \times n$-matrices, then $C = A \times B$ can be computed in $O(n^{2.376})$ time.*

Corollary 6 *The determinant of A can be computed in $O(n^{2.376})$ time.*

In fact one can use this is a smart way to actually produce a matching in the same time, but we will not consider this here.

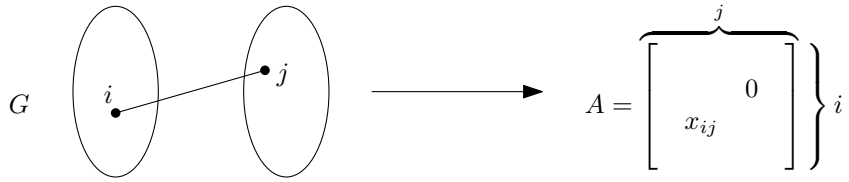

Figure 1: Transforming a bipartite graph into a matrix.

Given a bipartite graph G We create the matrix A, where the rows correspond to the vertices on the left side, and columns correspond to vertices on the right side of G. We set the entries $A_{ij} = x_{ij} \iff (i, j) \in E$ and $A_{ij} = 0$ otherwise (as illustrated in figure 1. Note that $\det A$ is a polynomial over the variables i, j, hence we can apply PIT to obtain our randomized algorithm.

Claim 7 *A perfect matching exists if and only if the determinant of A is not the zero-polynomial.*

Proof: The determinant of A can be written as:

$$\sum_{\sigma \in S^n} (-1)^{\text{sgn}(n)} \prod_{i=1}^{n} a_{i\sigma(i)}.$$

Each term in this sum is a potential perfect matching using $(i, \sigma(i))$ as the edges. If one of these edges does not exist in the graph, then the product multiplies by zero and the term does not contribute to the result of the sum. If no perfect matching exists, then all terms are zero and the determinant is the zero-polynomial. On the other hand, if a perfect matching $P \subset E$ does exist, the monomial $\prod_{(i,j)\in P} x_{ij}$ will survive. Hence the determinant is not the zero polynomial if a perfect matching exists. □