The Probabilistic Method

Lin Yang
LDCSEE
West Virginia University,
Morgantown, WV
{yang@csee.wvu.edu}

1 Introduction

In this lecture, we will study some basic principles of the *probabilistic method*, a combinatorial tool with many applications in computer science. This method is a powerful tool for demonstrating the existence of combinatorial objects. We introduce the basic idea through several examples drawn from early lecture notes, and follow that by a detailed study of the maximum satisfiability (MAX-SAT) problem. We then introduce the notion of expanding graphs and apply the probabilistic method to demonstrate their existence. These graphs have properties that prove useful in later lecture notes, and we illustrate these properties via an application to probability amplification.

In certain cases, the probabilistic method can actually be used to demonstrate the existence of algorithms, rather than merely combinatorial objects. We illustrate this by showing existence of efficient non-uniform algorithms for the oblivious routing. We then present a particular result, the Lovász Local Lemma, which underlies the successful application of the probabilistic method in a number of settings. We apply this lemma to the problem of finding a satisfying truth assignment in an instance of SAT problem where each variable occurs in a bounded number of clauses.

2 Overview of the Method

There are two recurrent themes in the probabilistic method.

1. Any random variable assume at least one value that is no smaller than its expectation, and at least one value that is no greater than its expectation.
2. If an object chosen randomly from a universe satisfies a property with positive probability, then there must be an object in the universe that satisfies that property. For instance, if we were told that a ball chosen randomly from a bin is red with probability 1/3, then we know that the bin contains at least one red ball.

Considering the following examples:

1. Ramsey Number of A Graph
   Let K_n be a complete graph with n vertices.

   Now let us consider this problem. Two-coloring of a graph is that each edge of the graph is colored either blue or red. The *Ramsey Number* $R(k, l)$ is the smallest integer n such that in any two-coloring of the edges of K_n by red and blue, either there is a red K_k (i.e., a complete subgraph on k vertices all of whose edges are colored red) or there is a blue K_l. Ramsey showed that$R(k, l)$ is finite for any two integers k and l. Let us obtain a lower bound for the Ramsey number $R(k, k)$.

Claim: 2.1 *if* $\binom{n}{k} 2^{1-\binom{k}{2}} < 1$ *then* $R(k,k) > n$. *Thus* $R(k,k) > 2^{\lfloor \frac{k}{2} \rfloor}$ *for all* $k \geq 3$

Proof: *Consider a random two-coloring of the edges of* K_n *obtained by coloring each edge independently either red or blue, where each color is equally likely. For any fixed set* R *of* k *vertices, let* A_R *be the event that the induced subgraph of* K_n *on* R *is monochromatic (i.e., that either all its edges are red or they are all blue). Each edge is colored either blue or red with a probability half. Thus the probability that all edges of the subgraph are colored either blue or red is* $(\frac{1}{2})^{\binom{k}{2}}$. *Clearly,*

$$\mathbf{Pr}[A_R] = 2(\frac{1}{2})^{\binom{k}{2}} = 2^{1-\binom{k}{2}}$$

Since there are $\binom{n}{k}$ *possible choices for* R, *the probability that at least one of the events* A_R *occurs is at most* $\binom{n}{k} 2^{1-\binom{k}{2}} < 1$. *Thus, with positive probability, no event* A_R *occurs and there a two-coloring of* K_n *without a monochromatic* K_k, *i.e.,* $R(k,k) > n$. *Note that if* $k \geq 3$ *and we take* $n = 2^{\lfloor \frac{k}{2} \rfloor}$ *then*

$$\binom{n}{k} 2^{1-\binom{k}{2}}$$

$$= \frac{n!}{k!(n-k)!} \times \frac{2}{2^{\frac{k!}{2(k-2)!}}}$$

Here we use the Stirling's approximation,

$$n! = (\frac{n}{e})^n$$

$$= \frac{n!}{k!(n-k)!} \times \frac{2}{2^{\frac{k!}{2(k-2)!}}}$$

$$= \frac{(\frac{2^{k/2}}{e})^{k/2}}{k!(n-k)!} \times \frac{2}{2^{\frac{k!}{2(k-2)!}}}$$

$$< \frac{2^{1+\frac{k}{2}}}{k!} \left(\frac{n}{2^{k/2}}\right)^k$$

$$\leq \frac{2^{1+\frac{k}{2}}}{k!} \ll 1$$

and hence

$$R(k,k) > 2^{\lfloor \frac{k}{2} \rfloor}$$

□

2. Binary Planar Partition
We have the following theorem from lecture note II.

Theorem: 2.1 *The expected size of the autopartition produced by* **RandAutois** $O(n \log n)$**.**

This asserts that for any set of n disjoint line segments in the plane, the expected size of the autopartition found by the **RandomAuto** algorithm is $O(n \log n)$. From this way we may conclude that for any set of n disjoint line segments in the plane, there is always an autopartition of size $O(n \log n)$. This follows directly from the fact that if we were to run the **RandoAuto** algorithm, the random defined to be the size of the autopartition can assume a value that is no more than its expectation; thus, there is an autopartition of this size on any instance.

3. Game Tree Evaluation
   Any algorithm for game tree evaluation that produces the correct answer on every instance develops a certificate of correctness: for each instance, it can exhibit a set of leaves whose values together guarantee the value it declares by the algorithm of the previous lecture related to this topic, the expected number of leaves inspected by the algorithm on any instance of $T_{2,k}$ is at most $n^{0.793}$, where $n = 2^{2k}$. It follows that on any instance of $T_{2,k}$, there is a set of $n^{0.793}$ leaves whose values certify the value of the root for that instance.

4. Set-Balancing Problem
   For every $n \times n$ 0-1 matrix A, for a randomly chosen vector $b \in \{-1, +1\}^n$, we have $\| Ab \|_\infty \leq 4\sqrt{n \ln n}$, with probability at least $1 - 2/n$. From this we may conclude that for every such matrix A, there always exists a vector $b \in \{-1, +1\}^n$, such that $\| Ab \|_\infty \leq 4\sqrt{n \ln n}$.

5. Max-Cut Problem
   Let us consider another example concerning the problem of finding a *large cut* in a graph. Given an undirected graph $\mathbf{G} =< \mathbf{V}, \mathbf{E} >$ with n vertices and m edges, we wish to partition the vertices of G into two sets of A and B so as to maximize the number of edges (u, v) such that $u \in A$ and $V \in B$. This problem is sometimes referred as the *max-cut* problem. The problem of finding an optimal max-cut is *NP*-hard; in contrast, the min-cut problem has a polynomial time algorithm.

Theorem: 2.2 *For any undirected graph $\mathbf{G} =< \mathbf{V}, \mathbf{E} >$ with n vertices and m edges, there is a partition of the vertex set V into sets A and B such that*

$$|\{(u, v) \ \in E \mid u \in A \text{ and } v \in B\}| \geq m/2$$

Proof: *Consider the following experiment. Each vertex of G is independently and equiprobably assigned to either A or B.*

For an edge u, v, the probability that its end-points are in different sets is $1/2$. *Following is a brief proof of why the probability is 1/2.*

Proof: The two end points of edge u is u_1 and u_2, and v has v_1 and v_2.
Let $x_1 = 1$, if either u_1 or u_2 v_1 or v_2 are in different set.
$x_2 = 0$, otherwise.

$$\mathbf{Pr}[\mathbf{x_1}] = \frac{1}{2}$$

$$\mathbf{E}[x_1] = 0 \times \frac{1}{2} + 1 \times \frac{1}{2}$$

The graph has m edges,

$$\mathbf{E}[X] = \sum_{i=1}^{m} \mathbf{E}[x_i]$$

By linearity of expectation,

$$\mathbf{E}[X] = \frac{m}{2}$$

□

By linearity of expectation, the expected number of edges with end-points in different sets is thus $m/2$. It follows that there must be a partition satisfying the theorem. □

3 Maximum Satisfiability

Given a set of m clauses in conjunctive normal form over n variables, decide whether there is a truth assignment for the n variables that satisfies all the clauses. We may assume without loss of generality that no clause contains both a literal and its complement, since such clause are satisfied by any truth assignment. Consider the following optimization version the problem: rather than decide whether there is an assignment that satisfies all the clauses, we instead seek an assignment that maximizes the number of satisfied clauses. This problem, called the MAX-SAT problem, is known to be ***NP-complete***.

Theorem: 3.1 *For any set of m clauses, there is an assignment to the input that satisfies at least $m/2$ clauses.*

Proof: *Suppose that each variable is set to TRUE or FALSE independently and equiprobably. For $1 \leq i \leq m$, let $Z_i = 1$ if the ith clause is satisfied and 0 otherwise. For any clause containing k literals, the probability that it is not satisfied by this random assignment is 2^{-k}, since this event takes place if and only if each literal gets a specific value, and the (distinct) literals in a clause are assigned independent values. This implies that that the probability that a clause with k literals is satisfied is $1 - 2^{-k} \geq 1/2$, implying that $\mathbf{E}[Z_i] \geq 1/2$ for all i. Let $Z = Z_1 + Z_2 + \cdots + Z_m$. By linearity of expectation, the expected number of clauses satisfied by this random assignment is $\sum_{i=1}^{m} \mathbf{E}[Z_i] \geq m/2$.* □

This results holds regardless of whether the instance has a satisfying assignment. Let us continue with the MAX-SAT problem, in which our goal is to maximize the number of clauses that are satisfied. This problem being **NP-complete**, we seek approximation algorithms. It turns out that variants of the probabilistic existence proof of Theorem 3.1 can actually be turned into approximation algorithms; we explore this theme for the reminder of this section.

Given an instance I, let $m_*(I)$ be the maximum number of clauses that can be satisfied, and let $m_A(I)$ be the number of clauses satisfied by an algorithm A. The performance ratio of an algorithm A is defined to be the infimum (over all instances I) of $m_A(I)/m_*(I)$. If A achieves a performance ratio of α, we call it an α-approximation algorithm. For a randomized algorithm A, the quantity $m_a(I)$ may be a random variable, in which case we replace $m_A(I)$ by $\mathbf{E}[m_A(I)]$ in the definition of performance ratio. Note that unlike the satisfiability problem (in which we seek to satisfy all clauses), we may choose to leave some clauses unsatisfied in the MAX-SAT problem.

We now give a simple randomized algorithm that achieves a performance ratio of 3/4. The idea is to formulate the problem as an integer linear program, solve the linear programming relaxation, and then to round using the randomized rounding technique introduced in Chapter 4.

With each clause C_j in the instance, we associate an indicator variable $z_j \in \{0, 1\}$ in the integer linear program to indicate whether or not that clause is satisfied. For each variable x_i, we use an indicator y_i. Let $y_i = 1$ if the variable x_i is set TRUE, ,and $y_i = 0$ otherwise. Let C_j^+ be the set of indices of variables that appear in the uncomplemented form in clause C_j, and C_j^- be the set of indices of variables that appear in the complemented form in clause C_j. We try to maximize

$$\sum_{j=1}^{m} z_j$$

where

$$y_i,\ z_j \in \{0, 1\}\ (\forall i \text{ and } j)$$

subject to

$$\sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_i) \geq z_j \ (\forall j)$$

Let $\hat{y_i}$ be the value obtained for y_i by solving this linear program, and let $\hat{z_i}$ be the value obtained for z_i by solving this linear program. Clearly $\sum_j \hat{z_j}$ is an upper bound on the number of clauses that can be satisfied in this instance.

For any positive integer k, let β_k denote $1 - (1 - 1/k)^k$. For a clause C_j with k literals, the probability that it is satisfied by randomized rounding is at least $\beta_k \hat{z_j}$.

Clause $C_j = (x_1, x_2, \ldots, x_k)$, and

$$\hat{x}_1 + \hat{x}_2 + \cdots + \hat{x}_k \geq \hat{z}_j$$

$$1 - \prod_{i=1}^{k} (1 - \hat{x}_i) \geq \beta_k \hat{z}_j$$

$$\beta_k = 1 - (1 - \frac{1}{k})^k$$

$$\lim_{k \to \infty} (1 - \frac{1}{k})^k = \frac{1}{e}$$

Therefore,

$$\beta_k = 1 - (1 - \frac{1}{k})^k \geq 1 - \frac{1}{e}$$

$f(x) = 1 - (1 - \frac{x}{k})^k$ is a concave function on $[0, 1]$ (See Appendix).
When $k = 1$,

$$f(x) = 1 - (1 - x) = x$$

When $k = 2$,

$$f(x) = 1 - (1 - \frac{x}{2})^2$$

$$= 1 - (\frac{x^2}{4} - x + 1)$$

$$= -\frac{x^2}{4} + 4$$

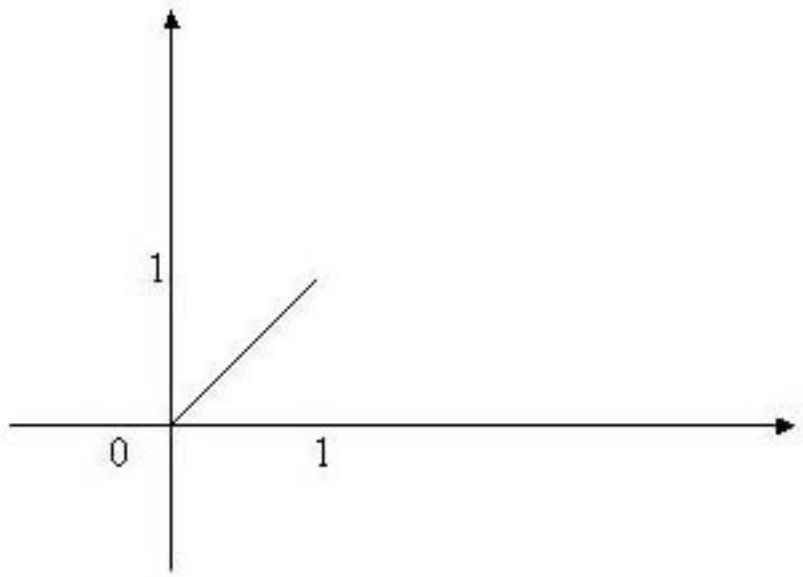

Figure 1: concave function $f(x) = x$

Lemma: 3.1 *Let C_j be a clause with k literals. The probability that it is satisfied by randomized rounding is at least $\beta_k \hat{z}_j$.*

Proof: *In the linear program,*

$$\hat{y_1} + \cdots + \hat{y_k} \geq \hat{z_j}$$

Since each variable is rounded independently, this occurs with probability $\prod_{i=1}^{k}(1 - \hat{y}_i)$. So we have

$$1 - \prod_{i=1}^{k}(1 - \hat{y}_i) \geq \beta_k \hat{z}_j$$

The expression on the left is minimized when $\hat{y}_i = \hat{z}_j/k$ for all i. Also $\max n_1, n_2 \geq \frac{n_1+n_2}{2}$. Therefore, it suffices to show that $1 - (1 - z/k)^k \geq \beta_k z$ for all positive integers k and $0 \leq z \leq 1$. Since $f(x) = 1 - (1 - z/k)^k$ is a concave (See Appendix A) function, to show that it is never less than a linear function $g(x)$ over the interval [0,1], it suffices to verify the inequality at the end-points $x = 0$ and $x = 1$. □

By Lemma (**??**) and from linearity of expectation we have:

Theorem: 3.2 *Given an instance of MAX-SAT, the expected number of clauses by linear programming and randomized rounding is at least $(1-1/e)$ times the maximium number of clauses that can be satisfied on that instance.*

We now argue that on any instance, one of the algorithm is a 3/4 approximation algorithm. Given any instance, we run both algorithms and choose the better solution. Let n_1 denote the expected number of clauses that are satisfied when each variable is independently set to 1 with probability 1/2. Let n_2 denote the expected number of clauses that are satisfied when we use the linear programming followed by randomized rounding.

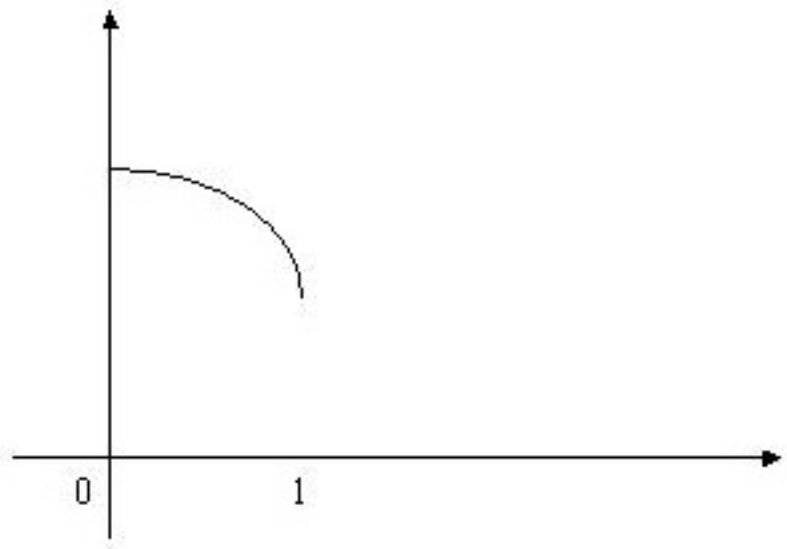

Figure 2: concave function $f(x) = -\frac{x^2}{4} + 4$

Theorem: 3.3

$$max\{n_1, n_2\} \geq \frac{3}{4} \sum_j \hat{z_j}$$

Proof: *Since* $\max\{n_1, n_2\} \geq \frac{n_1+n_2}{2}$, *it suffices to show that* $(n_1 + n_2)/2 \geq (3/4) \sum_j \hat{z_j}$. *Let* S^k *denote the set of clauses that contain k literals, we know that*

$$n_1 = \sum_k \sum_{C_j \in S^k} (1 - 2^{-k}) \geq \sum_k \sum_{C_j \in S^k} (1 - 2^{-k}) \hat{z_j}$$

By Lemma(??), we have

$$n_2 \geq \sum_k \sum_{C_j \in S^k} \beta_k \hat{z_j}$$

Thus

$$\frac{n_1 + n_2}{2} \geq \sum_k \sum_{C_j \in S^k} \frac{(1 - 2^{-k}) + \beta_k}{2} \hat{z_j}$$

Since $(1 - 2^{-k}) + \beta_k \geq 3/2$ *for all k, so that we have*

$$\frac{n_1 + n_2}{2} \geq \frac{3}{4} \sum_k \sum_{C_j \in S^k} \hat{z_j} = \frac{3}{4} \sum_j \hat{z_j}$$

□

4 Expanding Graphs

Expanding graphs have found many uses in computer science and in telephone switching networks.

Intuitively, an expanding graph is a graph in which the number of neighbors of any set of vertices S is larger than some positive constant multiple $|S|$. The following is a definition of a particular type of expanding graph called an OR-concentrator. Recall that in a graph $G(V, E)$ for any set $S \subseteq V$, the set of neighbors of S is $\Gamma(S) = \{w \in V \mid \exists v \in S,\ (v, w) \in E\}$.

Definition: 4.1 *An (n, d, α, c) OR-concentrator is a bipartite multigraph $G(L, R, E)$, with the independent sets of vertices L and R each of cardinality n, such that*

1. *Every vertex in L has degree at most d.*
2. *For any subset S of vertices from L such that $|S| \leq \alpha n$, there are at least $c|S|$ neighbors in R.*

In most applications, it is desirable to have d as small as possible and c as large as possible, $|\Gamma(S)| \geq c|S|$.

Theorem: 4.1 *There is an integer n_0 such that for all $n > n_0$, there is an $(n, 18, 1/3, 2)$ OR-concentrator.*

Proof: *Consider a random bipartite graph on the vertices in L and R, in which each vertex of L chooses its neighbors by sampling (with replacement) d vertices independently and uniformly from R. Since the sampling is with replacement, a vertex of L may choose a vertex in R more than once; we discard all but one copy of such multiple edges. Let ξ_s denote the event that a subset of s vertices of L has fewer than cs neighbors in R. We will first bound $\mathbf{Pr}[\xi_s]$, and then sum $\mathbf{Pr}[\xi_s]$ over the values of s no larger than αn to obtain an upper bound on the probability that the random graph fils to be an OR-concentrator with the parameters we seek.*

Fix any subset $S \subseteq L$ of size s, and any subset $T \subseteq R$ of size cs. There are $\binom{n}{s}$ ways of choosing S, and $\binom{n}{cs}$ ways choosing T. The probability that T contains all of the at most ds neighbors of the vertices in S is $(cs/n)^{ds}$. Thus, the probability of the event that all the ds edges emanating from some s vertices of L fall within and cs verticees of R is bounded as follows,

$$\boldsymbol{Pr}[\xi_s] \leq \binom{n}{s}\binom{n}{cs}\left(\frac{cs}{n}\right)^{ds}$$

We use the identity $\binom{n}{k} \leq (ne/k)^k$ and obtain

$$\begin{aligned}\boldsymbol{Pr}[\xi_s] &\leq \left(\frac{ne}{s}\right)^s \left(\frac{ne}{cs}\right)^{cs} \left(\frac{cs}{n}\right)^{ds} \\ &\leq (s^{d-c-1} n^{c+1-d} e^{1+c} c^{d-c})^s \\ &= \left[\left(\frac{s}{n}\right)^{d-c-1} e^{1+c} c^{d-c}\right]^s\end{aligned}$$

Using $\alpha = 1/3$ and $s \leq \alpha n$, we have

$$\begin{aligned}\boldsymbol{Pr}[\xi_s] &\leq \left[\left(\frac{1}{3}\right)^{d-c-1} e^{1+c} c^{d-c}\right]^s \\ &\leq \left[\left(\frac{1}{3}\right)^{d} 3^{c+1} e^{1+c} c^{d-c}\right]^s\end{aligned}$$

$$\leq \left[\left(\frac{c}{3}\right)^d (3e)^{c+1} \right]^s$$

Using $c = 2$ and $d = 18$, we have

$$\boldsymbol{Pr}[\xi_s] \leq \left[\left(\frac{2}{3}\right)^{18} (3e)^3 \right]^s$$

Let $r = (2/3)^{18}(3e)^3$, and note that $r < 1/2$. We obtain that

$$\sum_{s \geq 1} \boldsymbol{Pr}[\xi_s] \leq \sum_{s \geq 1} r^s = \frac{r}{1-r} < 1$$

and the desired result follows. □

We may easily see that by rounding the probabilities $\mathbf{Pr}[\xi_s]$ carefully, we can in fact show that our random graph has a fairly good (rather than merely non-zero) probability of being an $(n,\ 18,\ 1/3\ ,2)$ OR-concentrator. However, if we were to generate a random graph and argue that it has a very high probability of being an OR-concentrator, we still do not know of an efficient way of verifying that graph generated is indeed an OR-concentrator with the given parameters.

For many applications of expanding graphs, such a Monte Carlo guarantee is unacceptable - for instance, a telephone company may be uncomfortable that the network it plans to build may by chance be inadequate. Unfortunately, it is considerably harder to give a succinct "formula" or a deterministic algorithm that, given n, always generates such an expanding graph. We do have "explicit constructions" that will, given n, generate OR-concentrator with guaranteed bounds for d, α and c; but these bounds are somewhat weaker than the bounds attainable using the probabilistic method.

5 Oblivious Routing

In this section we address the problem of oblivious permutation routing on the hypercube. We focus on the number of random bits used by the randomized oblivious algorithm. Comparing the performance of the randomized algorithm with the negative result of Theorem 4.4, we find that our randomized oblivious algorithm achieves an expected running time that no deterministic oblivious algorithm can achieve. Given that randomness is absolutely necessary to beat the lower bound of $\sqrt{N/n}$ steps for deterministic oblivious algorithms, we approach the question that how much randomness is actually needed to achieve an algorithm with an expected time of $O(n)$.

Each deterministic oblivious algorithm for an N-node network is a set of N^2 routes, one for each source-sink pair. Every randomized oblivious algorithm can be expressed as a pair of sets, $\{A_1, \ldots, A_R\}$ and $\{p_1, \ldots, p_R\}$, where each A_j is a deterministic oblivious algorithm and p_j is the probability that we use A_j on a run of the randomized algorithm.

Theorem: 5.1 *Consider any randomized oblivious algorithm for permutation routing on the hypercube with $N = 2^n$ nodes. If this algorithm uses k random bits, then its expected running time is $\Omega(2^{-k}\sqrt{N/n})$.*

Proof: *We have observed that any randomized oblivious algorithm is a probability distribution on deterministic oblivious algorithms. Since only k random bits are used, at least one of these deterministic algorithms is chosen with probability at least 2^{-k}, on any execution. Denote this deterministic algorithm by A_1. By the lower bound of Theorem* 5.1*, there is an input that requires time $\Omega(\sqrt{N/n})$ on A_1. Feed this input to randomized algorithm;*

with probability 2^{-k}, the randomized algorithm chooses A_1 and takes time $\Omega(\sqrt{N/n})$. Thus, the expected running time of the randomized algorithm is $\Omega(2^{-k}\sqrt{N/n})$. □

Corollary: 5.1 *Any oblivious algorithm for permutation routing on the hypercube with $N = 2^n$ nodes must use $\Omega(n)$ random bits in order to achieve expected running time $O(n)$.*

Proof: *The expected running time of the algorithm is*

$$\Omega(\sqrt{N/n})$$

According to the definition of big-O, we have

$$2^{-k}\sqrt{N/n} = cn$$

c is a constant

$$2^{-k}\sqrt{2^n/n} = cn$$
$$\frac{2^{-k}2^{\frac{n}{2}}}{n^{\frac{1}{2}}} = cn$$
$$2^{\frac{n}{2}} = cn^{\frac{3}{2}}2^k$$
$$\lg 2^{\frac{n}{2}} = \lg\left(cn^{\frac{3}{2}}2^k\right)$$
$$\frac{n}{2} = \lg c + \frac{3}{2}\lg n + k$$
$$k = \frac{n}{2} - \lg c - \frac{3}{2}\lg n$$

We keep the dominant item and then have

$$k = \frac{1}{2}n = \Omega(n)$$

□

Theorem: 5.2 *For every n, there exists a randomized oblivious scheme for permutation routing on a hypercube with $N = 2^n$ nodes that uses $3n$ random bits and runs in expected time at most $15n$.*

Proof: *We will say that a set $\beta = \{B_1, B_2, \ldots, B_t\}$ of deterministic oblivious permutation routing algorithms on the N-node hypercube is an efficient N-scheme if, for any instance, the expected number of steps using a randomly chosen algorithm from β is at most $15n$. To prove the theorem, for every $N = 2^n$, there is an efficient N-scheme for $t = N^3$.*

The hypercube has N nodes. When routing on the hypercube, the intermediate nodes of each of the N node has a $N_1 + N_2 + \cdots + N_N = N^N$ execution. Each execution is a deterministic algorithms. So there are N^N possible deterministic algorithms on an execution. Let us denote these N^N algorithms by A_j, $1 \le j \le N^N$. On an N-node network, there are $N!$ distinct possible instances of permutation routing, one for each permutation on $\{1, \ldots, N\}$. By Theorem 4.7 of textbook, for any particular instance of the permutation routing problem, at most $1/N$ of A_j are bad.

Consider the following experiment: sample N^3 indices $i_1, i_2, \ldots, i_{N^3}$ independently and uniformly at random from the range $\{1, 2, \ldots, N^N\}$. We shall show that the set of deterministic algorithms $A = \{A_{i_1}, \ldots, A_{i_{N^3}}\}$ is an efficient scheme.

The expected number of algorithms in A that are bad is at most $N^3(1/N) \leq N^2$. *Let* X_j *be 1 if* A_j *is bad and* 0 *otherwise. Thus* $E[\sum_j X_j] \leq N^2$. *Since the* X_j *are independent Poisson trials, we use the Chernoff bound,*

$$\Delta^+(\mu, \epsilon) < \sqrt{\frac{4 \ln 1/\epsilon}{\mu}}$$

to obtain that the probability that more than $2N^2$ *of the algorithm in A are bad is less than* $e^{-N^2/4}$. *See following,*

$$(1+\delta)\mu = 2N^2$$

$$\mu = N^2$$

$$1 + \delta = 2$$

$$\delta = 1$$

$$\delta^+(\mu, \epsilon) = 1$$

$$1 < \sqrt{\frac{4 \ln 1/\epsilon}{\mu}}$$

$$\epsilon \leq e^{-N^2/4}$$

Let $\epsilon = e^{-N^2/4}$.

Let β_i *denote the bad event that more than* $2N^2$ *algorithms in A are bad. Then for* $n \geq 2$,

$$\mathbf{Pr}[\cup_{i=1}^{N!} \beta_i] \ \leq \ \sum_{i=1}^{N!} \mathbf{Pr}[\beta_i]$$

$$\leq N! \times e^{-N^2/4}$$

Here we use identity $N! = (\frac{N}{e})^N$.

$$\leq \frac{N^N}{e^N} \times e^{-N^2/4}$$

$$\leq \frac{N^N}{e^{\frac{N^2}{4}+N}}$$

Let $\alpha = \frac{N^N}{e^{\frac{N^2}{4}+N}}$.

$$ln\left(\frac{N^N}{e^{\frac{N^2}{4}+N}}\right) = \ln \alpha$$

$$N \ln N - (\frac{N^2}{4} + N) = \ln \alpha$$

Since N^2 *is a dominant term here, thus,*

$$\ln \alpha < 0$$

$$\alpha < 1$$

Therefore,

$$\mathbf{Pr}[\cup_{i=1}^{N!} \beta_i] \ \leq \ \sum_{i=1}^{N!} \mathbf{Pr}[\beta_i]$$

$$< 1$$

Therefore, with positive probability, no more than $2N^2$ of the algorithms in A are bad for any instance of permutation routing on the N-node hypercube. This means there exists a subset of N^3 algorithms from $\{A_1, \ldots, A_{N^N}\}$ with the property that at most $2N^2$ algorithms in this subset are bad for any instance. Let us denote this subset by $\beta = \{B_1, B_2, \ldots, B_{N^3}\}$.
On any instance π_i, a randomly chosen algorithm from β fails to route π_i within $14n$ steps with probability at most $2N^2/N^3 = 2/N$. By reasoning similar to that in Exercise 4.6 of textbook of textbook, Thus, the expected number of steps using an algorithm randomly chosen from β is less than $15n$. □

6 The Lovász Local Lemma

The Lovász Local Lemma is a tool in the probabilistic method that has found many applications in extremal graph theory, in Ramsey theory, and in the theory of random graphs.

Suppose we have n events, each of which occurs with probability at most $1/2$. In an instance of the probabilistic method, each of the n events may correspond to one of n ways in which the probabilistic experiment could fail. If the events were independent, we could then assert that with probability at least 2^{-n}, none of these events occurs. The Lovász Local Lemma generalizes this notion to the case where each of these events is independent of all but a small number of other events. Let $\xi_i,\ 1 \leq i \leq n$ be events in a probability space. Recall that ξ_i is mutually independent of a set S of events if $\mathbf{Pr}[\xi_i | \cap_{j \in T} \xi_j] = \mathbf{Pr}[\xi_i]$, where T is any subset of events from S. The main device in establishing this lemma is dependency graph G, in which there is a vertex representing each event ξ_i. An event ξ_i is mutually independent of all other events ξ_j such that $(\xi_i,\ \xi_j)$ is not an edge of the graph.

Lemma: 6.1 *Let G(V,E) be a dependency graph for events $\xi_1, \ldots, \xi_n$ in a probability space. Suppose that there exists $x_i \in [0,1]$ for $1 \leq i \leq n$ such that*

$$\mathbf{Pr}[\xi_i] \leq x_i \prod_{(i,j) \in E} (1 - x_i)$$

Then

$$\mathbf{Pr}[\cap_{i=1}^{n} \bar{\xi_i}] \geq \prod_{i=1}^{n} (1 - x_i)$$

Proof: *Let S denote a subset of the indices from $\{1, \ldots, n\}$. We first establish by induction on $k = |S|$ that for any S and for any i such that $i \notin S$,*

$$\mathbf{Pr}[\xi_i | \cap_{j \in S} \bar{\xi_j}] \leq x_i$$

(See Appendix A for the proof of the following probabilistic identity [1])

$$\mathbf{Pr}[A | (B \cap C)] = \frac{\mathbf{Pr}[(A \cap B) | C]}{\mathbf{Pr}[B | C]} \quad [1]$$

The base case is $S = \phi$. For the inductive step, ,we let $S_1 = \{j \in S :\ (i,j)\ \in\ E\}$, and let $S_2 = S_2/S_1$, by the definition of conditional probability,

$$\mathbf{Pr}[\xi_i | \cap_{j \in S} \bar{\xi_j}] = \frac{\mathbf{Pr}[\xi_i \cap (\cap_{j \in S_1} \bar{\xi_j}) | \cap_{m \in S_2} \bar{\xi_m}]}{\mathbf{Pr}[\cap_{j \in S_1} \bar{\xi_j} | \cap_{m \in S_2} \bar{\xi_m}]}$$

We can bound the numerator to:

$$\mathbf{Pr}[\xi_i \cap (\cap_{j \in S_1} \bar{\xi_j}) | \cap_{m \in S_2} \bar{\xi_m}] \leq \mathbf{Pr}[\xi_i | \cap_{m \in S_2} \bar{\xi_m}]$$

$$= \mathbf{Pr}[\xi_i]$$
$$\leq x_i \prod_{(i,j)\in E} (1 - x_j)$$

Suppose that $S_1 = \{j_1, \ldots, j_r\}$. *If* $r = 0$, *then the denominator is* 1; *for* $r > 0$, *we invoke the induction hypothesis:*

$$\mathbf{Pr}[\xi_{j_1} \cap \cdots \cap \bar{\xi_{j_r}}] = (1 - \mathbf{Pr}[\xi_{j_1} | \cap_{m\in S_2} \bar{\xi_m}]) \cdots (1 - \mathbf{Pr}[\xi_{j_r} | \bar{\xi_{j_1}} \cap \cdots \bar{\xi}_{j_r-1} \cap_{m\in S_2} \bar{\xi_m}])$$
$$\geq (1 - x_{j_1}) \cdots (1 - x_{j_r}) \geq \prod_{(i,j)\in E} (1 - x_j)$$
$$\mathbf{Pr}[\xi_i | \cap_{j\in S} \bar{\xi}_j] \leq x_i$$
$$\mathbf{Pr}[\cap_{i=1}^n \bar{\xi}_i] = (1 - \mathbf{Pr}[\xi_1])(1 - \mathbf{Pr}[\xi_2|\bar{\xi}_1]) \cdots (1 - \mathbf{Pr}[\xi_n | \cap_{i=1}^{n-1} \bar{\xi}_i])$$
$$\geq \prod_{i=1}^{n} (1 - x_i)$$

□

Corollary: 6.1 *Let* $\xi_1, \ldots, \xi_n$ *be events in a probability space, with* $\mathbf{Pr}[\xi_i] \leq p$ *for all* i. *If each event is mutually independent of all other events except for at most* d, *and if* $ep(d + 1) \leq 1$, *then* $\mathbf{Pr}[\cap_{i=1}^n \bar{\xi}_i] > 0$.

$$\left(1 - \frac{1}{d+1}\right) = \frac{1}{e}$$
$$x_i = \frac{1}{d+1}$$

We now apply Corollary 6.1 to prove k-SAT (each clauses has exactly k literals. Suppose that each of the n variables appears (complemented or uncomplemented) in at most $2^{k/50}$ clauses. Let m denote the number of clauses.

Consider a random truth assignment of values to the variables, in which each variable is independently fixed to be 0 or 1 with probability 1/2. For $1 \leq i \leq m$, let ξ_i denote the event that the ith clause is not satisfied by this random assignment. Since each clause contains k literals, we have $\mathbf{Pr}[\xi_i] = 2^{-k}$, for $1 \leq i \leq m$. The event ξ_i that the ith clause is not satisfied is independent of all other events ξ_j, except those j such that clause i and clause j share at least one variable. The number of clauses j that share a variable with a specific clause i cannot exceed the total number of clauses containing the variables that appear in clause i, and this is at most $k2^{k/50}$. We now apply Corollary 5.12 with $d = k2^{k/50}$, and conclude that with positive probability the random truth assignment satisfies all m clauses. Thus, there must be a satisfying truth assignment for any instance of SAT meeting these conditions.

Corollary 6.1 merely tells us that a random assignment is good with positive probability, but this probability is small. We may have to try the random assignment many times before we succeed in finding one that satisfies all m clauses. We now describing a Las Vegas randomized algorithm that runs in time polynomial in m, yielding a satisfying truth assignment.

Let G denote the dependency graph - each clause corresponds to a vertex of G, and two vertices are adjacent in G if the corresponding clauses share one ore more variables. Note that if clause C_1 contains literal x_1, ,and clause C_2 contains literal $\bar{x}_1$, then the vertices C_1 and C_2 are adjacent.

At any point in the algorithm, some variables have been fixed to 0 or 1, while others remain unspecified yet; initially, all variables are unspecified. The algorithm consists of two stages; the first stage will fix values for some od the variables and defer the rest to the second stage. We call a clause dangerous if both the following conditions hold:

1. $k/2$ literals of the clause C_i have been fixed.

2. C_i is not satisfied yet.

For any dangerous clause, we defer its remaining unspecified variables to the second stage, skipping them over in the sequential random assignment. At the end of the first stage, we say that a clause has survived if it is not satisfied by the variables fixed in the first stage.

For the second stage we need only consider the variables that were unspecified at the end of the first stage, and the clauses that survived. A clause C_i can survive the first stage for one two two reasons:

1. It became dangerous, or

2. All variables corresponding to its unspecified literals were deferred because other clauses containing these variables (and, hence, adjacent to C_i) became dangerous

Therefore, a clause C_i may survive as a result of any one of up to $d+1$ clauses becoming dangerous - C_i itself, and its d neighbors. Every clause that survived has at least $k/2$ unspecified variables.

Exercise: Apply Corollary 6.1 to show that there is a truth assignment of the deferred variables that satisfies all the surviving clauses. (Again, consider a random assignment.)

The second stage will find a truth assignment. The probability that any particular clauses becomes dangerous during the first stage is at most 2^{-k}, since exactly $k/2$ of its literals have their values fixed, and none of these random variables satisfy the clause. This implies that the probability that a clause survive is at most $(d+1)2^{-k/2}$.

Consider the subgraph of G induced by the vertices corresponding to the surviving clauses. In Lemma 6.2 below, we will show that with high probability, all connected components of this induced subgraph of G have size $O(\log m)$. Notice that two surviving clauses from different connected components of this subgraph cannot share a deferred variable. Therefore, the deferred variables can be uniquely assigned to distinct components of the subgraph G induced by the surviving clauses. For any particular connected component, the total number of deferred variables in its clauses must be $O(\log m)$; in time polynomial in m, we can enumerate the $2^{O(\log m)}$ truth assignments for these variables until that satisfies all clauses in this component. The second stage consists of repeating this process independently for each connected component, giving a polynomial time algorithm for assigning values to the deferred variables so as to satisfy all surviving clauses.

Lemma: 6.2 *With probability $1-o(1)$, all connected components of G induced by the clauses that survive the first stage have size at most zlogm, for a fixed constant z.*

Proof:*Consider a collection of clauses $C_1,\ldots,C_r$ such that every pair of these has distance as least 4 in G. Each clause C_i survives only if at least one of the $d+1$ clauses at distance at most 1 it turns dangerous during the first stage. For each C_i, let D_i be any one dangerous clause at distance at most 1 from it. Since the C_i's are at distance 4 from each other, the D_i's must be distinct.*

There are at most $(d+1)^r$ possible ways of choosing the clauses $D_1,\ldots,D_r$. Since each of the clauses $D_1,\ldots,D_r$ is at at distance 1 from some clause in the set $C_1,\ldots,C_r$, ,they must be at distance at least 2 from each other and hence have disjoint sets of variables. The probability that $D_1,\ldots,D_r$ all become dangerous is at most $2^{-rk/2}$. Thus, for a set of of r clauses every pair of which is distance at least 4 apart in G, the probability that they all survive is at most

$$[(d+1)2^{-k/2}]^r$$

We must bound the probability that some connected subgraph of G of size zlogm survives. To this end we introduce a graph-theoretic device known as $4-tree$. Call a subset T of clauses a $4-tree$ if the following two properties hold:

1. *The distance in G between every pair of these clauses is at least* 4.

2. *If we form a new graph in which two clauses are adjacent if their distance in G is exactly* 4, *T is connected.*

We first bound the number of 4-trees of size r and use this to bound the probability that a large 4-tree survives. By arguing that a large connected subgraph of G must contain a large 4-tree, we will finally conclude it is unlikely that a large connected subgraph survives.

Define a new graph G_4 as follows: there is a vertex for each clause, and two vertices are adjacent in G_4 if their distance in G is 4. Each vertex of G_4 has $O\left(d^4\right)$ neighbors. The number of 4-trees of size r in G is no more than the number of connected subgraphs in G_4 of size r. The number of subgraph of G_4 of size r is at most

$$amd^{8r}$$

For some constant a, and this is an upper bound on the number of 4-trees of size r in G. By having the analysis above, we conclude that the probability that any 4-tree of size larger than $b \log m$ survives the first round is $o(1)$, for a suitably large constant b.

For any connected subgraph in G there is a maximal 4-tree T, together with at most $3d^3 - 1$ other vertices within distance 3 of a vertex of T. Thus the size of this subgraph is at most $3|T|d^3$. We conclude that the probability of survival of any connected subgraph of size exceeding $3bd^3 \log m$ is $o(1)$.
□

If the first stage results in a connected component larger than this bound, we repeat it; the expected number of repetitions is less than 2. Thus, we assume that we enter the second stage of the algorithm with every surviving connected component having size $O(logm)$. The number of unspecified variables associated with each of these components is also $O(\log m)$, and in time polynomial in m we can find values for them that satisfy all the clauses. Since no variable is shared by two or more components, we can treat each component in isolation. Clearly the expected running time of this algorithm is polynomial in m.

Theorem: 6.1 *The above algorithm finds a satisfying truth assignment for any instance of k-SAT containing m clauses in which each variable is contained in at most $2^{k/50}$ clauses, in expected time polynomial in m.*

A Appendix: Some Important Results

A.1 Concave Functions

To extend the notions of concavity and convexity to functions of many variables we first define the notion of a convex set.

Definition: A.1 *A set S of n-vectors is convex if $(1-\lambda)x + \lambda x' \in S$ whenever $x \in S$, $x \in S$, and $\lambda \in [0,1]$.*

Therefore, we have the following definition for Concave Functions.

Definition: A.2 *Let f be a function of many variables, defined on a convex set S. We say that f is concave if the line segment joining any two points on the graph of f is never above the graph.*
Then f is concave on the set S if for all $x \in S$, all $x' \in S$, and all $\lambda \in (0,1)$ we have

$$f((1-\lambda)x + \lambda x') \geq (1-\lambda)f(x) + \lambda f(x')$$

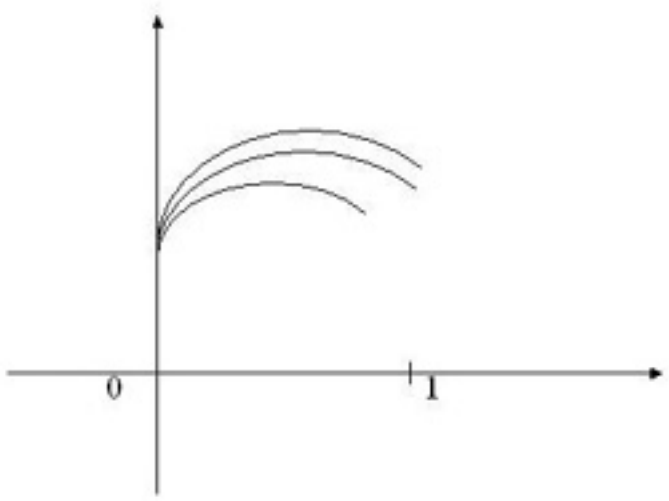

Figure 3: Concave Functions

$$f(x) = 1 - (1 - \frac{x}{k})^k$$

$$f((1-\lambda)x + \lambda x')$$

$$= 1 - \left(1 - \frac{(1-\lambda)x + \lambda x'}{k}\right)^k$$

$$\geq (1-\lambda)f(x) + \lambda f(x')$$

$$\geq (1-\lambda)\left(1 - (1 - \frac{x}{k})^k\right) + \lambda\left(1 - (1 - \frac{x'}{k})^k\right)$$

A.2 A Proof of A Probabilistic Identity

$$\mathbf{Pr}[A|(B \cap C)] = \frac{\mathbf{Pr}[(A \cap B)|C]}{\mathbf{Pr}[B|C]}$$

Proof: *By using the following identities,*

$$\mathbf{Pr}[E|F] = \frac{\mathbf{Pr}[EF]}{\mathbf{Pr}[F]}$$

$$\mathbf{Pr}[E \cap F] = \mathbf{Pr}[E|F] \times \mathbf{Pr}[F]$$

Therefore,

$$\mathbf{Pr}[A|(B \cap C)] = \frac{\mathbf{Pr}[(A \cap B \cap C)]}{\mathbf{Pr}[B \cap C]}$$

$$= \frac{\mathbf{Pr}[(A \cap B) \cap C]}{\mathbf{Pr}[B \cap C]}$$

$$= \frac{\mathbf{Pr}[(A \cap B)|C] \times \mathbf{Pr}[C]}{\mathbf{Pr}[B|C] \times \mathbf{Pr}[C]}$$

$$= \frac{\mathbf{Pr}[(A \cap B)|C]}{\mathbf{Pr}[B|C]}$$

□

References

[1] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge Univrsity Press, 1995.

[2] Martin J. Osborne. *Concave and convex functions of many variables.* http://www.chass.utoronto.ca/ osborne/MathTutorial/CVN.HTM