MA421 Advanced Algorithms Lent 2019

3 Randomised Algorithms

Reading:

- **Probability Background:** Cormen et al. *Introduction to Algorithms*, Appendix C. Third Edition. MIT Press, 2009.
- Motwani and Raghavan. *Randomized Algorithms.* Cambridge Press, 1997.

The presentation in Sections 3.1–3.4 is based on Motwani and Raghavan book. The writeup of Section 3.5 is mostly based on the lecture notes from a course taught at the University of California, Berkeley by Alistair Sinclair, but a very similar analysis can also be found in Motwani and Raghavan book.

3.1 Probability Background

In the study of randomised algorithms, we mostly need to use discrete and finite probability spaces. In this section, we will revise some background about probability distributions on discrete and finite probability spaces.

A *sample space* Ω is a (in our case, finite and discrete) set, and its elements are referred to as *elementary events.* An *event* of the probability space is a subset of the sample space: in other words, an element of 2^Ω. For our purposes, it is sufficient to concentrate on cases where each event in 2^Ω is possible (that is, the σ-field $(\Omega, 2^\Omega)$). Finally, a *probability measure* (or, equivalently for a discrete distribution, probability mass function) $\Pr : 2^\Omega \to \mathbb{R}$ is a function that satisfies the following conditions.

1. $\forall E \in 2^\Omega, 0 \le \Pr[E] \le 1$;
2. $\Pr[\Omega] = 1$;
3. For mutually disjoint events $E_1, E_2, \ldots$, $\Pr[\cup_i E_i] = \sum_i \Pr[E_i]$.

We next go over some fundamental concepts.

Definition 3.1 (Conditional Probability) *The* conditional probability *of E_1 given event E_2 is*

$$\Pr[E_1 | E_2] = \frac{\Pr[E_1 \cap E_2]}{\Pr[E_2]},$$

assuming that $\Pr[E_2] > 0$.

Definition 3.2 (Independent Events) *A collection of events $\{E_i \mid i \in I\}$ is* independent *if, for all subsets $S \subseteq I$,*

$$\Pr[\cap_{i\in S} E_i] = \prod_{i\in S} \Pr[E_i].$$

Equivalently, for any $j \in I$ and $S \subseteq I \setminus \{j\}$,

$$\Pr[E_j \mid \cap_{i\in S} E_i] = \Pr[E_j].$$

For intersection of k events $E_1, \ldots, E_k$,

$$\Pr[\cap_{i=1}^{k} E_i] = \Pr[E_1] \cdot \Pr[E_2 \mid E_1] \cdot \Pr[E_3 \mid E_1 \cap E_2] \cdots \Pr[E_k \mid \cap_{i=1}^{k-1} E_i]. \tag{1}$$

The following fact is very commonly used to bound the probability of at least one of the low-probability bad events happening.

Fact 3.3 (Union Bound) *For events $\{E_i \mid i \in I\}$,*

$$\Pr\left[\cup_{i\in I} E_i\right] \le \sum_{i\in I} \Pr[E_i].$$

Definition 3.4 (Random Variable) *A random variable $X : \Omega \to \mathbb{R}$ is a real-valued function over the sample space.*

Definition 3.5 (Independent Random Variables) *Random variables X and Y are independent if, for all $x, y \in \mathbb{R}$,*

$$\Pr[X = x \mid Y = y] = \Pr[X = x].$$

Definition 3.6 (Expectation) *The expectation of a random variable X is*

$$\mathbb{E}[X] = \sum_{x} x \cdot \Pr[X = x],$$

where the summation is over the range of X.

Fact 3.7 (Linearity of Expectation) *Let $X_1, \ldots, X_k$ be arbitrary random variables. Then,*

$$\mathbb{E}[X_1 + \cdots + X_k] = \mathbb{E}[X_1] + \cdots + \mathbb{E}[X_k].$$

Fact 3.8 *For independent random variables X and Y,*

$$\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y].$$

Definition 3.9 (Variance) *The variance of a random variable X is*

$$var[X] = \mathbb{E}[(X - \mathbb{E}[X])^2].$$

Fact 3.10 $var[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.

Fact 3.11 *For independent random variables X and Y, $var[X + Y] = var[X] + var[Y]$.*

3.2 Introduction to Randomised Algorithms

Use of randomness during a computation can have one of several purposes such as foiling an adversary, random sampling, finding a witness in an abundance of witnesses, fingerprinting and hashing, random re-ordering, load balancing, isolation and symmetry breaking, probabilistic methods and existence proofs.

A *randomised algorithms* uses, in addition to its input, a random string (say, of 0's and 1's) to make random moves or choices.

Given input x to a randomised algorithm A,

$$\Pr[A(x) \text{ is correct}] > \frac{3}{4},$$

where the probability is taken over the random choices the algorithm makes.

The choice of fraction $3/4$ is actually arbitrary. Any constant bounded away from $1/2$ would be sufficient. What if $3/4$ is not a good enough probability? We can run algorithm A on x independently for k times, using fresh randomness. Then, we can take the median/majority result out of these k independent outputs of A. Using tools to bound the probability of a large deviation from the expectation of a random variable, we can show that the error probability will be $O(\exp(-k))$. The idea is that we would expect that $3k/4$ of the outputs to be correct and the probability that only less than half of them are correct will decrease exponentially with k.

3.3 The Minimum-Cut Problem

In this section, we will present a randomised algorithm for the minimum-cut problem.

Definition 3.12 (Cut in a graph) *A* cut *in graph G is a set of edges whose removal results in G being broken up into two or more pieces. Alternatively, we will denote the cut as a partition $(A, V \setminus A)$ of vertices V of G. The edges of cut $(A, V \setminus A)$ are those that have one endpoint in each part.*

Definition 3.13 (Min-Cut Problem) *Given an undirected graph with unit edge weights, find a cut with minimum cardinality: find nonempty $A \subseteq V$ to minimise edges between A and $V \setminus A$.*

Two attempts:

1. Replace each undirected edge with two directed edges in opposite directions. For all s, t, compute max flows (n^2 max flows).
2. Pick s, for all other v, look at $s - v$ cuts (n max flows).

Next, we will show how a simple randomised algorithm can solve this problem very efficiently. It will be useful to work with graphs that can have multiple edges between any pair of nodes (but no loops).

Definition 3.14 (Multigraph) *A* multigraph *may contain multiple edges between any pair of vertices.*

Our algorithm will use a simple edge contraction operation.

Definition 3.15 (Contraction of an edge) Contraction of an edge *combines the end points of the edge.*

More specifically, when edge (u, v) is contracted

- vertices u and v are combined into a "supernode" w,
- edge (u, v) (and all edges parallel to it) is deleted,
- edges (u, x) and (v, x) for all $x \neq u, v$ are replaced by (w, x), retaining multiple edges.

Note that an edge contraction could possibly lead to self loops, when one of multiple parallel edges is contracted. Self loops can be discarded during the execution of the algorithm as they will never be in a cut.

Observation 3.16 *An edge contraction does not reduce the min-cut size.*

Below is a description of Karger's randomised minimum cut algorithm, which repeatedly contracts random edges until only two supernodes remain and output the partition corresponding to the vertices that make up each supernode. The hope is that the algorithm ends up with a minimum cut after missing the edges of that cut for each contraction.

Algorithm 1 Karger's randomised minimum cut algorithm

```
1: Algorithm RANDOMISED MIN-CUT($G$)
2:     while there are more than 2 supernodes in $G$ do
3:         Pick an edge $(u, v)$ in $G$ uniformly at random
4:         Contract $(u, v)$ (retaining the multiple edges between the nodes)
5:     Output the cut between the remaining two supernodes
```

Theorem 3.17 *Karger's algorithm outputs a minimum cut with probability at least* $\frac{2}{n^2}$.

Proof. Fix a minimum cut C of size k in graph G. Given the minimum cut size is k, the minimum degree is at least k. Hence, the graph has at least $kn/2$ edges. Hence, for the first contraction, the probability that one of the edges of C is contracted is at most $2/n$. Now, given that the first contraction has not destroyed C, we can argue analogously as above, that the number of remaining edges in the graph is $k(n-1)/2$ and the probability of contracting an edge of C in the second iteration is at most $2/(n-1)$. Following the same argument, given that the first $i-1$ contractions have not destroyed C, the probability that the ith contraction destroys cut C is at most $2/(n-i+1)$. Hence, the probability that cut C survives $n-2$ edge contractions is at least

$$\prod_{i=1}^{n-2}\left(1-\frac{2}{n-i+1}\right)=\prod_{i=1}^{n-2}\left(\frac{n-i-1}{n-i+1}\right)=\frac{(n-2)(n-3)\cdots 1}{n(n-1)\cdots 3}=\frac{2}{n(n-1)}>\frac{2}{n^2}.$$

The product above is obtained using Eq. (1) with E_i defined as the event that ith contraction does not destroy C. □

The running time of Algorithm 1 is $O(n^2)$, because there are $n-2$ contractions, each of which can take $O(n)$ time for merging the neighbours of the endpoints of the contracted edge.

Clearly, $2/n^2$ is a low probability of success. However, we can remedy the situation and increase the error probability by repeating the algorithm with fresh random choices each time. Suppose we repeat the algorithm n^2 times with independent random choices. Then, the probability that a min-cut is not found in any of the n^2 attempts is

$$\left(1-\frac{2}{n^2}\right)^{n^2}<\frac{1}{e^2}<\frac{1}{4},$$

where the first inequality follows from that $1-x<e^{-x}$ for all $x>0$. This error probability can further be reduced to any value δ by using $\ln(1/\delta)\cdot n^2/2$ attempts. For instance, to bound the error probability by $\delta=\frac{1}{n}$, $n^2\cdot\ln(n)/2$ attempts suffice. Thus, we can achieve an error probability of at most $1/n$ with a running time of $O(n^4\log n)$.

3.4 Some Useful Tail Inequalities

In the analysis of randomised algorithms, bounding the deviation of a random variable from its expectation is often useful. In this section, we state three tail inequalities that are most commonly used for these purposes.

Theorem 3.18 (Markov's Inequality) *Let Y be a random variable taking only non-negative values. Then, for all $t \in \mathbb{R}^+$,*

$$\Pr[Y \geq t] \leq \frac{\mathbb{E}[Y]}{t}.$$

Equivalently,

$$\Pr[Y \geq k \cdot \mathbb{E}[Y]] \leq \frac{1}{k}.$$

Note that Markov's Inequality requires little information about the random variable, but does not always provide a strong enough bound. When more information such as a bound on the variance of the random variable is available, the following inequality will serve better.

Theorem 3.19 (Chebyshev's Inequality) *Let X be a random variable with expectation μ_X and standard deviation σ_X. Then, for any $t \in \mathbb{R}^+$,*

$$\Pr[|X - \mu_X| \geq t \cdot \sigma_X] \leq \frac{1}{t^2}.$$

Finally, a common scenario in the analysis of randomised algorithms occurs when we have a collection of independent 0-1 random variables and we would like to bound the probability that the sum of these random variables deviates much from the expectation.

Theorem 3.20 (Chernoff Bounds) *Let $X_1, X_2, \ldots, X_n$ be independent Poisson trials such that, for $1 \leq i \leq n$, $\Pr[X_i = 1] = p_i$, where $0 < p_i < 1$. Then, for $X = \sum_{i=1}^n X_i$, $\mu = \mathbb{E}[X] = \sum_{i=1}^n p_i$, and any $\delta > 0$,*

$$\Pr[X > (1+\delta)\mu] < \left[\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right]^\mu,$$

for $0 < \delta \leq 1$,

$$\Pr[X > (1+\delta)\mu] < \exp(-\mu\delta^2/3)$$
$$\Pr[X < (1-\delta)\mu] < \exp(-\mu\delta^2/2).$$

for $\delta > 2e - 1$,

$$\Pr[X > (1+\delta)\mu] < 2^{-(1+\delta)\mu}.$$

3.5 A Randomised Oblivious Routing Algorithm

We will now see an example of a randomised algorithm whose analysis makes use of one of the above Chernoff bounds. The problem is defined as follows. Consider the network defined by the n-dimensional hypercube: i.e., the vertices of the network are the strings in $\{0,1\}^n$, and edges connect pairs of vertices that differ in exactly one bit. We shall think of each edge as consisting of two links, one in each direction. Let $N = 2^n$ be the number of vertices. Now let π be any permutation on the vertices of the cube. The goal is to send one packet from each i to its corresponding $\pi(i)$, for all i, simultaneously.

This problem can be seen as a building block for more realistic routing applications. A strategy for routing permutations on a graph can give useful inspiration for solving similar problems on real networks.

We will use a synchronous model: i.e., the routing occurs in discrete time steps, and in each time step, one packet is allowed to travel along each (directed) edge. If more than one packet wishes to traverse a given edge in the same time step, all but one of these packets are held in a queue at the edge. We assume any fair queuing discipline (e.g., FIFO).

The goal is to minimise the total time before all packets have reached their destinations. A priori, a packet only has to travel $O(n)$ steps (the diameter of the cube). However, due to the potential for congestion on the edges, it is possible that the process will take much longer than this as packets get delayed in the queues.

Definition 3.21 *An* oblivious strategy *is one in which the route chosen for each packet does not depend on the routes of the other packets. That is, the path from i to $\pi(i)$ is a function of i and $\pi(i)$ only.*

An oblivious routing strategy is thus one in which there is no global synchronisation, a realistic constraint if we are interested in real-world problems.

Theorem 3.22 (Kaklamanis Krizanc Tsantilas 90) *For any deterministic, oblivious routing strategy on the hypercube, there exists a permutation that requires $\Omega(\sqrt{N/n}) = \Omega(2^{n/2}/\sqrt{n})$ steps.*

This is quite an undesirable worst case. Fortunately, randomisation can provide a dramatic improvement on this lower bound.

Theorem 3.23 (Valiant Brebner 81) *There exists a randomised, oblivious routing strategy that terminates in $O(n)$ steps with very high probability (that is, with probability at least $1 - O(2^{-n})$).*

We now sketch this randomised strategy, which consists of two phases.

- In phase 1, each packet i is routed to $\delta(i)$, where the destination $\delta(i)$ is chosen uniformly at random.
- In phase 2, each packet is routed from $\delta(i)$ to its desired final destination, $\pi(i)$.

In both phases, we use "bit-fixing" paths to route the packets. In the bit-fixing path from vertex x to vertex y of the cube, we flip each bit x_i to y_i (if necessary) in the left-to-right order. For example, a bit-fixing path from $x = 0011001$ to $y = 1110001$ in the hypercube $\{0,1\}^7$ is

$$x = 0011001 \to 1011001 \to 1111001 \to 1110001 = y.$$

Bit-fixing paths are always shortest paths.

Note that δ is not required to be a permutation (i.e., different packets may share the same intermediate destination), so this strategy is oblivious.

This strategy breaks the symmetry in the problem by simply choosing a random intermediate destination for each packet. This makes it impossible for an adversary with knowledge of the strategy to engineer a bad permutation. In our analysis, we will see that each phase of this algorithm takes only $O(n)$ steps with high probability. In order to do this, we will take a union bound over all 2^n packets, so we will need an exponentially small probability of a single packet taking a long time. This is where Chernoff bounds will be required.

For each packet, Phase 1 starts from a fixed source i and routes to a random destination $\delta(i)$. Phase 2 starts from a random source $\delta(i)$ and routes to a fixed destination $\pi(i)$. By symmetry, it suffices to prove that Phase 1 terminates in $O(n)$ steps, with high probability.

Let $D(i)$ be the *delay suffered by packet* i (in Phase 1). Then, the total time taken by packet i in Phase 1 is at most $n + D(i)$. We will prove, for some constant c, that

$$\forall i \quad \Pr[D(i) > cn] \leq 2^{-6n}. \tag{2}$$

Taking a union bound, it follows that

$$\Pr[\exists i \; D(i) > cn] \leq 2^n 2^{-6n} < 2^{-5n}.$$

All that remains is to prove the inequality in (2). For a single packet from i to $\delta(i)$, let $P_i = (e_1, e_2, \ldots, e_k)$ be the sequence of edges on the route taken by packet i. Let $S(i) = \{j \neq i \mid P_j \cap P_i \neq \emptyset\}$, the set of packets whose routes intersect P_i. The bulk of the proof lies in the following claim.

Claim 3.24 $D(i) \leq |S(i)|$.

Proof. First note that, in the hypercube, when two "bit-fixing" paths diverge they will not come together again; i.e., routes which intersect will intersect only in one contiguous segment. With this observation, we can now charge each unit of delay for packet i to a distinct member of $S(i)$.

Definition 3.25 *Let j be a packet in $S(i) \cup \{i\}$. The lag of packet j at the start of time step t is $t - l$, where e_l is the next edge that packet j wants to traverse.*

Note that the lag of packet i proceeds through the sequence of values $0, \ldots, D(i)$. Consider the time step t when the lag of packet i goes from L to $L + 1$. Suppose this happens when i is waiting to traverse some edge e_l. Then, packet i must be held up in the queue at e_l (else its lag would not increase), so there exists at least one other packet at e_l with lag L, and this packet actually moves at step t.

Now consider the last time at which there exists any packet with lag L; say, this is time t' (hence, that particular packet wants to traverse edge $e_{t'-L}$ but it cannot). Then, some other packet at edge $e_{t'-L}$ must leave path P_i (or reach its destination) at time t', because, otherwise, it would have retained the same lag L if it remained on path P_i, which contradicts the choice of t'. So, we may charge this unit of delay for packet i (from lag L to lag $L + 1$) to that packet (leaving the path at time t'). Each packet is charged at most once, because it is charged only when it leaves path P_i (which, by the observation at the start of the proof, happens only once). Note that we charged the increase of the lag of packet i from L to $L + 1$ at time step t to a packet that leaves path P_i at time step $t' \geq t$. □

Now, we can prove the probability bound in (2).

We define

$$H_{ij} = \begin{cases} 1 & P_i \cap P_j \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

By Claim 3.24, $D(i) \leq \sum_{j:j \neq i} H_{ij}$. And, since the H_{ij}'s are independent, we can use a Chernoff bound to bound the tail probability of this sum. First, we need a small detour to bound the mean, $\mu = \mathbb{E}[\sum_{j:j \neq i} H_{ij}]$ (since the expectation of the H_{ij}'s are tricky to get hold of).

Define

$$T(e_l) = \text{the number of paths } P_j \text{ that pass through edge } e_l \text{ (on path } P_i).$$

Then, by symmetry and using the fact that the length of the path from i to $\delta(i)$ is expected to be of $n/2$,

$$\mathbb{E}[T(e_l)] = \frac{\mathbb{E}[\text{the total length of all paths}]}{\text{the number directed edges in the network}} = \frac{N(n/2)}{Nn} = 1/2.$$

So, the expected number of paths that intersect P_i is

$$\mathbb{E}[\sum_{j:i\neq j} H_{ij}] \leq k/2 \leq n/2.$$

Note that we used the fact that $\mathbb{E}[T(e_l)]$ does not depend on l, which follows by symmetry. (One might think this is false, because we chose a particular bit ordering. However, the ordering does not matter because for every source and every i, with probability 1/2, we choose a sink such that the bit-fixing algorithm flips the ith bit.)

We can now apply a Chernoff bound that, for $\beta > 2e - 1$,

$$\Pr[D(i) \geq (1+\beta)\mu] \leq \Pr[\sum_{j:j\neq i} H_{ij} \geq (1+\beta)\mu] \leq 2^{-(1+\beta)\mu},$$

where $\mu = \mathbb{E}[\sum_{j:j\neq i} H_{ij}]$. Note that we only have an upper bound on μ. Nevertheless, due to a monotonicity argument (left to the reader), we can plug in our upper bound $\mu \leq n/2$ and conclude that

$$\Pr[D(i) \geq 6n] \leq 2^{-6n}.$$

Note that we needed to confirm the bound on $\mathbb{E}[\sum H_{ij}]$ to make sure that we can use the particular version of the Chernoff Bounds that applied to only large enough deviation.

Finally, using a union bound as indicated earlier, we see that all packets reach their destinations in Phase 1 in at most $n + 6n = 7n$ steps with probability at least $1 - 2^n 2^{-6n} = 1 - 2^{-5n}$. The complete algorithm (Phases 1 and 2) thus terminates in at most $14n$ steps. (To keep the phases separate, we assume that all packets wait until time $7n$ before beginning Phase 2.) This completes the proof (of Theorem 3.23) that the randomised oblivious strategy routes any permutation in $O(n)$ steps with high probability: namely, with probability at least $1 - 2 \cdot 2^{-5n}$.