CS 388R: Randomized Algorithms Fall 2015

Lecture 1 — Aug. 26, 2015

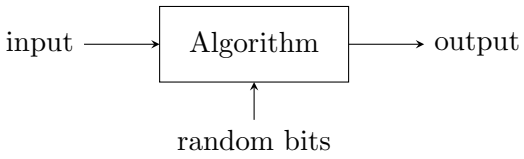
Prof. Eric Price *Scribe: Ashish Bora, Jessica Hoffmann*

1 Overview

In this lecture, we get a first taste of randomized algorithms, from definition to conception. We'll see different types of such algorithms, as well as two examples, QuickSort and Karger's Min-Cut.

2 What are randomized algorithms?

Randomized algorithms use randomization, which means they usually do something different if you use it different times.



Deterministic: For all inputs, output is good.

Random: For all inputs, with good probability, output is good.

Average case: Define a distribution over inputs, the algorithm works with good probability.

Issue: You usually don't have the distribution over the inputs, and we might lose structure by imposing a distribution. For instance, almost all matrices are invertible, but neglecting all non-invertible matrices might be super stupid for a given problem. Or okay. It depends.

Examples of famous random algorithms:

- Quicksort
- Bloom filters
- Cuckoo hash, and hash tables in general
- Routing packets

3 Why randomize?

The goals are simplicity, speed and sometimes other objectives depending on the problem (e.g., for Nash Equilibrium, the problem inherently requires randomization).

Themes:

- Avoiding adversarial inputs ("adversarial" may mean well-structured, i.e. natural). Randomization makes your data look random, even if it is not (think Quicksort).
- Finger printing (have a quick identifier for a complex structure)
- Load balancing (distribute workload across resources to optimize time consumption, or efficiency, or avoid overloads)
- Sampling (estimate properties of the whole population based on a subset)
- Symmetry breaking (e.g. what to do when two packets arrive at the same time?)
- Probabilistic existence proofs (Erdös, Lovasz... if a random object satisfies a property with non-zero probability, we just proved that such an object exists)

4 Quicksort

Algorithm 1 You all know Quicksort. It's for sorting, but quickly.

```
1: function QSORT(List)
2:     if List == [] then
3:         return []
4:     Choose t ∈ [n]
5:     return Qsort([x_i ∈ List | x_i < x_t]) + [x_t] + Qsort([x_i ∈ List | x_i ≥ x_t])
```

Time $\propto \#(i,j)$ that are compared.
Let us number the elements such that $x_i \leq x_j$ whenever $i < j$. Let $Z_{i,j}$ be the event that i and j are compared i.e. $Z_{i,j} = 1$ if i and j are compared, 0 otherwise.

Notation: $a \lesssim b \iff \exists c \in \mathbb{R}, a \leq c \cdot b$

Time $\lesssim \sum_{i<j} Z_{i,j}$. What is $\mathbb{P}[Z_{i,j} = 1]$?
$Z_{i,j}$ represents the event that i or j is chosen before $i+1, i+2, \cdots, j-1$.

$$
\begin{aligned}
\mathbb{P}[Z_{i,j}] &= \frac{2}{|j-i|+1} = \mathbb{E}[Z_{i,j}] \\
\mathbb{E}[Time] &\lesssim \mathbb{E}\left[\sum_{i<j} Z_{i,j}\right] \\
&= \sum_{i<j} \mathbb{E}\left[Z_{i,j}\right] \\
&= \sum_{i} \sum_{i<j} \mathbb{E}\left[\frac{2}{j-i+1}\right] \\
&= 2 \cdot \sum_{i} \mathbb{E}\left[\frac{1}{2} + \cdots + \frac{1}{n-i+1}\right] \\
&\leq 2 \cdot n \cdot (H_n - 1) \\
&\leq 2 \cdot n \cdot \log n
\end{aligned}
$$

Comment: All of that is possible because the expectation is linear. Yeah, you're all blasé, you know that already. But look, $\sum Z_{i,j}$ is a super complex variable, many $Z_{i,j}$'s are highly dependent on one another, and by using the expectation, suddenly we can treat them all separately. Don't be unimpressed, this is awesome!

5 Karger's minimum cut ('93)

Given a graph $G = (V, E)$ with n vertices and m edges, find a global minimum cut, therefore a set of vertices $S \subset V$, with $1 \leq |S| \leq n-1$, which minimizes the number of edges going from this subset S to the rest of the vertices. We denote the cut value for set S by $|E(S, \bar{S})|$ (where $\bar{S} = V \setminus S$).

The problem of finding a minimum cut with a given source and sink ($s-t$ min cut) is probably more familiar. Dinic's algorithm or Ford-Fulkerson algorithm lead to a solution. We can compute a $s-t$ minimum cut in $\mathcal{O}(n^3)$. By running this algorithm for all (s,t) pairs, we get a solution for the global minimum cut in $\mathcal{O}(n^5)$.

Algorithm 2 It looks stupid, but actually it's not *that* stupid.

```
1: function ALGO1
2:     while n > 2 do
3:         Choose an edge e = (u, v) randomly from the remaining edges
4:         Contract that edge
5:     return the two last vertices
```

Contracting an edge means that we remove that edge and combine the two vertices into a super-node. We note that self-loops thus formed are removed but any resulting parallel edges are *not* removed. This means at every step, we have a multi-graph without any self-loops.

This algorithm will yield a cut. The minimum cut? Probably not. But maybe! If the graph is somewhat dense, then choosing an edge in the minimum cut is rather unlikely at the beginning...and

very likely at the end.

Here is a successful run of this algorithm, courtesy to Wikipedia.

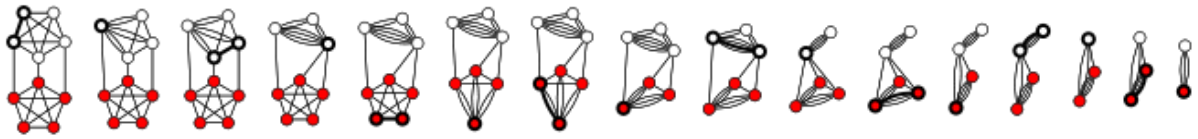


Figure 1: A (very) lucky run [1].

Let's say that the algorithm "fails" in step i if in that step, you choose to contract an edge in the minimum cut (here we suppose that there is only one minimum cut). For all $u \in V$, let $d(u)$ be the degree of u.

$$|E(S, \bar{S})| \leq \min_u d(u) \leq \frac{1}{n} \sum_{u \in V} d(u) \leq \frac{2 \cdot m}{n}$$

$$\frac{|E(S, \bar{S})|}{m} \leq \frac{2}{n}$$

We fail in the first step if we pick an edge in the minimum cut, an event that occurs with probability $\frac{|E(S, \bar{S})|}{m}$.

$$\begin{aligned}
\mathbb{P}(\text{fail in } 1^{st} \text{ step}) &\leq \frac{2}{n} \\
\mathbb{P}(\text{fail in } 2^{nd} \text{ step} \mid \text{success in } 1^{st} \text{ step}) &\leq \frac{2}{n-1} \\
&\vdots \\
\mathbb{P}(\text{fail in } i^{th} \text{ step} \mid \text{success till } (i-1)^{th} \text{ step}) &\leq \frac{2}{n-i+1}
\end{aligned}$$

Let Z_i be the event that the algorithm succeeds in step i. Thus, we have

$$\begin{aligned}
\mathbb{P}(Z_1) &\geq \frac{n-2}{n} \\
\mathbb{P}(Z_2|Z_1) &\geq \frac{n-3}{n-1} \\
&\vdots \\
\mathbb{P}(Z_i|Z_{i-1} \cap \cdots \cap Z_1) &\geq \frac{n-i-1}{n-i+1}
\end{aligned}$$

Therefore:

$$\begin{aligned}
\mathbb{P}(\text{Success}) &= \mathbb{P}(Z_1 \cap Z_2 \cap \cdots \cap Z_{n-2}) \\
&= \mathbb{P}(Z_1) \cdot \mathbb{P}(Z_2|Z_1) \cdot \cdots \cdot \mathbb{P}(Z_{n-2}|Z_1 \cap Z_2 \cap \cdots \cap Z_{n-3}) \\
&\geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \frac{n-5}{n-3} \cdot \ldots \cdot \frac{2}{4} \cdot \frac{1}{3} \\
&= \frac{1}{n} \cdot \frac{1}{n-1} \cdot \frac{2}{1} \cdot \frac{1}{1} \\
&= \frac{2}{n(n-1)} \\
&\geq \frac{2}{n^2}
\end{aligned}$$

If n is too large, this is a very poor guarantee. So what we do instead, is run the algorithm several times and choose the best min-cut we find. If we run this algorithm k times, we then have:

$$\mathbb{P}(\text{Success}) \geq 1 - \left(1 - \frac{2}{n^2}\right)^k$$

A classical inequality yields $(1-z)^{\frac{1}{z}} \leq \frac{1}{e}$. We hence choose to run the algorithm k times, with $k = \frac{n^2}{2} \log \frac{1}{\delta}$. We get:

$$\mathbb{P}(\text{Success}) \geq 1 - \left(\frac{1}{e}\right)^{\log \frac{1}{\delta}} = 1 - \delta$$

After running all these k trials, we choose the one with minimum cut value.

At every step of the algorithm, we have to contract an edge. This takes at most $\mathcal{O}(n)$ time. We do at most n such contractions. Thus for every run, we need $\mathcal{O}(n^2)$ time. We can choose δ to be, say $1/4$, and thus $k = \frac{n^2}{2} \log 4$ and this will ensure that within $\mathcal{O}(n^4)$ time the algorithm succeeds with probability at least $3/4$.

6 Types of randomized algorithms

Las Vegas Algorithm (LV):

- Always correct
- Runtime is random (small time with good probability)
- Examples : Quicksort, Hashing

Monte Carlo Algorithm (MC):

- Always bounded in runtime

- Correctness is random
- Examples : Karger's min-cut algorithm

It's easy to see the following:

1. LV $\implies$ MC. Fix a time T. If the algorithm terminates before T, we output the answer, otherwise we output 1.
2. MC does not always imply LV. The implications holds when verifying a solution can be done much faster than finding one. In that case, we test the output of MC algorithm and stop only when a correct solution is found.

7 Back to Min-Cut : Can we do better?

Initial stages of the algorithm are very likely to be correct. In particular, the first step is wrong with probability at most $2/n$. As we contract more edges, failure probability goes up. Moreover, earlier stages take more time compared to later ones.

Idea: Let us redistribute our iterations. Since earlier ones are more accurate and slower, why not do less of them at the beginning, and increasingly more as the number of edges decreases?

Let us formalize this idea. After t steps:

$$\mathbb{P}(Success) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-3}{n-1} \cdots \frac{n-t-1}{n-t+1} \approx \frac{(n-t)^2}{n^2}$$

We equate this to $\frac{1}{2}$ to get $t = n - \frac{n}{\sqrt{2}}$. Thus, we have the following algorithm:

Algorithm 3 Getting better

```
1: function ALGO2
2:     Repeat Twice:
3:         Run contraction from n → n/√2
4:         Recursively call Algo2 on n/√2 set
5:     return the last two vertices for the best cut among the two
```

Runtime analysis:

$$\begin{aligned} T(n) &= 2\left(\mathcal{O}(n^2) + T\left(\frac{n}{\sqrt{2}}\right)\right) \\ &= 2cn^2 + 2 \cdot 2 \cdot c \cdot \left(\frac{n}{\sqrt{2}}\right)^2 + \cdots \\ &= \mathcal{O}(n^2 \log n) \end{aligned}$$

Success Probability:

$$\begin{aligned} p(n) &= 1 - (1 - \textit{success in one branch})^2 \\ &= 1 - \left(1 - \frac{1}{2}\, p\left(\frac{n}{\sqrt{2}}\right)\right)^2 \\ &= 1 - \left(1 - \frac{1}{2}\, p\left(\frac{n}{\sqrt{2}}\right)\right)^2 \\ &= p\left(\frac{n}{\sqrt{2}}\right) - \frac{1}{4}\, p\left(\frac{n}{\sqrt{2}}\right)^2 \end{aligned}$$

To solve this recursion, we let $x = \log_{\sqrt{2}} n$. Thus, setting $f(x) = p(n)$, we get

$$f(x) = f(x-1) - f(x-1)^2$$

We observe that setting $f(x) = \frac{1}{x}$ gives:

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

Hence, we have $p(n) = \mathcal{O}\left(\frac{1}{\log n}\right)$. Thus, to get 3/4 probability of success, we need $\mathcal{O}(n^2 \log^2 n)$ time (which is much better than before, so we're happy).

References

[1] Thore Husfeldt. "Single run of Karger's Mincut algorithm." *Wikipedia*, N.p., 12 September 2012. Web. https://fr.wikipedia.org/wiki/Algorithme_de_Karger#/media/File:Single_run_of_Karger%E2%80%99s_Mincut_algorithm.svg