Randomized Algorithms 2022-3
Lecture 1

Introduction and the Min Cut Algorithm*

Moni Naor

The lecture introduces randomized algorithms. Why are they interesting? They may solve problems faster than deterministic ones, they may be essential in some settings, especially when we want to go to the sublinear time complexity realm[1]. They are essential in distributed algorithms e.g. for breaking symmetry. They yield construction of desirable objects that we do not know how to build explicitly and are essential for cryptography[2] and privacy[3]. Another type of study is to analyze algorithms when assuming some distribution on the input, or some mixture of worst case and then a perturbation of the input (known as *smoothed analysis*). But our emphasis would be worst case data where the randomness is created independently of it. That is we assume the algorithm or computing device in addition to the inputs gets also a random 'tape' (like the other tapes of the Turing Machine, but this one with truly random symbols).

One nice feature that some randomized algorithms have is that they may be simple. We demonstrated this in two algorithms (actually got only to see mostly min-cut algorithm in two versions).

Randomized algorithms existed for a long time, since the dawn of computing (for instance the numerical "Monte Carlo Method"[4]

The Minimum Cut Problem

The algorithm we saw demonstrates simplicity in a spectacular way. No need for flows, just pick a random edge and contract! The min-cut algorithm is due to Karger from SODA 1993 (the motivation was parallel algorithm). There is a faster version with Stein, where the repetition is done in a clever way (i.e. not starting from scratch each time), yielding a near $O(n^2)$ algorithm [1] (see notes by Nikolov on the algorithm [4]) and nearly linear in [2].

*These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. In the interest of brevity, most references and credits were omitted.

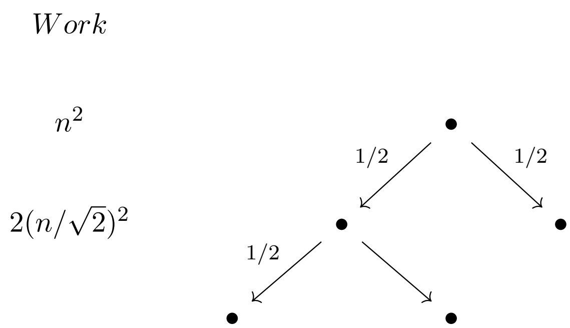
[1] For instance, the famed PCP Theorem, that states that every NP statement can be verified using a few queries *must* use randomness for picking the queries. Another area is property testing.

[2] Where no task is possible without good randomness

[3] Differential privacy is a notion for sanitizing data that involves necessarily randomization, e.g. adding noise to an aggregate of a population.

[4] Do not confuse with the term "Monte Carlo Algorithm" which is a general name for an algorithm whose running time is deterministic (usually polynomial) but may err.

The Karger-Stein algorithm essentially develops a tree as below, where each nodes makes two attempts on producing a smaller graph on $n/sqrt2$ nodes and calling it recursively. The total amount of work to produce a smaller graph is $O(n^2)$ (it gets a bit trickier when there are many parallel edges during the recursion). The total amount of work is therefore $O(n^2 \log n)$. The probability of success is $O(1/\log n)$.



Question: What happens if instead of picking a random edge you pick at random a pair of vertices and contract? Is the resulting algorithm a good min-cut algorithm?

The analysis of the algorithm had the form of analyzing the probability of a bad event in step i of the algorithm, given that a bad event had not occurred so far (the bad event was picking an edge from the cut). If that probability has an upper bound of P_i, then the probability of a bad event ever occurring is bounded by $\Pi_{i=1}^{n} P_i$. In this case $P_i = 1 - 2/(n - i + 1)$.

Question: The algorithm also showed a bound on the number of min-cuts, since for every min-cut the probability of outputting this specific cut was $1/n^2$. In contrast show that for s-t cuts (where there are two input nodes s and t and should be separated, there can be exponentially many min-cuts.

Another important idea we discussed is *amplification*. Given an algorithm which has some small probability of success, but running it many times, as a function of the probability, we can get a high probability of success. In this case the basic algorithm had probability $1/n^2$ of finding the min-cut, so after running it n^2 time and taking the best (smallest cut) we have probability $(1-1/n^2)^{n^2} \approx 1/e$. Repeating it a few more times gets us high probability of success.

Concentration bounds: Watch Ryan O'Donell's Lecture 5a on Markov and Chebychev's Inequality (and later the rest of lecture 5) from his course on a Theory Toolkit.
`https://www.youtube.com/watch?v=qqHHvOp5N6w`

References

[1] David Karger and Clifford Stein, *A new approach to the minimum cut problem.* Journal of the ACM 43 (4): 601, 1996.

[2] David Karger, *Minimum cuts in near-linear time.* J. ACM 47(1): 46-76 (2000)

[3] Jon Kleinberg and Eva Tardos, **Algorithm Design**. Addison Wesley, 2006. The relevant chapter 13.

[4] Aleksander Nikolov, The Karger-Stein Min Cut Algorithm, Advanced Algorithms Note 2, 2020. `http://www.cs.toronto.edu/~anikolov/CSC473W20/Lectures/Karger-Stein.pdf`

[5] Russell Impagliazzo and Avi Wigderson, *P = BPP if E Requires Exponential Circuits: Derandomizing the XOR Lemma,* STOC 1997, pp. 220–229.

[6] Dick Lipton's blog, "Rabin Flips a Coin", March 2009 `https://rjlipton.wordpress.com/2009/03/01/rabin-flips-a-coin/`

[7] Michael Oser Rabin, *Probabilistic algorithms*. In Algorithms and complexity: New Directions and Recent Results, pages 21 - 39. Academic Press, New York.

[8] Rene Schoof, *Four primality testing algorithms*, `http://arxiv.org/abs/0801.3840`