CS 3510 Algorithms Last updated: December 3, 2023

Lecture 27: Randomized Algorithms

Lecturer: Sahil Singla *Scribe(s): Joseph Gulian, Allen Chang*

The first part of this lecture is from Lecture 26; they're grouped together because we will not be focusing on this content in the final.

All the algorithms we've discussed so far, are fairly deterministic: running the algorithm on the same inputs will always yield the same outputs. This is due in part, to a lack of randomness; when using randomness, we sample from some distributions to make our decisions. This leads to the potential for making different decisions in different iterations of running the algorithm. This can be very useful because if we can speed up and simplify algorithms.

1 Max-Cut

Consider, for instance, the Max-Cut problem. Given a graph $G = (V, E)$, the goal is to find a set S such that we maximize

$$|E(S, V \backslash S)|$$

We can solve this simply by using the following algorithm: for each node, flip a coin, and if the coin is heads, add the node to S. It turns out that this is a 2-approximation of the maximum cut with high probability.

This is actually with $\frac{1}{2}$ chance, a 2-approximation of the true maximum cut. To show why this is 2-approximate, we will use the idea of linearity of expectation: $\mathbb{E}[X_1 + X_2 + \ldots + X_m] = \mathbb{E}[X_1] + \mathbb{E}[X_2] + \ldots + \mathbb{E}[X_m]$. [1] What we'll do is find the expectation that one edge is in the cut. Looking at it's endpoints, u, v there is a $\frac{1}{2}$ they're in the cut. Then there are a total of four options uv, Uv, uV, UV [2] , in half of the cases, the edge has one vertex in the subset and the other not in the subset.

$$\Pr[(u, v) \in \text{cut}(S, V \backslash S)] = \frac{2}{4} = \frac{1}{2}$$

We can use linearity of expectation to get

$$\mathbb{E}[\text{number of edges cut}] = \sum_{(u,v) \in E} \Pr[(u, v) \in \text{cut}(S, V \backslash S)] = \frac{|E|}{2}$$

Recall that in our previous algorithm, we did local search which, while bounded in runtime, could take a while; on the other hand, this is linear and performs about as well. If you were to perform 10 iterations of this, it would still be linear time, but you have a $\frac{1023}{1024}$ chance of finding a 2-approximate.

[1]This assumes that the distributions are independent.

[2]Using capital to represent being in the cut, so if u is capital, this implies $u \in S$, and the same for v.

You can see the power randomized algorithms give over deterministic algorithms from a design perspective. There's a conjecture that if there exists a randomized algorithm, for a problem, then there exists a similarly performant deterministic problem, but this is an open problem much like P vs NP.

2 Polynomial Identity Testing

Given two polynomials $f(x_1, x_2, \dots x_n)$ and $g(x_1, x_2, \dots x_n)$, this question asks whether or not these polynomials are the same. This could also be formulated as asking, if $h(x_1, x_2, \dots x_n) = f(x_1, x_2, \dots x_n) - g(x_1, x_2, \dots x_n)$, then if $h = 0$. This may seem obvious if the polynomial is given in a simple or low degree form like $x_1 \cdot x_2 + x_3$, though this is not always the case.

Consider a matrix's determinant which is the sum of a product of n variables and a sign. The precise details are not important, but working with this we will gain some understanding. So, suppose each entry in a matrix is a polynomial given in the form above with at most k terms. Is the determinant of the matrix always equal to zero?

For example given the matrix

$$\begin{pmatrix} 5x_1 & 0 & 7 \\ 2 & 3x_5 & 6 \\ -1 & 2x_1 & 5x_3 \end{pmatrix}$$

can we determine if the determinant is always zero. You may have learned that finding the determinant of a matrix can be done in polynomial time, but this only works if the matrix is filled with constants. However if the determinant has polynomial terms, it will take exponential time to find.

The most basic algorithm might just be to pick some point, evaluate each of the polynomials, and then solve the determinant with constants. If the result is zero, then say it is always 0; otherwise say it is not always 0.

This test actually succeeds with almost probability 1. [3] To give some intuition for this, consider the polynomial $(x-3)(x-2) = x^2 - 5x + 6$, what is the chance this evaluates to 0 out of all possible values of x. It is essentially zero because there are only 2 cases where the polynomial evaluates to zero, but infinite cases where it does not. This comes from the fact that any degree d polynomial has at most d real roots, meaning if you test over a good number of points, you know that the polynomial is very likely always zero.

This does not expand to more dimensions, if you think of a cone, it may have infinite roots, but this is actually very small. Although not finite, the space of roots is much smaller than the total space. Thus, the question of whether or not there exists a deterministic polynomial time algorithm for polynomial identity testing exists is open.

3 Wrap-Up

Now that we're at the end of this course, let's recap what this course was about. While this course was called "Design and Analysis of Algorithms", we focused mainly on the

[3]On real computers an issue arises where the bits of the real number are finite which causes some problems.

"Analysis" part of the algorithm. We care only about algorithms that are provably correct with a provable time guarantee.

1. Furthermore, this course also focused on the math and proof techniques that can be used to show ideas. The idea of Big O notation, coupled with ideas such as divide and conquer can be very powerful.
2. We also learned about graph algorithms, which have many real-world applications.
3. Then, we covered dynamic programming which is a powerful technique that can be applied in many algorithms to reduce its time complexity.
4. Next, we covered NP-Completeness, which introduced the general idea about the "difficulty" of algorithms. Approximate algorithms was introduced as a way to attack NP-Hard problems; while we may not be able to solve such problems perfectly, approximate algorithms give us a way to solve these problems partially.
5. Finally, the last algorithmic idea we covered was linear programming and randomized algorithms.

To conclude, there are a couple of classes that you can take if you're interested in algorithms:

1. CS 6515: Intro to Graduate Algorithms
2. CS 6550/8803: Advanced Algorithms and Uncertainty (Prof. Singla is teaching this next semester!)
3. CS 4540: Advanced Algorithms

4 Notes from the TAs

Hey! Thank you for taking this class and for a great semester–we really enjoyed getting to know you all in office hours. We hope you found these lecture notes helpful in exam preparation, and good luck in your future (and the final)!

Of course, if you have the time, please fill out the CIOS! We value student feedback a lot, and we're always looking for ways to improve.

- Joseph and Allen