Lecture 1

In which we describe what this course is about and discuss algorithms for the clique problem in random graphs.

1 About This Course

In this course we will see how to analyze the performance of algorithms (such as *running time*, *approximation ratio*, or, in the case of online algorithms, *regret* and *competitive ratio*) without resorting to worst-case analysis. The class will assume familiarity with basic combinatorics and discrete probability (as covered in CS70), linear algebra (as covered in Math54), and analysis of algorithms and NP-completeness (as covered in CS170). This course is based on on a course by the same name developed by Tim Roughgarden at Stanford, but our choice of topics will be slightly different.

A familiar criticism of the worst-case analysis of algorithms is that it can significantly overestimate the performance of algorithms in practice. For example, quicksort with a fixed pivot choice has worst-case quadratic time, but usually it runs faster than mergesort; hash tables with a deterministic hash function have worst-case linear time per operation, but they usually require constant time per operation; only quadratic-time algorithms are known for edit distance (and sub-quadratic worst-case performance is impossible under standard assumptions) but sub-quadratic running time occurs in practice, especially if one allows approximations; the simplex algorithm for linear programming has worst-case exponential running time in all known implementations but works well in practice; and so on.

In order to make a more predictive, non-worst-case, analysis of algorithms we need to first develop a model of the instances that we will feed into the algorithms, and this will usually be a probabilistic model. In this course we will look at models of various complexity, ranging from simple models involving only one or few (or even zero!) parameters, which are easy to understand but not necessarily a good fit for real-world instances, to more complex models involving a mix of adversarial and probabilistic choices.

We can roughly group the models that we will study in four categories.

1. *"Uniform" distributions.* In these models our input instances come from a sequence of i.i.d. choices. For example, in a problem like sorting, in which we are given a sequence of n elements from a universe Σ, we might look at the uniform distribution over Σ^n. In problems in which we are given an undirected graph over n vertices, we might look

at the $G_{n,p}$ distribution, in which each of the possible $\binom{n}{2}$ undirected edges exists with probability p, and choices for different edges are independent. In a problem in which the input is a $n \times m$ matrix, we might look at the distribution in which the entries are i.i.d. Gaussians. To generate an instance of k-SAT over n variable we might pick at random m of the $2^k\binom{n}{k}$ possible clauses, or choose independently for each possible clause whether to include it in the formula or not, and so on.

Although these models are very simple to describe, they often lead to deep and fascinating questions, and insights gained in these models can be stepping stones to analyses of more realistic models, or even worst-case analyses of randomized algorithms. For example, the analysis that quicksort with fixed pivot choice runs in expected $O(n \log n)$ time on random sequences naturally leads to an $O(n \log n)$ runtime analysis for quicksort with random pivot choice on worst-case sequences. An understanding of properties of random Gaussian matrices is critical to the smoothed analysis of the simplex, and an understanding of properties of $G_{n,p}$ random graphs is the starting point to develop algorithms for more realistic graph generative models, and so on.

2. *Planted-solution distributions.* In these models all choices are random as in (1), except that we force the instance that we produce to have a "solution" with a certain property. For example, in the "random graph with planted clique" problem with parameters n, p and k, we create an undirected graph on n vertices as follows: we choose a random set of k vertices, and add all edges between pairs of vertices in the set; all other edges are selected i.i.d., with each possible edge having probability p of being included. This distribution always creates graphs with a clique of size k, but it has several other properties in common with the $G_{n,p}$ model. In the "planted bisection" problem with parameters n, p and q, we first randomly split the vertices into two equal-size sets, then we add edges i.i.d, but edges with endpoints in the same set have probability p and edges with endpoints in different sets have probability q. If q is smaller than p, the cut defined by the initial partition will be much sparser than a random cut (or of any other cut, depending on how much q is smaller than p), but otherwise the model has a lot in common with the $G_{n,(p+q)/2}$ model.

   These models "break the symmetry" of i.i.d. models. While random fluctuations are the only source of structure in i.i.d. models, here we introduce structure by design. In planted-solution models it is interesting to see if an algorithm is able to find not just any good solution, but the particular solution that was created in the generative process. Usually, this is the case because, relying on our understanding of (1), we can establish that any solution that is significantly different from the planted solution would not be a near-optimal (or in some cases even a feasible) solution.

   These models capture problems studied in statistics, information theory and machine learning. Generally, if an existing algorithm that works well in practice can be rigorously proved to work well in a "planted-solution" model, then such a proof provides some insight into what make the algorithm work well in practice. If an algorithm is designed to work well in such a model, however, it may not necessarily work well in practice if the design of the algorithm overfits specific properties of the model.

3. *Semi-random distributions.* In these models we have a mix of probabilistic choices,

which might be of type (1) or type (2) and worst-case choices. For example, we may be generating a graph according to a distribution, often of type (2), and then allow an adversary to add or remove a bounded number of edges. In the opposite order, in the *smoothed analysis* of algorithms we may start from a worst-case instance, and then add a bounded amount of "noise," that is, make random changes.

Usually, performance in these models is a good predictor of real-world performance.

For an algorithm to perform well on semi-random graph models, the algorithm must be robust to the presence of arbitrary local structures, and generally this avoids the possibility of algorithms overfitting a specific generative model and performing poorly in practice.

In numerical optimization problems such as linear programming, the numerical values in the problem instance come from noisy measurements, and so it is appropriate to model them as arbitrary quantities to which Gaussian noise is added, which is exactly the model of smoothed analysis.

4. *Parameterized models.* When possible, we will endeavor to split our probabilistic analysis in two steps: first show that the algorithm works well if the instance has certain properties, possibly quantified by certain parameters, and then show that our probabilistic model produces, with high probability, instances with such properties. An advantage of this modular approach is that it allows steps of the analysis to be reused if one is looking at a new algorithm in the same generative model, or a different generative model for the same algorithm. Furthermore, one can validate the assumption that instances have certain properties on real-world data sets, and hence validate the analysis without necessarily validating the probabilistic model.

   (Note that here we are straining the notion of what it means to go "beyond worst-case analysis," since we are essentially doing a worst-case analysis over a subset of instances.)

We will see examples of all the above types of analysis, and for some problems like min-bisection we will work our way through each type of modeling.

We will study exact algorithms, approximation algorithms and online algorithms, and consider both combinatorial and numerical problems.

At the end of the course we will also do a review of average-case complexity and see how subtle it is to find the "right" definition of efficiency for distributional problems, we will see that there is a distribution of inputs such that, for every problem, the average-case complexity of the problem according to this distribution is the same as the worst-case complexity, and we will see some highlights of Levin's theory of "average-case NP-hardness," including the surprising roles that hashing and compressibility play in it.

The course will be more a collection of case studies than an attempt to provide a unified toolkit for average-case analysis of algorithms, but we will see certain themes re-occur, such as the effectiveness of greedy and local search approaches (which often have very poor worst-case performances) and the power of semidefinite programming.

2 Clique in Random Graphs

We will start by studying the Max Clique problem in $G_{n,p}$ random graphs, starting from the simplest case of the $G_{n,1/2}$ distribution, which is the uniform distribution over all $2^{\binom{n}{2}}$ undirected graphs on n vertices.

2.1 The typical size of a largest clique

A first fact about this problem, is that, with $1 - o(1)$ probability, the size of the maximum clique of a graph sampled from $G_{n,1/2}$ is $(2 \pm o(1)) \cdot \log n$ where the logarithm is to base 2. (This will be our standard convention for logarithms; we will use ln to denote logarithms in base e.)

We will not provide a full proof, but note that the expected number of cliques of size k in a graph sampled from $G_{n,1/2}$ is

$$\frac{1}{2^{\binom{k}{2}}} \cdot \binom{n}{k} \tag{1}$$

which is at most $2^{k \log n + \frac{k}{2} - \frac{k^2}{2}} = 2^{-\frac{k}{2} \cdot (k - 1 - 2 \log n)}$ and, if $k = 2 \log n + 2$, it is at most $2^{-\Omega(\log n)}$. By applying Markov's inequality, we get that there is a $1 - n^{-\Omega(1)}$ that a graph sampled from $G_{n,1/2}$ has a clique of size at most than $2 \log n + 1$.

On the other hand, (1) is at least

$$2^{-\frac{k^2}{2}} \cdot \left(\frac{n}{ek}\right)^k = 2^{k \log n - k \log k - k \log e - \frac{k^2}{2}} = 2^{\frac{k}{2} \cdot (2 \log n - 2 \log k - 2 \log 2 - k)}$$

and if, for example, we choose $k = 2 \log n - 10 \log \log n$, we see that the above quantity goes to infinity like $n^{\Omega(\log \log n)}$. Thus there is an expected large number of cliques of size $2 \log n - 10 \log \log n$ in a $G_{n,1/2}$ random graph. This is not enough to say that there is at least one such clique with probability tending to 1, but a second-moment calculation would show that the standard deviation of the number of cliques is small, so that we can apply Chebyshev's inequality.

2.2 The greedy algorithm

How about *finding* cliques in $G_{n,1/2}$?

Consider the following simple greedy algorithm: we initialize a set S to be the empty set, and then, while $V - S$ is non-empty, we add an (arbitrary) element v of $V - S$ to S, and we delete v and all the non-neighbors of v from V. When $V - S$ is empty, we output S.

The algorithm maintains the invariants that S is a clique in G and that all the elements of S are neighbors of all the elements of $V - S$, so the algorithm always outputs a clique.

Initially, the set $V - S$ has size n and S is empty and, at every step, $|S|$ increases by 1 and $V - S$, on average, shrinks by a factor of 2, so that we would expect S to have size

$\log n$ at the end. This can be made rigorous and, in fact, the size of the clique found by the algorithm is concentrated around $\log n$.

In terms of implementation, note that there is no need to keep track of the set $V - S$ (which is only useful in the analysis), and a simple implementation is to start with an empty S, scan the nodes in an arbitrary order, and add the current node to S if it is a neighbor to all elements of S. This takes time at most $O(n \cdot k)$, where k is the size of the clique found by the algorithm and one can see that in $G_{n,1/2}$ the expected running time of the algorithm is actually $O(n)$.

So, with $1-o(1)$ probability, the greedy algorithm finds a clique of size $\geq (1-o(1)) \log n$, and the maximum clique has size at most $(2+o(1)) \log n$ meaning that, ignoring low-probability events and lower-order terms, the greedy algorithm achieves a factor 2 approximation. This is impressive considering that worst-case approximation within a factor $n^{.99}$ is NP-hard.

Can we do better in polynomial time? We don't know. So far, there is no known polynomial time (or average polynomial time) algorithm able to find with high probability cliques of size $\geq 1.001 \log n$ in $G_{n,1/2}$ random graphs, and such an algorithm would be considered a breakthrough and its analysis would probably have something very interesting to say beyond the specific result.

2.3 Certifying an upper bound

Approximation algorithms with a worst-case approximation ratio guarantee have an important property that is lost in an average-case analysis of approximation ratio like the one we sketched above. Suppose that we have a 2-approximation algorithm for a maximization problem that, for every instance, finds a solution whose cost is at least half the optimum. Then, if, on a given instance, the algorithm finds a solution of cost k, it follows that the analysis of the algorithms and the steps of its execution provide a polynomial time computable and checkable certificate that the optimum is at most $2k$. Note that the optimum has to be at least k, so the certified upper bound to the optimum is off at most by a factor of 2 from the true value of the optimum.

Thus, whenever an algorithm has a worst-case approximation of a factor of r, it is also able to find upper bound certificates for the value of the optimum that are off at most by a factor of r.

This symmetry between approximate solutions and approximate upper bounds is lost in average-case analysis. We know that, almost always, the optimum of the Max Clique problem in $G_{n,1/2}$ is about $2 \log n$, we know how to find solutions of cost about $\log n$, but we do not know how to find certificates that the optimum is at most $4 \log n$, or even $100 \log n$ or $(\log n)^2$. The best known polynomial time certificates only certify upper bounds of the order $\Theta(\sqrt{n})$, with the difference between various methods being only in the multiplicative constant. There is also some evidence that there is no way to find, in polynomial time, certificates that most graphs from $G_{n,1/2}$ have Maximum Clique upper bounded by, say, $O(n^{.499})$.

We will sketch the simplest way of finding, with high probability, a certificate that the

Maximum Clique of a $G_{n,1/2}$ graph is at most $O(\sqrt{n})$. Later we will see a more principled way to derive such a bound.

Given a graph G sampled from $G_{n,1/2}$, we will apply linear-algebraic methods to the adjacency matrix A of G. A recurrent theme in this course is that A, with high probability, will "behave like" its expectation in several important ways, and that this will be true for several probabilistic generative models of graphs.

To capture the way in which A is "close" to its expectation, we will use the spectral norm, so let us first give a five-minute review of the relevant linear algebra.

If M is a symmetric $n \times n$ real valued matrix, then all its eigenvalues are real. If we call them $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, then the largest eigenvalue of M has the characterization

$$\lambda_n = \max_{\mathbf{x} \neq \mathbf{0}} \frac{\mathbf{x}^T M \mathbf{x}}{||\mathbf{x}||^2}$$

and the smallest eigenvalue of M has the following characterizations

$$\lambda_1 = \min_{\mathbf{x} \neq \mathbf{0}} \frac{\mathbf{x}^T M \mathbf{x}}{||\mathbf{x}||^2}$$

The largest eigenvalue in absolute value can be similarly characterized as

$$\max\{|\lambda_1|, \ldots, |\lambda_n|\} = \max\{-\lambda_1, \lambda_n\} = \max_{\mathbf{x} \neq \mathbf{0}} \frac{|\mathbf{x}^T M \mathbf{x}|}{||\mathbf{x}||^2}$$

The *spectral norm* of a square matrix is its largest singular value, and is characterized as

$$||M||_2 = \max_{\mathbf{x} \neq 0} \frac{||M\mathbf{x}||}{||\mathbf{x}||}$$

if M is symmetric and real valued, then $||M||_2$ is the largest eigenvalue in absolute value, so we have

$$||M||_2 = \max_{\mathbf{x} \neq 0} \frac{||M\mathbf{x}||}{||\mathbf{x}||} = \max_{\mathbf{x} \neq \mathbf{0}} \frac{|\mathbf{x}^T M \mathbf{x}|}{||\mathbf{x}||^2}$$

Furthermore, the spectral norm of a symmetric matrix can be determined up to $1+1/poly(n)$ approximation in polynomial time.

We have the following simple fact.

Lemma 1 *Let $G = (V, E)$ be a graph, A its adjacency matrix, J be the matrix all whose entries are 1, and k be the size of the largest clique in G. Then*

$$k \leq 2||A - J/2||_2 + 2$$

PROOF: Let $S \subseteq V$ be a clique of size k and let $\mathbf{1}_S$ be the indicator vector of S. Then

$$\mathbf{1}_S^T A \mathbf{1}_S = k^2 - k$$

and

$$\mathbf{1}_S^T J \mathbf{1} = k^2$$

so

$$\mathbf{1}_S^T (A - J/2)\mathbf{1}_S = \frac{k^2}{2} - k$$

Noting that $||\mathbf{1}_S||^2 = k$, we have

$$||A - J/2||_2 \geq \frac{|\mathbf{1}_S^T (A - J/2)\mathbf{1}_S|}{||\mathbf{1}_S||^2} = \frac{k}{2} - 1$$

□

Note that $J/2$ is essentially the average of A (to be precise, $J/2 - I/2$ is the average of A, but adding or subtracting $I/2$ changes the spectral norm by at most $1/2$) so it remains to show that A is usually close in spectral norm to its average. The following bound is known, and best possible up to the value of the constant c.

Lemma 2 *There is a constant c such that, with $1 - o(1)$ probability, if we sample G from $G_{n,1/2}$ and we let A be the adjacency matrix of G, we have*

$$||A - J/2||_2 \leq c\sqrt{n}$$

More specifically, it is known that with high probability we have $||A - J/2||_2 = (\sqrt{2} \pm o(1)) \cdot \sqrt{n}$.

Thus, with high probability, we can certify in polynomial time that a graph sampled from $G_{n,1/2}$ has Max Clique at most $O(\sqrt{n})$.