CS 270 Combinatorial Algorithms and Data Structures, Spring 2015

Lecture 1: Introduction

- Course overview and administration
- randomized min-cut

Course homepage: http://www.cse.cuhk.edu.hk/~chi/cs270

Course overview

In an undergraduate algorithms course, we learned how to use combinatorial techniques (e.g. greedy dynamic programming, local search) and data structures (e.g. heaps, balanced trees) to design and analyze fast algorithms for various problems (e.g. shortest path, maximum flow).

These techniques are very nice and insightful, and they are deterministic and always return optimal solutions.

In this course, we study more recent techniques in designing randomized algorithms and approximation algorithms. In a sense, they are worse algorithms because they may make mistakes and may not return optimal solutions, but in return they allow us to design faster and simpler algorithms, tackle NP-hard problems, and sometimes achieve something impossible for deterministic algorithms (e.g. sublinear algorithms).

The techiques involved are based on probability, linear algebra and continuous optimization.

These are the three main components in this course.

① Randomized algorithms:

Actually, there are more than twenty five years of active research in randomized algorithms, and probabilistic ideas are indispensable in the design and analysis of algorithms.

In the first part of the course, we will introduce basic tools such as concentration inequalities (e.g. Chernoff bound), probabilistic methods, random walks, Schwartz-Zippel lemma and see applications in graph sparsification (fast algorithms), data streaming (sublinear algorithms), Markov chain Monte Carlo method in sampling and counting (only know polytime

algorithms), network coding (distributed computing), etc.

② Linear programming and Semidefinite programming

These are general continuous optimization frameworks to design exact and approximation algos.

In the second part of the course, we will first see that a large class of combinatorial optimization problems can be modelled as LP problems, and extreme point solutions of those LPs are integral. Then we see that this method can be extended to design approximation algorithms for NP-hard variants of the problems.

We will also introduce semidefinite programming and see that they provide the best-known approximation algorithms for many problems.

We will study general methods to apply these optimization framework, such as duality theorems, primal-dual methods, and multiplicative weights update method, which have diverse applications beyond combinatorial problems.

③ Spectral analysis

More recently, linear algebraic ideas such as spectral techniques have been used to make surprising progress on fundamental combinatorial problems.

In the third part of the course, we will first see how eigenvalues and eigenvectors of the adjacency matrix can be used to design graph partitioning algorithms and analyze random walks.

Then, we will view graphs as electrical networks and see applications in graph sparsification and analysis of random walks.

Then, we will see that these ideas can be used to design faster algorithms for solving linear equations, which can then be used back to design faster algorithms for combinatorial problems. These show very interesting interactions between linear algebra and graphs.

Approach: Each of this topic can take the entire semester. Instead, we will take a broader view on the main ideas and won't try to see as many applications as possible or go into the most difficult results in these topics. I try to cover basic and general ideas useful to a broader audience. You're encouraged to go deep in one subject in the course project. I will

also try to provide pointers and references for further reading.

A motivating example

The above description of the course content may be a bit abstract, so let me illustrate it with a concrete example, the basic min s-t cut problem.

In a simple setting, we are given an unweighted undirected graph and two specified vertices s and t, and the goal is to remove a minimum number of edges to disconnect s and t.

In undergraduate algorithms, we learned the max-flow min-cut theorem, and a simple implementation of the augmenting path algorithm would solve the problem in $O(mn)$ time, where m is the number of edges and n is the number of vertices.

With additional combinatorial ideas (blocking flow, binary length function) and advanced data structures (dynamic trees), Goldberg-Rao gave a $O(\min\{m^{1.5}, mn^{\frac{2}{3}}\})$ time algorithm, which is deterministic and can be extended to the weighted case and directed graphs.

Then, it was not known how to improve on this result.

① Randomized algorithms: In dense graphs when there are $\Theta(n^2)$ edges, Goldberg-Rao's runtime is $O(n^{\frac{8}{3}})$.
Karger introduced the idea of random sampling, and showed that one can sparsify the graph into $O(n\log n)$ edges, while all the cuts' capacity are preserved within $(1+\varepsilon)$-factor.
This idea leads to an $\tilde{O}(n^2)$-time algorithm for undirected unweighted maximum flow.
We'll also see a randomized algebraic algorithm to solve graph matching in matrix multiplication time.

② Linear program: We will see the min s-t cut problem can be formulated as an LP, from which we can read off optimal integral solutions in the extreme point solutions.
The max-flow min-cut theorem is a special case of the LP duality theorem.
General techniques such as primal-dual method and multiplicative update method for LP can be used to design interesting algorithms for the problem (but they are not faster yet).
Interestingly, random walks can be used to design a very fast algorithm to turn a fractional flow solution into an integral solution.

Moreover, we can use LP to design approximation algorithms for NP-hard variants of the problem (e.g. multiway cut, sparsest cut, etc).

③ Spectral algorithms: Eigenvalues come in. They are useful for graph partitioning, graph sparsification, and analysis of numerical method.

Spielman-Teng showed how to combine them elegantly to design a very fast algorithm for solving SDD (symmetric, diagonally dominant) linear equations, which implies that one could compute quickly electrical flow.

Surprisingly, electrical flow combined with multiplicative update method or interior point method finally lead to an improvement of the Goldberg-Rao result!

We will see many of these results (or at least the main ideas) in this course, along with the applications of probabilistic and linear algebraic techniques to otle problems.

Administration

Background :- Undergraduate algorithms and discrete math (e.g. graphs, recursion, max flow, etc)
- First course in probability (random variables, Gaussian, expectation, variance, etc)
- First course in linear algebra (rank, linear dependency, determinant, eigenvalues, etc)

For basic probability, you can try out homework 0 (no need to submit) to check your knowledge. For other topics, you could check my previous courses on LP, SDP, randomness and spectral (all have links in the course homepage) to get a good idea of the course topics. Also, you could check out the assignments in those courses to get a very good idea on the length and difficulty of the homework assignments.

Requirements: assignments 50% (3 or 4 assignments)
course project 50% (tentatively 2 undergrads or 1 postgrad in one group)

Project: The purpose is for you to study a topic in depth. See a list of topics in the course page. You are encouraged to pick any topic close to your own (research) interests, but the topic must be theoretical, meaning that there are nice mathematical proofs.

The basic requirement is to read a topic in depth and write a survey-type report, but of course you are highly encouraged to include original work such as simplified proofs, unified approach, different perspectives, or new results.

Sometime in March/April, you will be asked to send me a project proposal. I will give you feedback, and you will submit a final report in the end of the semester.

Request: Try not to use electronic devices in lectures, especially playing games / browsing / facebook / email are strongly discouraged.

Reading or taking notes only are okay.

Randomized minimum cut

Okay, let's see the first algorithm in this course.

In the minimum cut problem, we are given an unweighted undirected graph $G=(V,E)$, and our objective is to find a minimum subset of edges $F \subseteq E$ so that $G-F$ is disconnected.

A simple observation is that a minimum cut is also a min s-t cut for some s,t. So, one can compute min s-t cut for all s,t and take a minimum one, and this naive algorithm requires n^2 calls of maximum flow.

Then, one observes that it is enough to just fix an arbitrary vertex as s and just compute min s-t cut for all possible t, and this gives an algorithm in n maxflow time.

For a while, it seems that this global min-cut problem is more difficult than min s-t cut.

Then, Matula (1987) gave a very nice algorithm that computes min-cut in $O(mn)$ time.

His new idea is to do a graph search and identify an edge that can be contracted and repeat.

Karger came up with the idea of random contraction and improved the runtime to $\tilde{O}(n^2)$, which is faster than computing min s-t cut. Eventually he gave a near-linear time $\tilde{O}(m)$ algorithm for the minimum cut problem, which is very interesting and surprising.

We will present an $O(n^4)$-time algorithm and mention how it can be improved to $\tilde{O}(n^2)$.

The algorithm is very simple.

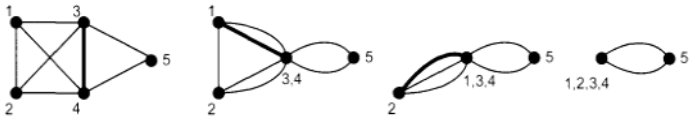
While there are more than two vertices in the graph

    Pick a random edge and contract the two endpoints

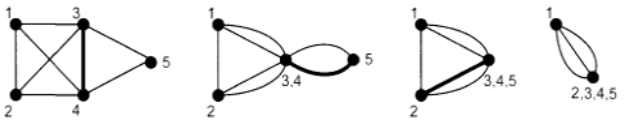
Output the edges between the remaining vertices.

By "contracting" an edge, we mean to identify the two endpoints as a single vertex.

See the picture below from the book by Mitzenmacher and Upfal.



(a) A successful run of min-cut.



(b) An unsuccessful run of min-cut.

Observation: Each vertex in an intermediate graph is a subset of vertices in the original graph.

So, each cut in an intermediate graph corresponds to a cut in the original graph.

Hence, a min-cut in an intermediate graph is at least a min-cut in the original graph.

Theorem The probability that the algorithm outputs a minimum cut is at least $2/n(n-1)$.

Proof Let F be a minimum cut and let $k = |F|$ be the number of edges in F.

If we never contract an edge in F until the algorithm ends, then the algorithm succeeds.

What is the probability that an edge in F is contracted in the i-th iteration?

By the observation, the min-cut value in the i-th iteration is at least k.

Note that it implies that every vertex is of degree at least k, as otherwise we can disconnect a single vertex from the graph by removing less than k edges.

Therefore, the number of edges in the i-th iteration is at least $(n-i+1)\cdot k/2$.

Since we pick a random edge to contract, the probability that we pick an edge in F is at most $k \Big/ \left((n-i+1)k/2\right) = \frac{2}{n-i+1}$.

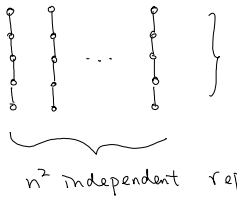
So, the probability that F survives until the end is at least

$$\left(1-\frac{2}{n}\right)\cdot\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{3}\right) = \frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdot\frac{n-4}{n-2}\cdots\frac{2}{4}\cdot\frac{1}{3} = \frac{2}{n(n-1)}. \quad \square$$

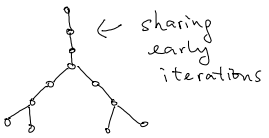
Improving success probability : By repeating the whole procedure t times, the failure probability is at most $\left(1-\frac{2}{n(n-1)}\right)^t$. By setting $t = 100n(n-1)$, this is at most e^{-200} which is tiny, using the inequality $1-x \le e^{-x}$.

Running time : One execution can be implemented in $O(n^2)$ time, and the total runtime is $O(n^4)$.

Improving running time: The observation is that the failure probability is only large near the end of one execution, while it is very small in the beginning. So, the idea is that we only repeat the later iterations but not the early iterations. Pictorically, the $O(n^4)$ algorithm is like



, while it is much better to do



You may see more details about this in homework 1.

Combinatorial Structure : An interesting corollary of the theorem is that there are at most $O(n^2)$ min-cuts in an undirected graph, because each min-cut survives with probability $\Omega\left(\frac{1}{n^2}\right)$ and the events that two different min-cut survive are disjoint. This is a non-trivial statement to prove using other arguments.

minimum k-cut The algorithm can be extended to give a $n^{O(k)}$ algorithm for finding a minimum k-cut, which is nontrivial to do by a deterministic algorithm.
Notice that the minimum k-cut problem is NP-hard if k is given as an input.

Open questions Can you design a fast algorithm for computing global vertex connectivity?