CS 466/666 Algorithm Design and Analysis . Spring 2020

Lecture 1: Introduction

We begin with the course overview and information, and then we study Karger's minimum cut algorithm.

Course overview

In a first algorithms course (e.g. CS 341), we learned different combinatorial techniques (divide and conquer, greedy algorithms, dynamic programming, local search, etc) and data structures (heaps, balanced trees) to design and analyze efficient algorithms for various problems (shortest paths, maximum flow).

These techniques are elegant and insightful.

When they work, they are deterministic and always return optimal solutions.

However, as we also learned in a first algorithms course, most combinatorial problems are NP-complete, and we don't know (and perhaps we don't expect) efficient algorithms to solve these problems optimally.

So, we need new techniques to cope with NP-complete problems.

Also, many of the fastest deterministic algorithms for polynomial time solvable problems are complicated and difficult to be implemented (shortest paths, minimum spanning trees, maximum flow).

It would be very nice to have simple and fast algorithms that can be implemented in practice.

In this course, we will learn more modern techniques in designing algorithms, using tools from probability theory, linear algebra, and continuous optimization.

These techniques will help us design faster and simpler algorithms for basic problems, approximation algorithms for NP-hard problems, and sometimes achieve something impossible for deterministic algorithms (e.g. sublinear algorithms).

① Randomized Algorithms:

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

In the first half 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 known polytime algorithms), network coding (distributed computing), parallel matching (parallel algorithms).

② Linear Algebra / Spectral Analysis

In the second part of the course, we will learn how to use concepts from linear algebra such as eigenvalues and eigenvectors to design and analyze algorithms.

Traditionally, eigenvalues and eigenvectors are useful in analyzing random walks and in designing algorithms for graph partitioning.

Through the analysis of random walks, we will see the interesting perspective of seeing a graph as an electrical network and use concepts such as effective resistances for graph problems.

Recently, this perspective has found surprising applications in graph sparsification and solving linear equations, that lead to faster algorithms for many combinatorial problems.

③ Linear Programming and Continuous Optimization

In the third part of the course, we will see that a large class of combinatorial problems can be modeled as a linear programming problem, using the concept of extreme point solutions.

Using linear programming, we will see examples of designing approximation algorithms for NP-hard problems.

Recently, this idea of solving combinatorial optimization problems using continuous optimization techniques have led to some breakthroughs, and we will try to see one of them.

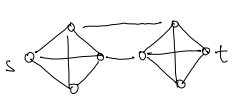
Approach: Each of this topic can take the entire semester. Instead, we will take a broader view on the main ideas and won't try to see as many applications as possible or go into the most difficult results in these topics. I try to cover basic and general ideas useful to a broader audience. I will also try to provide pointers and references for further reading.

Maximum Flow Minimum Cut : A Motivating Problem

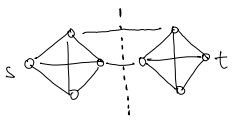
We are given an unweighted undirected graph $G=(V,E)$ and two specified vertices s and t.

The minimum s-t cut problem is to remove a minimum number of edges to disconnect s and t.

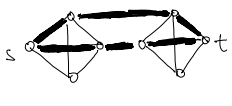
The maximum s-t flow problem is to find a maximum set of edge-disjoint paths from s to t.



graph



min s-t cut : 2 edges



max s-t flow : 2 paths

It is not difficult to see that the max s-t flow value is at most the min s-t value.

The famous max-flow min-cut theorem proves that the optimal values are always the same.

One way to prove this is to use the augmenting path algorithm (similar to the one for bipartite matching), and this provides an $O(|V||E|)$ time algorithm to solve both problems.

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

Then, for thirty years, no one knows how to improve this result.

In this course, we will look at this problem using different techniques.

Graph sparsification: Using random sampling or spectral method, we will see that any graph can be approximated by a sparse graph such that all the cut values are approximately the same.

This implies a faster approximation algorithm for the min s-t cut problem when the graph is dense.

Linear programming: The maximum flow problem can be formulated as a linear programming problem.

$$\begin{aligned} \max \quad & \sum_{e \in \delta^{out}(s)} f_e \\ \text{subject to} \quad & 0 \le f_e \le 1 && \text{// capacity constraints} \\ & \sum_{e \in \delta^{in}(v)} f_e = \sum_{e \in \delta^{out}(v)} f_e \qquad \forall v \neq \{s,t\} && \text{// flow conservation constraints} \end{aligned}$$

Here, we formulate the maximum s-t flow problem in directed graphs, where $\delta^{in}(v)$ denotes the set of incoming edges to v and $\delta^{out}(v)$ denotes the set of outgoing edges from v.

The problem on undirected graphs can be reduced to directed graphs (exercise).

We will learn techniques to prove that there is always an optimal integral solution (i.e. $f_e \in \{0,1\}\ \forall e$) to the above linear program, and to derive the max-flow min-cut theorem from linear programming duality.

Moreover, we will learn the recent algorithm to "round" an optimal fractional solution to an optimal integral solution in nearly linear time, using random walks!

Spectral techniques and continuous optimization: Recently, graph sparsification techniques and combinatorial ideas are combined to give a nearly linear time algorithm for computing "electrical flow".

Interestingly, using the multiplicative update method, this electrical flow solver can be used to give a fast approximation algorithm for computing maximum flow.

This is the first breakthrough that leads to an exciting line of active research that eventually improves the long standing Goldberg-Rao result.

We hope to 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 other problems.

Course information

The course outline is posted on the course homepage. Detailed information can be found there.

The following is a list of background knowledge that is required for this course.

- first course in discrete math (induction, contradiction, asymtotics, graphs).
- first course in algorithms (time complexity, recursion, reductions, basic data structures and algorithms).
- first course in probability (random variables, Gaussians, expectation, variance).
- first course in linear algebra (rank, linear independence, determinant, eigenvalues, orthonormal basis).

I encourage you to try out the practice problems in probability in the course homepage to test your skills in basic probability.

You can also take a look at the course page of CS 466/666 in Spring 2018 to get a good idea about the pace and the difficulty of the course.

Requirements: Homework 50% (5 assignments, see course outline for tentative deadlines)
Course project 50%
(No exams!)

Course project: The aim is to acquire a deeper understanding of a problem of your interest.
The project topic should be relevant, i.e. it is about design and analysis of algorithms, and the focus is mathematically oriented with proofs and theorems as this is the perspective that we take in this course.
The basic requirement is to read some papers and write a survey.
You will be asked to submit a project proposal by around midterm exam period, and I will give comments and suggestions.
Then you will submit a project report by the end of the assessment period (e.g. Aug 14).
Undergraduate students are allowed to work in a group of two students, while graduate students must work on their own.
We can discuss more about the project requirement in the course piazza page.

Randomized minimum cut [MU 1.4, MR 10.2]

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

Simple ideas: A simple observation is that a minimum cut is also a min s-t cut for some s,t.

Simple ideas: 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 minimum s-t cut (which can be solved in polynomial time).
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 min s-t cut time.

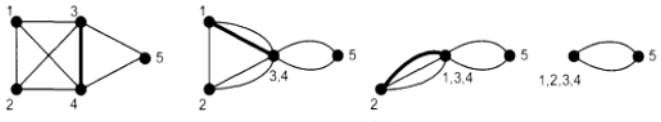
Deterministic algorithms: For a while, this global min-cut problem seems more difficult than min s-t cut.
Then, Matula (1987) gave a very nice algorithm that computes min-cut in $O(|V||E|)$ time.
His new idea is to do a graph search and identify an edge that can be contracted and repeat.

Randomized algorithms: Karger came up with the idea of "random" contraction and improved the runtime
to $\tilde{O}(|V|^2)$, which is faster than computing min s-t cut. Eventually, he gave a near-linear time $\tilde{O}(|E|)$
algorithm, for the minimum cut problem, which is very interesting and surprising.
Here, the $\tilde{O}$ notation hides some polylog factor in the runtime, e.g. $O(n^2 \log^3 n) = \tilde{O}(n^2)$.
We will present an $O(|V|^4)$-time algorithm and mention how it can be improved to $\tilde{O}(|V|^2)$.

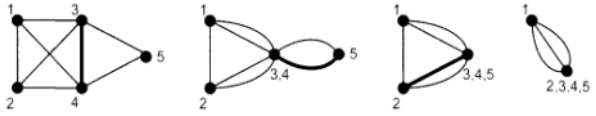
Karger's algorithm is very simple.

While there are more than two vertices in the graph
    Pick a uniformly 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, putting both on the same side.
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 as large as a min-cut in the

original graph.

Theorem The probability that the algorithm outputs a minimum cut is at least $2/n(n-1)$, where n is the number of vertices in the input graph.

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 still 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 $(n-i+1)\cdot k/2$.

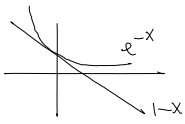
Since we pick a random edge to contract, the probability that we pick an edge in F is at most $k \Big/ \left((n-i+1)k/2\right) = \frac{2}{n-i+1}$.

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 A simple way is to repeat the whole procedure many times.

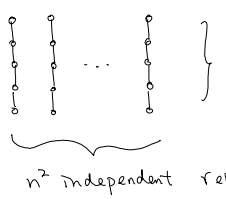
Suppose we repeat for t times. Then the failure probability is at most $\left(1-\frac{2}{n(n-1)}\right)^t$

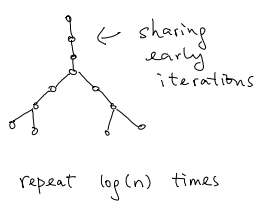
Recall that $1-x \le e^{-x}$, 

which can be derived from Taylor expansion (good to review some basic calculus, e.g. Stirling's approx)

Therefore, $\left(1-\frac{2}{n(n-1)}\right)^t \le e^{-\frac{2t}{n(n-1)}}$. So, if we set $k = 10\,n(n-1)$, then failure probability $\le e^{-20}$.

Running time: One execution can be implemented in $O(n^2)$ time (an exercise in data structures), and thus the total time complexity is $O(n^4)$.

Improving running time: The observation is that the failure probability is only large near the end of one executation, 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 

See more details in [MR 10.2]. This is called the Karger-Stein algorithm (1993).

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 $\Omega(\frac{1}{n^2})$ and the events that two different min-cuts 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.

Question: Can you modify the algorithm for the minimum s-t cut problem?

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

Remark: After we studied graph sparsification, we may discuss Karger's near linear time min-cut algorithm.

Quick basic probability review [MU, chapter 1 and 2]

Sample space Ω: set of all possible outcomes, each with a "probability" associated with it.

Event E: subset of outcomes. $Pr(E)$ = sum of the probabilities of its outcomes

Axioms: ① $0 \le Pr(E) \le 1$ ② $Pr(\Omega) = 1$ ③ $Pr(\bigcup_i E_i) = \sum_i Pr(E_i)$ for disjoint E_i.

Union bound: $Pr(\bigcup_i E_i) \le \sum_i Pr(E_i)$

Inclusion-exclusion principle: $Pr(\bigcup_i E_i) = \sum_i Pr(E_i) - \sum_{i,j} Pr(E_i \cap E_j) + \sum_{i,j,k} Pr(E_i \cap E_j \cap E_k) - \dots + (-1)^{l+1} \sum_{1 \le i_1 < i_2 < \dots < i_l} Pr(\bigcap_{r=1}^{l} E_{i_r}) \dots$

Conditional probability: $Pr(E|F) = Pr(E \cap F) / Pr(F)$

Independence: $Pr(\bigcap E_i) = \prod_i Pr(E_i)$

Total probability: Let E_i be disjoint and $\bigcup_i E_i = \Omega$.

Then $Pr(B) = \sum_i Pr(B \cap E_i) = \sum_i Pr(B|E_i) Pr(E_i)$.

Baye's law: Let E_i be disjoint and $\bigcup_i E_i = \Omega$.

Then $$Pr(E_j|B) = \frac{Pr(B|E_j) Pr(E_j)}{\sum_i Pr(B|E_i) Pr(E_i)}$$

Random variable X is a function from $\Omega \to \mathbb{R}$ $Pr(X=a) = \sum_{s \in \Omega : X(s)=a} Pr(s)$.

Independence X and Y are independent if and only if $Pr(X=x \cap Y=y) = Pr(X=x) \cdot Pr(Y=y)$.

Expectation $E[X] = \sum_i i \, Pr(X=i)$.

Linearity of expectation $E[\sum_i X_i] = \sum_i E[X_i]$.

It is important to note that this holds even for dependent variables, e.g. $E[X_i] + E[X_i^2] = E[X_i + X_i^2]$.

Conditional expectation $E[Y|Z=z] = \sum_y y \Pr(Y=y|Z=z)$

$E[Y|Z]$ is a random variable of Z that takes on the value $E[Y|Z=z]$ if $Z=z$.

Please review some basic random variables such as binomial and geometric random variables [MU 2.2-2.4].

Homework: ① Try out the practice exercises.

② Read chapter 1 and 2 of Mitzenmacher and Upfal to review the background.