Lecture 1

1 Overview

An admittedly simplistic view of undergraduate algorithms covers 2-3 decades perhaps to the nineties. It introduces analytical tools, basic algorithmic techniques, tends to cover cleanly stated problems such as minimum spanning trees, shortest paths, and maximum flow and with clever solutions. [1] The techniques tend to be combinatorial in nature.

In this course, we will cover ideas from the last 10-20 years of computer science. Some problems include dealing with large data, and modelling. The set of techniques used are probabilistic, algebraic, and continuous. These techniques are more applicable to today's problems, and closer to the cutting edge of research in algorithms.

1.1 Example Problems

One example (rather general) problem is clustering. This may be clustering points in high dimensional space which in turn may be representing documents, or individual's dna, or individual's preferences. It may also be graph clustering which also in turn is a basic subroutine for divide and conquer approaches to problems ranging from VLSI layout to parallel processing. It is also directly useful for example in image segmentation.

The problem is: given an image, find a region corresponding to a single object. This has been modelled as a graph problem where each pixel is a node, and edges and their weights indicate how closely related two pixels are; for example, a high weight indicating proximity as well as similarity of color. An object perhaps corresponds to a subset of vertices which can easily be separated from the others; for example, a red circle in a blue background. There are relatively few low weight edges connecting the pixels in the circle to the background. This has been formalized as finding the minimum normalized cut in a graph. That is, finding the subset S that minimizes

$$\frac{w(S,\overline{S})}{w(S)\cdot w(\overline{S}}))$$

where $w(S,\overline{S})$ is the weight of edges between S and $\overline{S}$ and $w(S)$ is the total weight of edges incident to S. The closely related quantity,

$$\frac{w(S,\overline{S})}{w(S)}$$

is perhaps more intuitive as chooses the cut with minimum cost per unit weight cut off; this is useful in many recursive schemes as it makes the most progress in the recursion with minimum costs.

[1] And, of course, they typically discuss dynamic programming and hashing which are oh so critical for those technical questions in Google or Microsoft interviews.

These problems and their variants are NP-complete, and techniques. The current best approach for solving this problem is to embed the vertices into some high dimensional space where the edges are short and the vertices are spread out. Then a hyperplane cut through the center of mass of the set of vertices will typically cut few edges and yet have many nodes on each side. Computing the optimal embedding uses ideas linear algebra and optimization. The analysis of the cutting process uses ideas from the area of approximation algorithms.

Another problem arises in recommendation systems. For example, given some information about a person's movie preference predict their preferences about other movies. This can be viewed as the matrix representation problem; the initial matrix has an entry for each person, movie pair. In fact, each person is perhaps a combination of a few types of people, and each movies is the combination of a few types of movies. Thus, the matrix is actually of low rank.

An example being that Sarah Palin may like True Grit, and dislike Black Swan and The Social Network, whereas Hillary Rodham Clinton may view things oppositely. We also happen to know their preferences for thousands of movies. The rest of us are (rather speculatively) part Hillary and part Sarah; that is, the set of people's preferences are rank 2. My preference would then simply be the combination of my inner Sarah and my inner Hillary. Of course, the situation would typically have more archetypes, and include noise (or even perhaps individuality), but this view can be quite useful to explain some amount of people's preference.

The big tool here has been termed principal components analysis which uses ideas from linear algebra.

1.2 Algorithmic Techniques

Some of the algorithmic techniques that are particularly relavent include.

- Sketching. Finding a small digest or core set of a large set of data which is useful for the problem at hand. One example is finding a sparse graph which represents a dense graph for the purposes of finding small cuts. Another is to process a large data stream into a summary of the data stream which can then be used to predict profit or devise strategies. The latter problems can have the added constraint that the data has to be analyzed as a stream.

- High Dimensional Geometry and Convexity. Gradient descent. This is a really inspired by using calculus and indeed constrained optimization tools you learned likely before you ever took an algorithms class.

- Techniques from Linear Algebra. Principal components analysis and computing the embeddings into space we discussed above both use the notion of eigenvalues and eigenvectors. Semidefinite programming which combines eigenvalue methods with linear programming methods have proven to be quite useful, of late.

- Dueling subroutines. One view of this is that when you try to solve a problem it is good to know when you are done. A bit more adversarially; one subroutine works to find better and better solutions while the other works to prove you can't do better.

> Remarkably, the evidence provided by each allows the other to do an increasingly better job.

Perhaps this last description is painfully vague, so we will illustrate the concept with an extended example below.

1.3 Course Assessment

Before proceeding, here is some information on assessment. The grading will be based on 5 homeworks (40 %) where collaboration, full credited, is allowed, 1 homework/midterm (25 %) where collaboration is prohibited, and a project (35 %). The project can be done in groups of 2 or 3 and should either be connecting your research to topics in this class, or digesting a topic of interest related to this class. The main output of the project is a report and a presentation to the course staff.

2 Dueling Subroutines: Congestion Minimization.

Given graph $G = (V, E)$ and pairs of nodes $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ we want to find paths connecting each pair such that the number of paths that use any one edge is minimized. One could define this problem as routing one unit of flow for each pair while minimizing the maximum flow on any edge. We refer to the number of paths (or the amount of flow) on the edge as the congestion. The problem is thus to route a path for each pair while minimizing the maximum congestion.

This problem has many applications. For example, VLSI routing, internet routing, travelling on roads, etc.

How do we solve this problem? Well, we can at least find a feasible solution by simply routing *some* path for each pair. Use something like depth first search or breadth first search (or shortest path) to choose such a path. But does this minimize the maximum congestion? To get started, let's just minimize the average congestion. Thinking for a moment, the average congestion is simply the sum of the path lengths divided by the number of edges:

$$\frac{\sum_i \ell(p_i)}{m}$$

where p_i is the path connecting the ith pair and $\ell(p_i)$ is the length of p_i and m is the number of edges. Thus, for the average congestion problem we can simply route along shortest paths; we use the minimum resources for each demand pair.

Moreover, the optimal solution for the average congestion problem is clearly a lower bound on the optimal value for the maximum congestion problem. Do we get close to the right answer with ths method?

Unfortunately, as seen by the example in Figure 1, the congestion is k if we route along the shortest paths but the optimal value of the congestion is 1. An approach to fix this would be to reroute when we congest edges too much. For example we could route along shortest paths, assign high tolls to congested edges, reroute along shortest paths with respect to the tolls, assign high tolls on congested edges and so on.

Let's consider some options for assigning tolls, a toll $f : E \to \mathbb{R}$ is an increasing function of the congestion. The first function to try is the identity function $f(e) = c_e$.

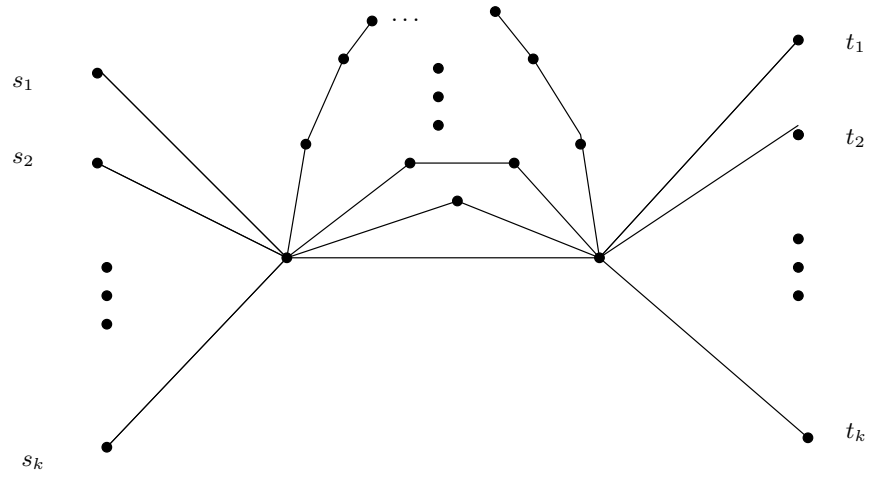

Figure 1: The shortest path method does not quite work.

We route along shortest paths with the weight of edge e being $f(e)$, the routing changes congestion on the edges and we re-route according to the new value of the congestion. For now, let us assume that the process has converged to an equilibrium i.e. routing along shortest paths with respect to weights $f(e)$ that are a function of the congestion c_e does not change the congestion on the edges. The issue of the time taken to converge to an equilibrium will be addressed later.

The equilibrium condition for the graph in figure 1 says that the lengths of all the k paths between u and v are equal under weights $f(e)$ are equal, else we would find an improved routing. If the common length is c_0 the congestion on the path of length i must satisfy $c_i i = c_0$ to make all the lengths equal. The total flow is k so we have,

$$c_0 \left(1 + \frac{1}{2} + \cdots + \frac{1}{k}\right) = k$$

The maximum congestion c_0 has been reduced slightly to $k/\ln k$ compared to the k obtained from the route along shortest paths strategy. If we used a quadratic function $f(e) = c^2$, the congestion on the path of length i satisfies $c_i^2 i = c_0$ showing that $c_i = \frac{c_0}{\sqrt{i}}$,

$$c_0 \left(1 + \frac{1}{\sqrt{2}} + \cdots + \frac{1}{\sqrt{k}}\right) = k$$

The bound on the maximum congestion c_0 improves to $\sqrt{k}$ for the quadratic function, this suggests that rapidly increasing functions lead to better bounds. Consider the exponential function, say $f(c) = 2^c$,

$$2^{c_0} = i.2^{c_i} \Rightarrow c_i = c_0 - \log i$$

Substituting in the congestion equation, we have $kc_0 - \log k! = k$, using the approximation $\log k! = k \log k - k$ we conclude that the maximum congestion is $O(\log k)$ a considerable improvement over routing along shortest paths.

The preceding discussion was for a specific example, in fact, for *any* congestion minimization problem, routing along shortest paths with respect to the exponential function, yields a congestion that is within $2\log m$ of the optimal. We introduce the problem dual to congestion minimization in order to prove this bound.

The dual problem (see lecture 7 for the precise meaning of duality and a derivation of the dual problem) is to assign weights w_e to the edges such that the total weight is 1. Let p_i denote the shortest path between (s_i, t_i) under the weights w_e. The objective is to maximize the sum of the lengths of the shortest paths between pairs (s_i, t_i).

$$\begin{aligned} &\max \sum_{i\in[k]} w(p_i) \\ w_e \geq 0, &\sum_{e\in E} w_e = 1 \end{aligned} \tag{1}$$

The maximum congestion for the metric w_e is greater that the value of the dual objective function,

$$\max_e c(e) \geq \sum_e w(e)c(e) = \sum_i w(p_i) \geq \sum_i w(s_i, t_i)$$

where p_i are the paths used in a particular routing: perhaps the optimal congestion routing. The key step is the equality in the middle step; here we "change the order of summation" which done more slowly goes as follows

$$\sum_i w(p_i) = \sum_i \sum_{e\in p_i} w(e) = \sum_e \sum_{p_i \ni e} w(e) = \sum_e w(e)c(e).$$

We use the fact that the weights provide a lower bound on the congestion, routing along shortest paths says the weighted average of the congestion is lower than this upper bound, and note that all the edges with congestion lower than $c_{\max} - 2\log m$ have at most $1/m$ of the total weight, and thus the average congestion is at least $c_{\max} - 2\log n$ (almost.) That is, the average congestion of the shortest path routing under exponential weights is a lower bound on the optimal and is within an additive $2\log m$ of the optimal.

This is laid out in the following derivation.

$$\begin{aligned} c_{opt} &\geq \sum_i w(s_i, t_i) \\ &= \sum_i w(e)c(e) \\ &= \frac{\sum_e 2^{c(e)}c(e)}{\sum_e 2^{c(e)}} \\ &> \frac{(c_{\max} - 2\log m)\sum_{e:c(e)>c_{\max}-2\log m} 2^{c(e)}}{(1+1/m)\sum_{e:c(e)>c_{\max}-2\log n} 2^{c(e)}} \\ &> (c_{\max} - 2\log n)/(1+1/m) \end{aligned}$$

That is,

$$c_{opt}(1+1/m) + 2\log m > c.$$

2.1 Convergence

We should point out that here we set up (carefully) a situation where the path routing and and resulting tolls were in equilibrium. In general, a toll function is not even sufficient, one actually needs to get an algorithm where the shortest path under the current tolls are stable. Indeed, a single path may oscillate between two choices.

This situation can be remedied in a variety of ways. One is to move a small fraction of the flow to new paths. Still, this moves the situation to a different regime. Another way is to send flow along approximately shortest paths. For the exponential function and for integer paths, perhaps only shift paths of length within a factor of two of optimal.

The difference now is that the first equality becomes an inequality where a factor of two is introduced.

That is,

$$c_{opt} \geq \sum_i w(s_i, t_i) \geq \frac{1}{2} \sum_e c(e)w(e).$$

This leads to a total bound of

$$2(1 + 1/m)c_{opt} + 2 \log m \geq c_{\max}.$$

By manipulating the exponential function to be $f(c) = (1 + \epsilon)^c$, and routing along $1 + \epsilon$ approximately, we get an expression like

$$(1 + \epsilon)c_{opt} + 2\frac{\log m}{\epsilon} \geq c_{\max}.$$

2.2 Wrap up.

To address the congestion minimization problem, we noted that the related average minimization problem seemed easier. That is, minimizing one constraint is easier than simultaneously dealing with many. Moreover, the average problem also provides a lower bound on the value of the real problem. We noted that different ways of weighting constraints also provided the lower bound and remained easy to solve.