U.C. Berkeley — CS270: Algorithms | Lecture 1
Professor Satish Rao | Jan 22,2013
Lecturer: Satish Rao | Last revised February 3, 2013

Lecture 1

These are slightly modified from previous years course notes.

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,

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

[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.

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.

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 relevant 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 source target pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ we want paths connecting each pair such that the maximum congestion on an edge is minimized. This problem has many applications. For example, VLSI routing, internet routing, travelling on roads, etc.

A feasible solution routes along *some* path for each (s_i, t_i) pair. Something like depth first search or breadth first search (or shortest path) can be used to find the paths. To get started, let's just minimize the average congestion. The average congestion is the sum of the path lengths divided by the number of edges:

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

here $\ell(p_i)$ is the length of path p_i connecting (s_i, t_i) and m is the number of edges. To minimize the average congestion we should route along shortest paths.

2.1 Illustrative example:

For the example in Figure 1, the optimal congestion is 1 but the congestion is k if we route along shortest paths. An approach to fix this is to reroute when we congest edges too much. Such an algorithm would route along shortest paths, assign high tolls to congested edges and iterate routing along shortest paths with respect to the tolls.

A toll $f : E \to \mathbb{R}$ is an increasing function of the congestion. The most obvious toll function to try is the identity $f(e) = c_e$. For now, let us assume that the process has converged to an equilibrium i.e. routing along shortest paths with respect to weights $f(e)$ does not change the congestion c_e on the edges. The issue of existence/convergence to an equilibrium will be addressed later.

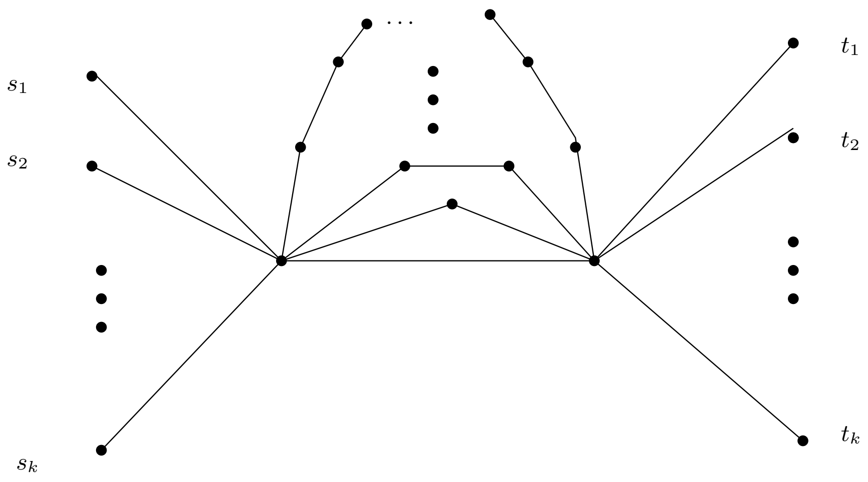

Figure 1: The shortest path method does not quite work.

All k paths in figure 1 have equal lengths in equilibrium, because if one path is shorter congestion changes by routing along it. If the common length is c_0 the congestion on the path of length i must satisfy $c_i i = c_0$ to make the path lengths equal. (For the concept of equilibrium, we assume that we are routing on unit of flow is routed among the pairs and the flow can be fractional.) The total flow is k so we have,

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

where c_0 units of flow are routed along the bottom path in the figure, $c_0/2$ on the next, $c_0/3$ on the next and so on.

The maximum congestion c_0 has been reduced 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 $f(c) = 2^c$,

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

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 to prove this bound.

2.2 The dual problem:

The problem dual to congestion minimization (the precise meaning of duality will be discussed later) is to assign weights w_e to the edges such that the total weight is 1. The objective is to maximize the sum of the lengths of the shortest paths between pairs (s_i, t_i).

$$\begin{array}{c} \max \sum_{i\in[k]} w(s_i,t_i) \\ \sum_{e\in E} w_e = 1, w_e \geq 0 \end{array} \tag{1}$$

The maximum congestion for any routing 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 the particular 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

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

Let c_{opt} be the optimal solution to the congestion minimization problem and c_{max} be the congestion for the equilibrium routing under tolls $f(e) = 2^{c(e)}$.

We note that edges an edge with congestion lower than $c_{\max} - 2\log m$ has weight at most $\frac{2^{c_{max}}}{m^2}$ and hence the total weight of such edges is at most $1/m$ fraction of the total weight. The following derivation shows that the shortest path routing under exponential weights is within an additive $2\log m$ of the optimal.

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

Rearranging,

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

2.3 Convergence

The argument above depends crucially on the fact that the path routing and and resulting tolls are in equilibrium. In general, one needs to get an algorithm where the shortest path under the current tolls are stable. Indeed, a single path may oscillate between two choices.

One way to obtain an algorithm for the general setting is the following:

(1) Route all the paths arbitrarily.
Let p_i denote the path connecting s_i and t_i.

(2) Assign $w(e) \propto 2^{c(e)}$, where $c(e)$ is the number of paths that use e.

(3) If there is a path where $w(p_i) \geq 3w(s_i, t_i)$, (where $w(s_i, t_i)$ is the length of the shortest path connected s_i and t_i), reroute along shortest path.

(4) Goto step (2).

The difference in analysis is that the first equality becomes an inequality where a factor of three is introduced.

$$c_{opt} \geq \sum_i w(s_i, t_i) \geq \frac{1}{3} \sum_i w(p_i) = \frac{1}{3} \sum_e c(e) w(e).$$

This leads to a total bound of

$$3(1 + 1/m) c_{opt} + 2 \log m \geq c_{\max}.$$

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

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

It remains to show that the process terminates after a finite number of steps, to prove this we introduce the potential function $\sum_e 2^{c(e)}$. If a path p with $w(p) = X$ is rerouted to a path with q with $w(q) \leq X/3$ then the potential function changes by $-X/2$ due to edges on p and at most $+X/3$ due to edges on q. The net change in the potential function $-X/6$ is negative, hence the process eventually terminates as the minimum potential is positve and the initial potential is finite. One can modify the algorithm and analysis to obtain a polynomial time bound on the algorithm's runtime.

2.4 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 lower bounds and remained easy to solve.