U.C. Berkeley — CS270: Graduate Algorithms | Lecture 19
Professor Satish Rao | November 6, 2006
Lecturer: Satish Rao | Last revised Scribe so far: Satish Rao

---

Lecture 19

1 Overview

Over the last few lectures, we have discussed the costs of communication and how to deal with it. For example, in the context of VLSI, a divide and conquer technique based on finding small cuts was used. In the last lecture, a low communication overhead embedding into the underlying network was assumed.

How do we obtain such things? For planar graphs (or data structures), one could use the Lipton-Tarjan theorem to find small bisections. Today, we will show one method for doing this in general.

2 Approximation Algorithm

We discussed previously a version of graph partitioning called bisection, i.e., the bisection with minimal number of edges crossing. This is, of course, an NP-complete problem as are many related problems. Today, we will describe a polynomial time "approximation algorithm" for this.

An $f(n)$-approximation algorithm is an algorithm that given an instance of size n, produces a solution whose cost is within a factor of $f(n)$ of that of the optimal problem.

For example, consider the problem of finding a maximum independent set. There is a trivial n approximation algorithm that just chooses one vertex. OK, stupid example. (On the other hand, for this example, there is reason to believe one cannot do too much better.)

3 Graph Partitioning

We will find cuts that minimize the following quantity.

$$\frac{\#edges(S, \overline{S})}{|S| \cdot |overlineS|}$$

This has been called the sparsest cut measure, or was called the normalized cut measure in the vision community. It is related to the edge expansion of the cut. That is,

$$\frac{\#edge(S, \overline{S})}{\min(|S|, \overline{|S}|).}$$

Since the bigger side of the cut always has between n and $n/2$ nodes, it is easy to see that a solution to the sparsest cut problem is a factor of two approximation to the minimum edge expansion problem. Indeed, an α-approximate solution to the sparsest cut is also a 2α-approximate solution to the minimum edge expansion cut.

4 Relaxations

A commonly used "relaxation" for the the sparsest cut problem is a "spectral" version. We will leave a discussion of this notion to the next lecture.

Today, we will discuss a linear programming relaxation of the problem. In particular, we consider the following problem: Route the maximal amout, f units, of flow between all pairs of vertices such that no edge carries more than one unit of flow.

Question 1: Write this problem out as a linear program. Hint, this is a multicommodity flow problem.

Now, consider a subset of nodes S. The amount of flow that needs to be routed across the associated cut is

$$f|S||\overline{S}|,$$

the number of edges that can be used to route flow across the cut is at most

$$\#edges(S, \overline{S}).$$

Thus, we have an upper bound on f. That is,

$$f \leq \frac{\#edges(S, \overline{S})}{|S| \cdot |\overline{S}|}.$$

Hey, the right hand side is the sparsest cut measure. This argument holds for any cut, so, f gives a lower bound on the sparsest cut ratio.

In fact, we will prove that computing f gives an upper bound as well. Specifically, we prove that f is $\Omega(\mathcal{S}/\log n)$, where $\mathcal{S}$ is the optimal value. Some of you might see the analogy here to the max flow-min cut theorem, which states that this relationship holds with equality for single commodity flows.

Ok, so now I must admit that this was a diversion. We will really work with the dual linear program, and could have spoken directly about it. The above has some historical context and intution that is nice, I hope.

In any case, the dual linear programming problem is the following.

$$\min \sum_e d(e).$$

$$\sum_{ij} d(i,j) \geq 1,$$

where $d(i,j)$ is the length of the shortest path from i to j in G under length function $d(\cdot)$. This may look familiar to you as the potential function we used in the routing of disjoint paths from lecture 10.

Note that for any cut, $(S, \overline{S})$, we can assign

$$d(e) = \frac{1}{|S| \cdot |S|}$$

and we have a feasible solution. Moreover, the cost of this solution is

$$\frac{\#edges(S, \overline{S})}{|S| \cdot |\overline{S}|}.$$

Yes, there exists a solution of value at most $\mathcal{S}$, the optimal of the sparsest cut.

We will now show that *no* solution has value less than $\frac{\mathcal{S}}{\log n}$.

In fact, we do something that is better.

LEMMA 1
There is a polynomial time algorithm, that given a solution of cost W to the above linear program, finds a cut whose sparseness is $O(W \log n)$.

This proves that every solution has cost $\Omega(\mathcal{S}/\log n)$. Thus, the value of the linear program gives an approximation to the value of the sparsity of the cut.

5 The algorithm and its proof.

We are given a solution to the "distance" LP on our graph and wish to find a small cut.

The crux of the algorithm is essentially to find a shortest path tree, and consider all cuts that one observes. Recall that in a shortest path algorithm, one has the set of nodes that one knows the distances to (already pulled from priority queue in Dijksta's) at any step. These are the observed cuts. Ok, so this might not quite work, but really it is all that we do. There are hiccups that we address later.

We proceed to prove a lemma about the shortest path cuts, and then discuss the full details.

5.1 The crux

We alter our solution to the linear program such that each node has a weight of W/n. Then, we prove the following lemma.

LEMMA 2
For the shortest path algorithm above, one of the cuts, $(S, \overline{S})$, has the property that

$$\#edges(S, \overline{S}) \leq 2n^2 W(S) \log n,$$

where $W(S)$ is the total weight of the linear program on edges incident to S and the nodes in S and $|S| \leq |\overline{S}|$.

PROOF: We prove the lemma by first mulitplying the weights of the edges by n^2. This ensures that there are two nodes that are at least 1 unit apart. We assume we capture less than 1/2 the nodes in the first 1/2 of this 1 unit. Further, we divide this 1/2 unit into $2 \log n$ intervals. That is, we consider balls of radius, $1/4 \log n, 2/4 \log n, \ldots ..$ $(r_0, \ldots, r_i, \ldots .)$

The weight of a ball is the weight of all the nodes in the ball plus the edges inside the ball, plus the appropriate portion of partially captured edges. That is, if there is a length $1.5/\log n$ edge incident to the center, the radius $1/\log n$ edge captures 2/3 of the weight of the edge.

Notice that this definition ensures that when growing a ball by δ that we capture a weight of $\delta\#edges(S, \overline{S})$ where S is the set of nodes inside the ball.

Now, for each of the $2logn$ radii the weight of the ball either doubles over the previous one or doesn't. In fact, it is clear that at least half of them don't double, since otherwise we would have captured more than $2^{\log n} W/n > W$ weight.

Now consider, one of this non doubling radii, say $i+1$. The weight increase in this interval is at least the minimum cut size, $C_{\min}$, in this interval times the radius, $1/\log n$. Moreover, this weight increase is less than W_i, where W_i is the weight in the ball of radius r_i.

That is, we have

$$C_{\min}\frac{1}{4\log n} \leq W_i.$$

That is,

$$C_{\min} \leq W_i 4\log n.$$

Scaling by n^2 gives us the lemma that we wish to prove. □

If the cut that is produced in the lemma above is reasonably balanced in terms of nodes, then we are done. (That is the cut has sparsest cut ratio $O(W\log n)$ if the S and $\overline{S}$ have size $\Omega(n)$.

This is unfortunately not the case. Thus, we simply repeat the shortest path algorithm above until we produce a balanced cut. That is the "repeated shortest path algorithm" is to use the lemma above to find a cut, place the small side of the cut

In fact, as we do this we may have a subgraph which is no longer of high diameter.

Question 2. Show that if for any subset S where there are more than $n/2$ nodes that there exists a pair of nodes $i,j \in S$ where $d(i,j) > 1/(8n^2)$, that the repeated shortest path algorithm produces a cut of sparsity at most $O(W\log n)$.

Thus, we need to tweak things a bit.

5.2 A large small piece.

The following lemma details the case where the repeated shortest path algorithm may not apply.

Lemma 3

If there is a solution to the distance LP, with a subset S with $n/2$ or more nodes, where there is a node r such that

$$\sum_{r,j\in S} d(r,j) \leq 1/16n$$

,

then there is a cut of sparsity $O(W)$.

We prove this in the next lecture.