Massachusetts Institute of Technology
6.854J/18.415J: Advanced Algorithms
Ankur Moitra

Wednesday, March 9, 2016

Problem Set 5

Due: Wednesday, March 16, 2016 – 7 pm
Dropbox Outside Stata G5

Collaboration policy: collaboration is *strongly encouraged*. However, remember that

1. You must write up your own solutions, independently.
2. You must record the name of every collaborator.
3. You must actually participate in solving all the problems. This is difficult in very large groups, so you should keep your collaboration groups limited to 3 or 4 people in a given week.
4. Write each problem in a separate sheet and write down your name on top of every sheet.
5. **No bibles. This includes solutions posted to problems in previous years.**

1 Optimizing Search Results

The Re:Search engine is a web search engine designed to do a better job of returning results when users re-issue (possibly after a substantial time) a query they have issued in the past. Studies have shown that people remember roughly where in a result list a given result was found, and get annoyed and slowed down if the result is in a different place when they repeat the search. In tension with this is the fact that if users repeat, there may be new results that we want to include because they are better. How can we build a good "merged" result list that contains new results without disrupting what users remember about old ones?

The Re:Search engine starts from the assumption that each old and each new result had some (predicted) *value* v_i to the user which would be accrued if the result was in the merged list at position i. It also used experimental data to specify a *change penalty* p_{ij} for moving a result that was at position i in the old list to a different position j in the merged list. The system also enforces that any old item the user actually clicked on must appear somewhere in the merged result set, and also that at least k new results must appear to provide an up to date sense of the results (you can assume there is enough room for this, i.e. that the number of slots in the merged list exceeds k plus the number of old clicked links.)

Given old and new result lists with all the values described above as input, give an algorithm for building a best merged result list of n items. In particular, formulate the problem as a min-cost max-flow problem.

2 Min-Cost Flow Variants

Give efficient algorithms for the following variants of min-cost flow.

(a) Find the minimum-cost flow subject to the constraint that the flow value is at least 90% of the maximum flow value.

(b) Find the flow that minimizes the cost of the flow plus K times the difference between the flow value and the maximum flow value, for a given weight K.

You can rely black-box on the the min-cost maximum flow algorithms discussed in class.

3 Alternative Cycle Canceling Algorithm

In class we analyzed a version of Klein's cycle canceling algorithm for min-cost maximum flow due to Goldberg and Tarjan. That algorithm always chose to cancel the cycle with *minimum mean cost*, which can be found efficiently. This served as an alternative to finding the cycle of *minimum total cost*, which is an NP-hard problem.

In this problem we'll look at a third alternative for choosing cycles. Specifically, assume as usual that we already have a maximum flow f for our graph G and let H be the residual graph corresponding to that flow. Let $u(e)$ be the residual capacity of an edge e in H and let $c(e)$ be the cost of the edge. Cancel the negative cost cycle C in H that minimizes:

$$\frac{\sum_{e \in C} c(e)}{\sum_{e \in C} 1/u(e)}. \tag{1}$$

Recall that we're minimizing a negative number here. This rule favors cycles with large negative costs on edges that also have a lot of capacity available. (1) has large denominator and thus small absolute value if C has a bottleneck edge with very little capacity remaining.

For this problem you should assume that G has integer capacities and edge costs. The maximum capacity is U and the maximum cost is Z.

(a) First we claim that a large negative value of (1) immediately implies that we make significant progress when canceling cycle C. Specifically, let $\Delta_H(C) = \min_{e \in C} u(e)$ be the the bottleneck capacity in C. After canceling C, we change the cost of our flow f by the negative value $\Delta_H(C) \sum_{e \in C} c(e)$. Show that:

$$\Delta_H(C) \sum_{e \in C} c(e) \leq \frac{\sum_{e \in C} c(e)}{\sum_{e \in C} 1/u(e)} \tag{2}$$

(b) Let m be the number of edges in G. If f^* is a min-cost maximum flow, show that, for our chosen cycle C,

$$\frac{\sum_{e \in C} c(e)}{\sum_{e \in C} 1/u(e)} \leq \frac{Cost(f^*) - Cost(f)}{m}, \tag{3}$$

and conclude from part (a) that:

$$\Delta_H(C) \sum_{e \in C} c(e) \leq \frac{Cost(f^*) - Cost(f)}{m}. \tag{4}$$

In other words, we reduce our flow by more than a $\frac{1}{m}$ fraction of our current distance from optimal.

Hint: f^* can be obtained by adding some fixed set of negative cost cycles to f . Argue that one of *these* cycles satisfies (3). The optimal cycle can only do better.

(c) Argue that after $O(m \log(UZm))$ iterations of cycle canceling, we will obtain a min-cost maximum flow.

(d) **(Optional)** Describe an efficient algorithm for finding a cycle C which minimizes (1).

4 Separating Polyhedra

Suppose you are given two polyhedra $P = \{\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\}$ and $Q = \{\mathbf{x} \mid \mathbf{Dx} \leq \mathbf{e}\}$.

(a) Using Farkas' lemma, prove that if polyhedra P and Q have empty intersection (i.e. no point is in both), then there are $\mathbf{y}, \mathbf{z} \geq \mathbf{0}$ such that $\mathbf{y}^T\mathbf{A} + \mathbf{z}^T\mathbf{D} = \mathbf{0}$ but $\mathbf{y}^T\mathbf{b} + \mathbf{z}^T\mathbf{e} < 0$.

(b) Conclude that if polyhedra P and Q have empty intersection (i.e. no point is in both), then there is a separating hyperplane for P and Q (i.e., a vector $\mathbf{c}$ such that $\mathbf{c}^T\mathbf{x} < \mathbf{c}^T\mathbf{w}$ for all $\mathbf{x} \in P$ and $\mathbf{w} \in Q$).

5 Linear Program for Minimum Mean Cost Cycle

Recall that in the Goldberg-Tarjan *minimum mean cost* cycle canceling algorithm we sought a cycle C in the residual graph H that minimizes:

$$\frac{\sum_{(u,v) \in C} c(u,v)}{|C|} \tag{5}$$

Consider a linear program MinMeanCost with m variables $x(u,v)$ for each edge in H:

$$\begin{aligned} \min \sum_{u,v} c(u,v)x(u,v) \text{ such that:} & \\ \text{for all nodes } v, \sum_u x(u,v) - \sum_u x(v,u) &= 0 \\ \sum_{u,v} x(u,v) &= 1 \\ x(u,v) &\geq 0 \end{aligned}$$

(a) Show that the minimum value of MinMeanCost is equal to the mean cost of the minimum mean cost cycle. **Hint:** Show that the values of $x(u, v)$ form a *circulation* in H. A circulation is like a flow, but flow is conserved at every node – i.e. there are no special source/sink nodes s and t. As in flow decomposition, we can decompose circulations, but the decomposition will only contain cycles.

(b) Write down a dual linear program for MinMeanCost. Show that the optimum of the this dual program is equivalent to ϵ from our original analysis of Goldberg-Tarjan.

(c) Conclude that ϵ is equal to the mean cost of the minimum mean cost cycle in H. We proved this statement in class without going through linear programming.