Problem Set 5

Due: Wednesday, October 10, 2012.

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. **No bibles. This includes solutions posted to problems in previous years.**

NONCOLLABORATIVE Problem 1. Critical edges for maximum flow.

(a) An edge is *upward critical* if replacing its capacity c by any $c' > c$ increases the maximum flow value. Does every network have an upward-critical edge? Give an algorithm to identify all upward-critical edges in a network. Its running time should be substantially better than that of solving m maximum-flow problems.

(b) An edge is *downward critical* if replacing its capacity c by any $c' < c$ decreases the maximum flow value. Is the set of upward-critical edges the same as the set of downward-critical edges? Describe an algorithm for identifying all downward-critical edges, and analyze your algorithm's worst-case complexity. Its running time should be substantially better than that of solving m maximum-flows.

Problem 2. The minimum flow problem is a close relative of the max flow problem with nonnegative lower bounds on edge flows (if an edge (i, j) has a lower bound $l_{ij} \geq 0$, then f_{ij} must satisfy $l_{ij} \leq f_{ij} \leq u_{ij}$). In the minimum flow problem we wish to send a minimum amount of flow from the source s to the destination t while satisfying given lower and upper bounds on edge flows f_{ij}.

(a) Show how to solve the minimum flow problem by using two applications of any maximum flow algorithm that applies to problems with zero lower bounds on edge flows (e.g., the algorithms described in the lecture). Your algorithm should detect if there is a feasible flow and, if there is one, return a minimum flow. **Hint**: first construct any feasible flow and then convert it into a minimum flow.

(b) Prove the following min-flow-max-cut theorem. Define the lower capacity bound of an s-t cut $(S, T = V \setminus S)$ as $\sum_{(i,j)\in S\times T} l_{ij} - \sum_{(i,j)\in T\times S} u_{ij}$. Show that the minimum value of all feasible flows from node s to node t is equal to the maximum lower capacity bound over all s-t cuts.

(c) A group of students wants to minimize their lecture attendance by sending only one of the group to each of n lectures. Lecture i begins at time a_i and ends at time b_i. It requires r_{ij} time to commute from lecture i to lecture j. Do not assume these times are integers. Develop a min-flow-based algorithm for identifying the minimum number of students needed to cover all the lectures.

Problem 3. It is visit day for the new graduate admits, and each wants a one-on-one visit with a certain set of faculty members. The day will be divided into time slots during which each student can meet at most one faculty member, and vice versa.

(a) Suppose that the numbers of faculty and students are equal, each student wants to meet exactly d faculty, and each faculty member is on the request list of d students. Conclude that one can schedule a single slot in which every student is meeting someone. **Hint:** show the minimum cut of the desired-meetings graph is large.

(b) Conclude that it is possible to schedule all the meetings to take place in d time slots.

(c) Consider an arbitrary set of desired meetings. Obviously one needs at least as many slots as there are faculty to meet a given student, and students to meet a given faculty. Prove that one can arrange all meetings with no more slots than this number s.

(d) Show that the schedule can be computed in $O(s^2 n^{3/2})$ time, where s is defined in (c) and n is the total number of students and faculty members.

Note: this algorithm is actually used to schedule grad visit day, but some extra complexities need to be factored in (specifically, the fact that some faculty are unavailable at certain times), making the problem NP-hard.

Problem 4. At a certain point in the season, each team i in the American League has won a certain number w_i of games, and there remain q_{ij} games to be played between teams i and j. Assuming no ties or cancelled games, develop an efficient, flow-based algorithm for deciding if the Red Sox can still win the league pennant, i.e. whether a particular team can still win more games than any other team after all games have been played.

Problem 5. 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 a polynomial-time algorithm for building a best merged result list of n items. In particular, formulate the problem as a min-cost max-flow problem.

OPTIONAL Problem 6. In class we proved a running time bound of $O(m^{3/2})$ for finding a maximum flow in a unit-capacity graph with m edges using blocking flows. Here we will prove some related bounds.

(a) Prove that in the "bipartite matching graph" discussed in class, blocking flows find a max-flow in $O(m\sqrt{n})$ time.

(b) Consider a unit-capacity simple graph (no parallel edges), and let d be the distance between the source and sink. Prove that the value of the maximum flow is at most n^2/d^2. Conclude that $O(n^{2/3})$ blocking flows suffice to find a maximum flow. Note that we end up with two incomparable bounds for blocking flow: $O(m^{3/2})$ (tighter for sparse graphs) and $O(mn^{2/3})$ (tighter for dense graphs). **Hint:** argue that there are two adjacent layers in the admissible graph with few vertices in each. How much flow can cross between the two layers?)

OPTIONAL (c) Using the ideas from the previous part, prove that any simple unit-capacity graph with a flow of value f has one that uses only $n\sqrt{f}$ edges. **Hint:** if you find the flow using shortest augmenting paths, what can you say about the length of each augmenting path that you use?

Problem 7. How long did you spend on this problem set? Please answer this question using the Google form at http://bit.ly/nZ9Q5m . This problem will be graded.