CO 602: Fundamentals of Optimization
Fall 2010
Problem Set 6
S. Vavasis

Handed out: 2010-Nov-3.

Due: 2010-Nov-10, in lecture.

1. Consider the Ford-Fulkerson algorithm applied to a network $G = (N, A, \mathbf{u})$. Here, $\mathbf{u}$ stands for the vector of capacities, one entry per arc in A. Let $\mathbf{f}^*$ be the optimal flow found by the FF algorithm and let $G' = (N, A', \mathbf{u}')$ be the residual network at the time of termination of the algorithm.

   (a) Show that if G' contains a directed cycle of length more than 2, then the optimizer $\mathbf{f}^*$ of the original problem is not unique. Provide an example of a network with a nonunique optimizer.

   (b) Conversely, show that if the original problem has more than one optimizer, then the residual network will contain a directed cycle of length more than 2. [Hint: Consider the subgraph of G', say G'', containing those arcs in which the two optimizers have different flows.]

2. The *courses-with-prerequisites* problem takes as input a directed graph $G = (N, A)$ that has no directed cycles (called a "directed acyclic graph", or "dag" for short). Each node v of the dag is labeled with a number $b(v)$ (positive, negative or zero). The problem is to select a subset of nodes S such that $\sum_{v \in S} b(v)$ is maximized. The constraint is that if $v \in S$ and $(u, v) \in A$, then $u \in S$ also. (Applying this rule recursively therefore implies that all ancestors of v must be included in S.)

   One can think of the nodes of a graph as courses that a student may take. The label $b(v)$ is the benefit for taking that course, but the course may have prerequisites that also must be taken.

   Picard showed that the Ford-Fulkerson algorithm can solve solve the courses-with-prerequisites problem. In particular, given an instance of courses-with-prerequisites, create a new digraph $\tilde{G}$ as follows. Graph $\tilde{G}$ has all the nodes of G plus a single source s and a single sink t. Insert an arc (s, v) into $\tilde{G}$ for every node v of the original dag with positive cost. Insert an arc (v, t) for every node v of the original dag with negative cost. Finally, insert all the arcs of G into $\tilde{G}$ with reversed orientation. Assign capacities as follows. The capacity of (s, v) is $b(v)$. The capacity of (v, t) is $-b(v)$. (Thus, all capacities are nonnegative.) Finally, the capacity of (v, u), in the case (u, v) is an original arc, is infinity.

   Show that the min-cut solution to $\tilde{G}$ corresponds to the optimal solution to the courses-with-prerequisites problem for G.

   [Hint: If a cut of $\tilde{G}$ includes only finite-capacity arcs, then relate its capacity to $B - b(S)$, where $b(S) = \sum_{v \in S} b(v)$ $B = \sum_{b(v)>0} b(v)$.]

3. Let $\mathbf{x}$ be an optimal solution to the LP relaxation of the assignment problem, and suppose that some entries of $\mathbf{x}$ are fractional. Use the term *fractional* to mean a noninteger numeric value. Describe an algorithm to transform $\mathbf{x}$ to an integer (0-1) solution. Establish the correctness of your algorithm (i.e., finite termination; feasibility; optimality).

   [Hint: Form a graph whose nodes are workers and tasks such that edge (i, j) is inserted into the graph if x_{ij} is fractional. Argue that this graph cannot have any degree-1 vertices. Argue, therefore, that if it has any edges at all, then it must have a cycle. Argue that the cycle must have an even number of edges. Update the x_{ij}'s around the cycle using alternating signs to maintain feasibility and optimality. Then repeat. For the proof that your algorithm produces an optimizer, you may use as a hypothesis the optimality of the original fractional solution.]

4. *Breadth-first search* (BFS) of a directed graph G starting from a node s is defined as follows. Let the distance between a node s and a node t be the number of arcs on the shortest path from s to t, else ∞ if there is no directed (s, t) path. BFS starts from a node s. Next it visits nodes of distance 1 from s, then distance 2, and so on. This is accomplished using a data structure called a FIFO ("first in, first out") queue Q. This data structure supports three operations: 'insert', 'remove' and 'size'. The 'insert' operation inserts a data item (say an integer) into the queue. The 'remove' operation removes the data item that is the oldest still remaining in the queue, and it returns the removed data item. Finally, 'size' returns the number of elements in the queue.

   BFS initializes $Q = \{s\}$ and labels s with 0. Then it repeatedly removes an item from the queue, say v whose label is d, finds all its out-neighbors. Of the the out-neighbors not already labeled, it labels them $d + 1$ and inserts them in the queue. This iteration is repeated until the queue is empty.

   Implement BFS in Matlab. Two helper routines, `makeoutarcs.m` and `queue.m` are provided; the former changes an arc-list representation of G into an out-adjacency representation, and the latter implements the FIFO queue data structure. Note that `queue` will not work under Matlab version 6 or earlier. Note that the `inf` function returns infinities. Your function should have the following header:

```
function labels = bfs(nnode,arclist,s)
% labels = bfs(nnode, arclist, s)
% Carries out a breadth-first search on a digraph.  Input arguments are as
% follows: nnode = number of nodes in the digraph.  arclist is an m-by-2
% array; each row is one directed arc of the digraph.  The two entries of
% the row are integers in the range 1:nnode and indicate the two endpoints
% of the arc.  Finally, s lies in 1:nnode and is the starting point of the
% bfs.
% The output labels is a column vector of length nnode.  The entries are
% the distances (integers) from s.  A node that is not reachable from s is
% labeled with Inf.
```

Try your BFS on the graph in the course website. Try it with $s = 1$ and $s = 2$. In the case of $s = 1$, the correct answer for labels is given on the course website so you can check your code.

Hand in a listing of your code and sample runs with $s = 1$ and $s = 2$.