COMP 360 Algorithm Design Techniques

Assignment 4: Solutions

(1) *Maximum Flows with Vertex Capacities.*

Create nodes and arcs as follows:

nodes: Copy the source s and the sink t to the new problem instance. For each intermediate (non-source, non-sink) vertex v in the original problem instance, create two intermediate vertices v_{in} and v_{out} in the new problem instance. Also, create a directed arc (v_{in}, v_{out}) of capacity $c(v_{in}, v_{out}) = u_v$, the capacity on the vertex v in the original problem instance.

arcs: If the original instance had an arc (s, t), copy this arc and its capacity to the new instance.

For each arc (s, a) in the original instance that connected the source to some intermediate vertex a, create an arc of the form (s, a_{in}) of the same capacity in the new instance. For each arc (a, t) in the original instance that connected some intermediate vertex a to the sink, create an arc (a_{out}, t) of the same capacity in the new instance. Finally, for each arc (a, b) between intermediate vertices in the original instance, create an arc (a_{out}, b_{in}) of the same capacity in the new instance.

Note that the time to carry out this transformation is linear in the length $|x|$ of the encoding x of the original problem instance.

To solve the original problem instance, find a maximum flow in the new problem instance and copy the flow values of all the arcs except ones having the form (v_{in}, v_{out}) back to the corresponding arcs in the original problem instance. Since (v_{in}, v_{out}) is the only arc leaving v_{in}, and also, the only arc entering v_{out}, by the flow balance property in the new instance, the flow into v_{in} must be equal to the flow from v_{in} to v_{out} through the arc (v_{in}, v_{out}), which must be equal to the flow leaving v_{out}. Therefore, in the original instance, we obtain a solution such that the flow into v equals the flow out of v, for each intermediate node v.

Since the flow through (v_{in}, v_{out}) is at most u_v, the flow through v in the original instance is at most u_v as desired.

(2) *The Student Societies Problem.*

Set up a maximum flow problem as follows. Create one vertex for each organization, one vertex for each table, and also, source and sink vertices s and t. For each organization i, add an arc of capacity s_i from the source s to the vertex i representing organization i; the capacity s_i will ensure that at most s_i members of organization i get seated. Add an arc of capacity c_j from the vertex j representing table j to the sink t; the capacity c_j will ensure that at most c_j people sit at table j. Finally, add an arc of capacity 1 from vertex i representing organization i to vertex j representing table j; the capacity 1 will ensure that at most one member of organization i is placed at table j.

Now find a maximum flow from s to t in the graph. Each unit of flow from s to t must go through exactly one organization vertex u and one table vertex v; this corresponds to placing one member of organization u at table v. The capacity constraints given above ensure that we have a good seating arrangement if and only if we have a flow of value $\sum_i s_i$. (Note that the flow value cannot be greater than this quantity.)

(3) *Updating the Maximum Flow.*

(a) If arc (i,j) is by chance part of *every* cut having minimum capacity, then increasing the capacity on (i,j) by one actually increases the value of the minimum cut capacity by one. This means that the maximum flow value is actually one greater than we thought. Of course, we don't know in advance whether arc (i,j) has this property, but we do know that the flow can go up by at most one unit. Hence we simply increase the capacity of (i,j) by one, and run one more iteration of the Ford-Fulkerson algorithm. This will either "get stuck" without reaching the sink, meaning that the flow value stays the same, or will "break through" to the sink, increasing the flow value by one.

(b) If the arc (i,k) is by chance part of *any* cut having minimum capacity, then decreasing the capacity on (i,j) decreases the minimum cut capacity by one, and hence the true maximum flow value is one less than what we thought. Of course we don't know if this is the case in advance, and also, we would like to find the set X of actual flows in the arcs after correcting the capacity in (i,j) downward by one. Instead of resolving the problem from the beginning, we can do the following.

If the flow in (i,j) is less than the old capacity on (i,j), we do nothing: if (i,j) had been part of some minimum capacity cut, then the arc would have been full. Therefore, (i,j) is not part of a minimum capacity cut, so there is no change to the value of the maximum flow when we reduce the capacity of (i,j) by one, and also, there is still room for the old flow in (i,j).

Assume now that arc (i,j) is full. In particular, it contains at least one unit of flow. Remove arc (i,j). Then starting at j, do BFS on arcs with positive flows to find a path to t. Similarly, starting at i and reversing the directions of the arcs, find a path from i to s. Use these two paths together with (i,j) to obtain a directed path from s to t through (i,j), using only arcs with positive flows. Reduce each of these flows by one. The resulting flow will be legal and maximum in the new network.

Note that it is not possible that some arc in the path from s to i also appears in the path from j to t. (That would be a problem since we would have to reduce its flow by two, which might not be legal.) This would imply the existance of a directed cycle, through intermediate vertices only, that uses (i,j). We could reduce the flow in this cycle by one without affecting the maximum flow, and still satisfying flow balance and the new capacity on (i,j); this contradicts that we know the maximum flow value must decrease by one in the current case.

(4) *Linear Programming.*

(a) max $-5x_1 + 7x_2 - 2x_3 + x_4$ s.t.
$8x_1 + x_2 - x_3 - 3x_4 \leq 6$
$-2x_2 - x_4 \leq -10$
$x_1 + 3x_3 + x_4 \leq -3$
$x_2 - 5x_3 - 2x_4 \leq 0$
$-x_2 + 5x_3 + 2x_4 \leq 0$
$x_1, x_2, x_3, x_4 \geq 0$

(b) Formulating the dual problem as the problem of finding the "best multipliers" for the original (primal) problem gives the following:
min $6y_1 - 10y_2 - 3y_3$ s.t.
$8y_1 + y_3 \geq -5$
$y_1 - 2y_2 + y_4 - y_5 \geq 7$
$-y_1 + 3y_3 - 5y_4 + 5y_5 \geq -2$
$-3y_1 - y_2 + y_3 - 2y_4 + 2y_5 \geq 1$
$y_1, y_2, y_3, y_4, y_5 \geq 0$

Expressing this dual problem as a linear programming problem in the standard form for a primal problem gives:
max $-6y_1 + 10y_2 + 3y_3$ s.t.
$-8y_1 - y_3 \leq 5$
$-y_1 + 2y_2 - y_4 + y_5 \leq -7$
$y_1 - 3y_3 + 5y_4 - 5y_5 \leq 2$
$3y_1 + y_2 - y_3 + 2y_4 - 2y_5 \leq -1$

$y_1, y_2, y_3, y_4, y_5 \geq 0$

(5) *The Shortest Paths Problem.*

For $1 \leq i \leq n$, let x_i denote the length of the shortest path from source $s = 1$ to vertex i.

Then for each directed edge (i, j), we know that

$x_j \leq x_i + d(i, j)$, where $d(i, j)$ is the weight or length of the edge (i, j).

This inequality can be rearranged to the form

$x_j - x_i \leq d(i, j)$.

Also, we have that $x_1, \ldots, x_n \geq 0$; in fact, we want $x_1 = 0$, which can be expressed by adding the constraint $x_1 \leq 0$ to the above constraints.

Suppose, for an objective function, that we *maximize* $x_1 + \cdots + x_n$ subject to all the above inequalities.

Now we know that the true shortest distances satisfy the above constraints. Furthermore, it can be shown that there is no other solution $x_1^{'}, \ldots, x_n^{'}$ that gives a larger value of the objective function.

Note that it does not work to minimize this objective function, as the all 0 solution would be obtained.

(6) The Kayak Problem

For each person j, the net distance that person swims toward the opposite side plus the net distance that person paddles toward the opposite side must be equal to the width of the river, which is 10.

$s_j x_j - s_j x_j^{'} + p_j y_j - p_j y_j^{'} = 10$

From the kayak's point of view, the net distance the kayak moves toward the opposite side should be between 0 and 10, inclusive. Since a kayak can only hold one person,

$0 \leq p_1 y_1 - p_1 y_1^{'} + \cdots + p_n y_n - p_n y_n^{'} \leq 10.$

Of course, $x_1, y_1, x_1^{'}, y_1^{'}, \ldots, x_n, y_n, x_n^{'}, y_n^{'} \geq 0.$