

CS 222A Fall 2007 Homework 4, Due November 7

1. The book shows how to use network flow to determine if a team  $i$  is eliminated in baseball, or alternatively if it can at least tie for the lead at the end of the season. How can we efficiently solve the problem of determining if  $i$  can *strictly* lead at the end, i.e., having strictly more wins than any other team?

Answer: Reduce the capacities all on the edges into  $t$  by one. Previously, they were set to allow another team to get as many wins as team  $i$  would if team  $i$  wins all its remaining games. By reducing the capacities of the edges into  $t$ , we are now specifying that team  $i$  will have strictly more wins than any other team.

2. In Problem 2 from HW 3, a “legal assignment” of values to the empty cells, is one such that with the assigned values included, every row adds up to its given row total, and every column adds up to its given column total. In HW 3 you solved the problem of finding one legal assignment if there is one, and otherwise determining that there is no legal assignment. For technical simplicity, we assume now that each of the cell values is an integer, and therefore the row and column totals are also integers.

We would now like to know for each empty cell  $(u, v)$ , what is the *maximum* possible value, call it  $v^*(u, v)$ , such that there is some legal assignment of values to the empty cells which assigns value  $v^*(u, v)$  to cell  $(u, v)$ .

Show how to use network flow to determine  $v^*(i, j)$ . A single network flow computation suffices, after determining some legal assignment. Hence if we want to compute  $v^*(i, j)$  for each empty cell  $(i, j)$ , it suffices to compute one flow to determine a first legal assignment, followed by one flow per empty cell. Hint: Use the answer from problem 2 in HW 3, and the kind of thinking from, or the actual results of, Problem 4 from HW 3.

Answer: See the notes already posted on the class website concerning this problem.

3. Solve problem 13 on page 420. There are two ways to do this problem: 1) the long and hard way where you essentially recapitulate all the material we did on network flows, changing edge capacities to node capacities, or 2) the simple and easy way, based on converting the node capacity problem to the edge capacity problem and applying what we already know there. Try to answer this problem using the second way.

Answer: Let  $G$  denote the original graph. Convert any node  $v$  of  $G$  into two nodes  $v_1$  and  $v_2$  and put a directed edge from  $v_1$  to  $v_2$  with capacity equal

to  $w(v)$ . Next, for every edge that was directed into  $v$ , now direct it into  $v_1$ , and for every edge that was directed out of  $v$ , now direct it out of  $v_2$ . The capacities of all original edges are infinite. Let  $G'$  denote the new graph. Then a maximum flow s-t flow in  $G'$  specifies a maximum s-t flow in  $G$ . Further, since all original edges in  $G$  have infinite capacity in  $G'$ , the minimum s-t cut in  $G'$  consists only of new edges. Therefore the minimum cut in  $G'$  can be considered as a set of nodes in  $G$ , with total capacity exactly the same as the minimum edge capacity in  $G'$ . Therefore, by the Max-Flow-Min-Cut theorem (stated for edge capacities) it follows that the maximum s-t flow equals the minimum total weight of a set of nodes whose removal disrupts all s-t paths in  $G$ .

4. When the capacity of every edge that is in a directed graph is 1, then the Ford-Fulkerson algorithm runs in  $O(nm)$  time. What about the Preflow-Push algorithm in this case? (Go through the time analysis of the Preflow-Push algorithm carefully to find the answer.) Does your analysis need the modification of the Preflow-Push algorithm that picks as the node to push from, the node  $v$  with excess that has the highest label  $h(v)$ ?

Now answer the question when every edge has integral capacity at most  $k$ , for some fixed integer  $k$  (i.e., independent of  $n$  or  $m$ ).

Answer: When all capacities are 1, then every push is a saturating push, since every edge flow and every residual capacity is 1. Now in the analysis of the Preflow-Push algorithm, the number of operations is bounded by the number of node relabels, which is  $O(n^2)$ , the number of saturating pushes, which is  $O(nm)$ , and the number of non-saturating pushes, which is zero in this case. Hence when all capacities are 1, the Preflow-Push algorithm runs in  $O(nm)$  time. This bound does not depend on which node (with excess) is chosen to push from.

When all capacities are INTEGRAL and at most  $k$  (I didn't state the integral assumption, but I think students mostly assumed it), then the time bound is  $O(nm)$  also.

When the capacities are all integral and the capacity of an edge  $(u,v)$  is bounded by  $k$ , then there can be at most  $k$  pushes from  $u$  to  $v$  (uninterrupted by a back push from  $v$  to  $u$ ) before saturation. So look at the history of forward pushes on  $(u,v)$  and backward pushes from  $v$  to  $u$ . The history consists of alternating runs of non-saturating forward pushes, and non-saturating backward pushes. But forward pushes require that  $h(u) > h(v)$ , and backward pushes require that  $h(v) > h(u)$ , and since  $h(u)$  and  $h(v) < 2n$ , there

can be at most  $2n$  alternating runs of forward and backward pushes on  $(u,v)$  and  $(v,u)$ . In each run, the number of non-saturating pushes is at most  $k$ , since each push is an integral amount and a run of  $k$  forward pushes would reach the capacity of  $(u,v)$  and a run of  $k$  backward pushes would create a flow of 0 on edge  $(u,v)$ . Therefore, on edges  $(u,v)$  and its backedge  $(v,u)$ , there can be at most  $2nk$  non-saturating pushes over the life of the algorithm. Hence, there can be at most  $O(nmk)$  non-saturating pushes on any edge over the life of the algorithm. Now if  $k$  is a fixed number, independent of  $n$  and  $m$ , then  $O(nmk) = O(nm)$ .