Massachusetts Institute of Technology
6.854/18.415: Advanced Algorithms
David Karger

Handout 9
October 14, 1999

Problem Set 5

Due: October 26, 1999.

Problem 1. Critical arcs for maximum flow

(a) An arc is *upward critical* if increasing the capacity of this arc increases the maximum flow value. Does every network have an upward critical arc? Describe an algorithm for identifying all upward critical arcs in a network. The worst-case complexity of your algorithm should be substantially better than that of solving m maximum flow problems.

(b) An arc is *downward critical* if decreasing the capacity of this arc decreases the maximum flow value. Is the set of upward critical arcs the same as the set of downward critical arcs? If not, describe an algorithm for identifying all downward critical arcs; analyze your algorithm's worst-case complexity.

Problem 2. In this problem we develop a scaling algorithm for shortest paths that achieves running time $O(m \log C)$ with no data structure more complicated than an array. Recall that when we discussed shortest paths with positive integer edge lengths, we showed how Dial's algorithm uses a single array of buckets to find shortest paths in $O(m + nC)$ time for maximum edge length C.

Consider a solution to a shortest path problem with edge lengths l_{vw} from v to w in which the (shortest path) distance of vertex w from the source s is d_v. Define the *reduced edge lengths* l^d_{vw} by the rule $l^d_{vw} = l_{vw} + d_v - d_w$.

(a) Argue that in fact the running time of Dial's algorithm is $O(m + D)$, where D is the maximum distance, and that the algorithm still works if some edges have length 0.

(b) Show that the reduced edge lengths defined above are all nonnegative integers.

(c) Show that the shortest paths under the reduced length function are the same as those under the original length function. What is the length of shortest paths under l^d?

(d) Devise a scaling algorithm for shortest paths. Use the reduced edge lengths and Dial's bucketing algorithm to keep the task of shifting in a new bit simple.

(e) Is base-2 scaling the best possible, or can you achieve a (slightly) better running time by scaling with a different base?

Problem 3. Extending unit-capacity flow bounds to other graphs.

(a) Consider a graph in which all edges are unit capacity, *except* for the edges incident on p vertices. Extend the unit-capacity flow bounds to give good bounds in terms of m, n, and p on the number of blocking flows needed to find a maximum flow.

(b) Derive good bounds (using dynamic tres) on the running time needed to find a maximum flow in graphs of the type just described.

(c) Show that if the high-capacity edges are incident only on s and t, then the good running times can be achieved with only minor changes to the implementation and analysis of the unit-capacity blocking flow algorithm.

(d) The B-matching problem is a generalization of bipartite matching in which vertex v needs to be matched to exactly b_v neighbors. Show that B-matching can be solved in $O(m\sqrt{n})$ time using your algorithm of the previous part.

Problem 4. Suppose that at some time during the execution of a push-relabel algorithm there are no nodes with distance label l. Show that no excess at a node with label greater than l can ever reach the sink. What implementation heuristic does this suggest?

Problem 5. In this question we consider the problem of converting a preflow into a flow with the same value. (Define the value of a preflow to be the total flow entering the sink.)

(a) Give an algorithm to solve this problem on a unit-capacity network in $O(m)$ time.

(b) Give a simple algorithm to solve this problem on a general capacity graph in $O(nm)$ time.

(c) By using fancy data structures, give an algorithm to solve this problem in $O(m \log n)$ time

(Hint: consider a preflow decomposition.)

Problem 6. Suppose we run a push-relabel algorithm on a unit-capacity graph, and never do any pushes from or relabels of a node with label greater than $\sqrt{m}$.

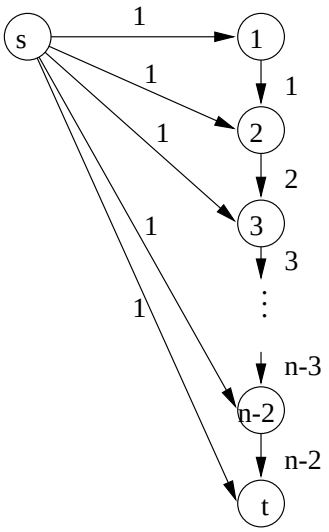
(a) Argue that this will terminate in $O(m\sqrt{m})$ time.

(b) How would the time bound for part (a) change if all arcs had integer capacity at most two, instead of one?

(c) Describe how we can take the result of this computation and turn it into a maxflow in $O(m\sqrt{m})$ time. (Your solution should be reasonable in the sense that it makes real use of part (a). Throwing away part (a) and running a unit capacity maxflow algorithm is not to be considered an answer to this question.)

(d) Show that push-relabel does not automatically achieve $O(m\sqrt{m})$ performance on a unit capacity graph. (That is, pick an order for pushes and relabels and give a graph on which this ordering will use more than $O(m\sqrt{m})$ time.)

Problem 7. Consider the behavior of the highest label push-relabel algorithm (ie, push-relabel where the highest labeled node is selected to be discharged) on the following graph:



(a) If, as described in class, we start by saturating all arcs out of the source, give the source distance label n, and give all other nodes distance label 0, how many pushes might we end up doing on this graph if we are unlucky about how we break ties?

(b) If instead we initialize the distance labels with the starting distance from the sink, how many pushes might we end up doing?

(c) What does this suggest about being lazy about distance labels versus having correct ones? What implementation heuristic does it suggest?