CIS 502: homework 5

Yezheng Li

October 22, 2015

Contents

1 Network flow

Question (HW5, 7-10) *Suppose you are given a directed graph $G = (V, E)$, with a positive integer capacity c_e on each edge e, a designated source $s \in V$, and a designated sink $t \in V$. You are also given a maximum s-t flow in G, defined by a flow value f_e on each edge e. The flow $\{f_e\}$ is* acyclic*: there is no cycle in G on which all edges carry positive flow.*

Now, suppose we pick a specific edge $\vec{e}^* \in E$ and reduce its capacity by 1 unit. Show how to find a maximum flow in the resulting capacitated graph in time $O(m)$, where m is the number of edges in G.

Answer (HW5, 7-10) *Denote capacity of directed edge $\vec{e} \in E_2$ by $c_{\vec{e}}$ in the last residual graph $G_2 = \langle V, E_2 \rangle$ with $\{c_{\vec{e}} \in \mathcal{N}_{\geq 0} : \vec{e} \in E_2\}$. Denote $\vec{e}^* = (u, w)$.*

1. *If $c_{(w,u)} = 0$, then the maximum flow does not change since $\vec{e}^*$ must have not been take advantage of to ship any flow to form the maximum flow;*

2. *Else,*

   (a) *use DFS to find a $u \to s$ path $P = uv_{k-1}v_{k-2} \dots v_1 s$ and a $t \to w$ path $P = tv_{l-1}v_{l-2} \dots v_{k+2}w$ in the residual graph with $c_{(v_{j+1},v_j)} > 0$ $(s = v_0, u = v_k, w = v_{k+1}, t = v_l)$. These run with $O(m + n)$ time.;*

(b) *Update capacities in the following way:*

i. $c'_{(v_k,v_{k+1})} \leftarrow c_{(v_k,v_{k+1})}, c'_{(v_{k+1},v_k)} \leftarrow c_{(v_{k+1},v_k)} - 1;$

ii. $c'_{(v_j,v_{j+1})} \leftarrow c_{(v_j,v_{j+1})} + 1,, c'_{(v_{j+1},v_j)} \leftarrow c_{(v_{j+1},v_j)} - 1,\ j = 0, 1, \ldots, l-1, j \neq k.$

Now we construct new $\{c_{\vec{e}} \in \mathcal{N}'_{\geq 0} : \vec{e} \in E_2\}$ *for* $G_2 = \langle V, E_2 \rangle$*. This runs with time* $O(m)$*.*

3. *Use DFS to search whether there is a positive* $s \to t$ *flow. This runs with* $O(m+n)$ *time. If exists, the value of maximum flow does not change (although the flow does change); if not, the value maximum flow is reduced by* 1 *unit.*

Total running time is at most $O(m+n)$*.*
Correctness: ♢

Remark *How about add* 1 *unit to capacity of the specific edge* $e^* \in E$*?* ♢

Answer (refers in [5]) *Add the new ow to the residual ow graph in* $O(m)$ *time. Perform a tree traversal from the source node to detect whether a path now exists to the sink. If so, augment along that path and increase the maximum ow by one. If exists, augment along that path and increase the maximum ow by one.*
The running time is $O(n)$*.*
Correctness: ♢

Question (HW5, 7-12) *Consider the following problem. You are given a flow network with unit-capacity edges: it consists of a directed graph* $G = (V, E)$*, a source* $s \in V$*, and a sink* $t \in V$*; and* $c_e = 1$ *for every* $e \in E$*. You are also given a parameter* k*.*

The goal is delete k *edges so as to reduce the maxmimum* s*-*t *flow in* G *by as much as possible. In other words, you should find a set of edges* $F \subseteq E$ *so that* $|F| = k$ *and the maximum* s*-*t *flow in* $G' = (V, E - F)$ *is as small as possible subject to this.*

Give a polynomial-time algorithm to solve this problem.

Answer (HW5, 7-12, refers to [3]) *If the minimum* s*-*t *cut has size* $\leq k$ *(since the flow network with unit-capacity edges), then we can reduce the flow to* 0*. Otherwise, let* $f > k$ *be the value of the maximum* s*-*t *flow. We identify a minimum* s*-*t *cut* (A, B)*, and delete* k *of the edges out of* A*. The resulting subgraph has a maximum flow value of at most* $f - k$*.*

> *[**Claim**] For any set of edges F of size k, the subgraph $G' = (V, E - F)$ has an s-t flow of value at least $f - k$.*

Indeed, consider any cut (A, B) of G'. There are at least f edges out of A in G, and at most k have been deleted, so there are at least $f - k$ edges out of A in G'. Thus, the minimum cut in G' has value at least $f - k$, and so there is a flow of at least this value. ♢

Remark *What if not "with unit-capacity edges"?* ♢

Question (HW5, 7-14) *We define the* escape problem *as follows. We are given a directed graph $G = (V, E)$ (picture a network of roads); a certain collection of nodes $X \subset V$ are designated as* populated nodes, *and a certain other collection $S \subset V$ are designated as* safe nodes. *(Assume that X and S are disjoint.) In case of an emergency, we want evacuation routes from the populated nodes to the safe nodes. A set of evacuation routes is defined as a set of paths in G so that (i) each node in X is the tail of one path, (ii) the last node on each path lies in S, and (iii) the paths do not share any edges. Such a set of paths gives a way for the occupants of the populated nodes to "escape" to S, without overly congesting any edge in G.*

(a) *Given G, X, and S, show how to decide in polynomial time whether such a set of evacuation routes exists.*

(b) *Suppose we have exactly the same problem as in (a), but we want to enforce an even stronger version of the "no congestion" condition (iii). Thus, we change (iii) to say "the paths do not share any* nodes."

With this new condition, show how to decide in polynomial time whether such a set of evacuation routes exists.

Also, provide an example with a given G, X, and S, in which the answer is "yes" to the question in (a) but "no" to the question in (b).

Answer (HW5, 7-14) *(a) Redistribute nodes in X left side and nodes in S right side as shown in figure 1. Add source s, sink t to V and add directed edges to E. Assign each edge with capacity 1 – this is important to avoid paths sharing edges. In all, denote*

$$E' \leftarrow \{(s, v) : v \in S\} \cup \{(u, t) : u \in X\} \cup E, V' \leftarrow V \cup \{s, t\},$$

and we actually construct a new graph $G' = \langle V', E' \rangle$ (with capacity on each edge) on which can apply algoritm to find maximum flow.

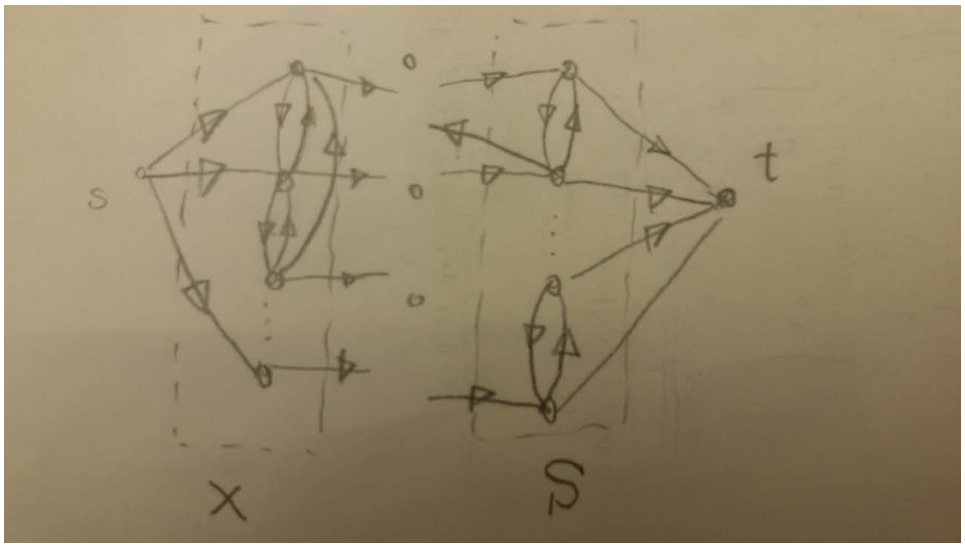


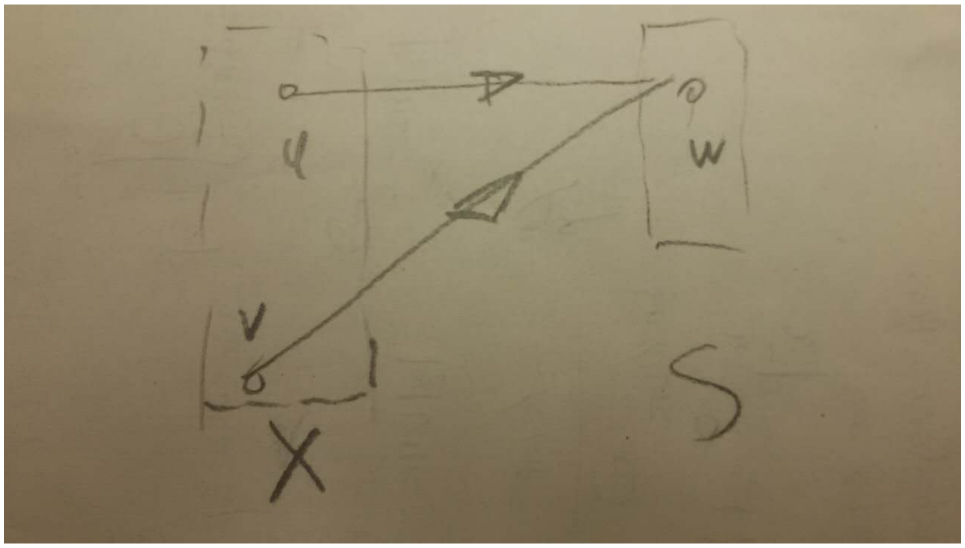
Figure 1: Construction of graph

If apply generic Preflow–Push Maximum–flow Algorithm in section 7.4 of [2], by statement (7.33) of [2] we can obtain it with running time $O\left((m+n)n^2\right) = O(mn^2 + n^3)$*; in the version where we always select the node at maximum height runs in* $O(n^3)$ *time.*

(b)

Question (HW5, 4-17) *You've been called in to help some network administrators diagnose the extent of a failure in their network. The network is designed to carry traffic from a designated source node* s *to a designated target node* t*, so we will model it as a directed graph* $G = (V, E)$*, in which the capacity of each edge is* 1*, and in which each node lies on at least one path from* s *to* t*.*

Now, when everything is running smoothly in the network, the maximum $s - t$ *flow in* G *has value* k*. However, the current situation – and the reason you're here – is that an attacker has destroyed some of the edges in the network, so that there is now no path from* s *to* t *using the remaining (surviving) edges. For reasons that we won't go into here, they believe the attacker has destroyed only* k *edges, the minimum number needed to separate* s *from* t *(i.e. the size of a minimum* $s - t$



◇

Figure 2: An example showing (a) ”yes” but (b) ”no”.

cut); and we'll assume they're correct in believing this.

The network administrators are running a monitoring tool on node s, which has the following behavior: if you issue the command ping(v), for a given node v, it will tell you whether there is currently a path from s to v. (So ping(t) reports that no path currently exists; on the other hand, ping(s) always reports a path from s to itself.) Since it's not practical to go out and inspect every edge of the network, they'd like to determine the extent of the failure using this monitoring tool, through judicious use of the ping command.

So here's the problem you face: give an algorithm that issues a sequence of ping commands to various nodes in the network, and then reports the full set of nodes that are not currently reachable from s. You could do this by pinging every node in the network, of course, but you'd like to do it using many fewer pings (given the assumption that only k edges have been deleted). In issuing this sequence, your algorithm is allowed to decide which node to ping next based on the outcome of earlier ping operations.

Give an algorithm that accomplishes this task using only $O(k \log n)$ *pings.*

Answer (HW5, 4-17)

References

[1] en.wikipedia.org.

[2] Jon Kleinberg, va Tardos, *Algorithm Design*, Addison-Wesley: 2005-3-26, 864 pages

[3] solcornell.tex.

[4] http://fulinyuntools.googlecode.com/svn-history/r459/trunk/compalgo/ps3/as3.tex.

[5] http://courses.csail.mit.edu/6.046/spring04/handouts/prac-final-sol.pdf.