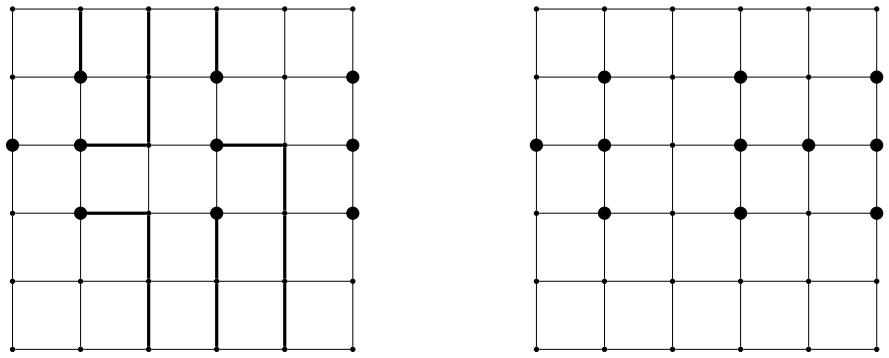
Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
M. Ahmadi, O. Saukh, A. R. Molla December 22, 2015

Algorithm Theory, Winter Term 2015/16
Problem Set 8 - Sample Solution

Exercise 1: Rectangular City (4 points)

Given a city map that looks like a rectangular grid. In this city in some points that are marked, people have built their houses. Since people in the city do not like other citizens, they want to build individual roads from their houses to the border of the city in such a way, that they do not cross in any point with the roads of others (so that they do not have to see other people driving). Roads can only go along the edges of the rectangular grid.
Design an algorithm which finds a solution (how to design such roads for the city) for a given grid with houses depicted on it, or indicates that it is not possible to construct such road network. Your algorithm should run in polynomial time.
Remark 1: If you use any flow network, describe it explicitly.
Remark 2: There is a solution in the left hand side example but no solution in the right hand side example.



Solution:

The problem is reduced to a max flow problem with vertex disjoint paths (note that any two vertex disjoint paths are also edge disjoint).
Let $A = \{a_1, \ldots, a_k\}$ be the houses and $B = \{b_1, \ldots, b_l\}$ the hubs that lead out of the city on the rim of the graph. Construct a flow network as follows:

1) Start with the grid as a graph, where each grid line is substituted with two directed edges.

2) **Every edge in the flow network will have capacity** 1.

3) Add a source s and a sink t.

4) Connect the source with every house node, that is, introduce an edge (s, a_i) for $i = 1, \ldots, k$.

5) Furthermore connect every hub with the sink t, that is, introduce an edge (b_i, t) for $i = 1, \ldots, l$.

Now a maximum flow of value k (if it exists) in the above network leads a solution to the given problem neglecting that street paths are not allowed to cross. We overcome this by ensuring that the flow through every node is at most 1 (vertex disjoint paths).
This can be achieved by substituting each node v with two nodes v_{in} and v_{out} and redirecting its edges. Every edge into v is substituted with a corresponding edge into v_{in} and every edge leaving v is substituted with an edge from v_{out}. Furthermore we add an edge (v_{in}, v_{out}) (e.g., see the following example).

The total capacity of all edges leaving s is k, so the value of a max flow will be smaller or equal to k. If there is a max flow with value k there will be vertex disjoint (and path disjoint) paths from the houses to the hubs.
The max flow problem can be solved in polynomial time.

Exercise 2: Network Flows (2+3 points)

(a) In this problem, we are given a flow network with unit-capacity edges: It consists of a directed graph $G = (V, E)$, a source node $s \in V$, a sink node $t \in V$, and capacity $c_e = 1$ for every $e \in E$. We are also given a positive integer parameter k.

The goal is to delete k edges so as to reduce the maximum 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 $G' = (V, E \backslash F)$ is as small as possible.

Give a polynomial-time algorithm to solve this problem.

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

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

Solution

a) Let g be the value of maximum flow of network G. First note that removing k edges can never result in a flow less than $g - k$, as edge has capacity 1. (1)

According to the max-flow min-cut theorem, there is an $s-t$-cut with g edges. If g is smaller or equal to k, then we simply remove all edges in that $s-t$-cut, disconnecting s and t, decreasing the maximum flow to 0. If $g > k$, by removing k edges from the min-cut, we create a cut with value $g - k$. (2)

From (1) and (2), we can claim that the min-cut in the new graph has the value of $g' = \max\{0, g-k\}$. Then based on max-flow min-cut theorem, the max-flow of the new graph is g'.

In both cases whether max-flow equals zero or $g - k$, the max-flow cannot be decreased any more which satisfies the algorithm requirement. The algorithm has polynomial running time, since we need polynomial time for computing the minimal $s - t$-cut, and linear in k time to remove the k edges.

Remark: In case the capacity of all edges are not equal to 1, removing the k edges from the min-cut does not guarantee to have the minimum possible max-flow.

b) Note first that the maximum flow in the new graph is either the same or it decreases at most by one.

If the flow value on edge $e^* = (u, v)$ was smaller than the capacity before the change, the maximum flow does not change. Thus we assume that $f_{e*} = c_{e*}$ in the original graph. By reducing c_{e*}, the flow becomes invalid. To make it a valid flow, we first reduce the flow on e^* by one. This change creates an imbalance between the in-flows and out-flows at u and v respectively. To fix that we look in the residual graph and try to find a path from t to v and also a path from u to s, which can be done in $O(m + n)$. Then we decrease the flow on all edges of the two paths from t to v and u to s. This way we resolved the imbalance of in-flow and out-flow at u and v. However the max-flow in the graph is decreased by one.

Now we can check whether the maximum flow in the graph can be the max-flow before decreasing the capacity of edge e^*. For this purpose, we look for an augmenting path from s to t, which can have a flow of at most one. If there exists such an augmenting path, then we increase the flow on the edges of this path by one. Therefore, we have the same max-flow in the graph before decreasing the e^*'s capacity. Otherwise, the new max-flow is less than the previous max-flow by one unit difference.

We can find these augmenting paths by Ford-Fulkerson algorithm which takes $O(m + n)$ time.

Exercise 3: Forward-Only Paths (3 points)

A friend of you has written some very fast maximum flow code. Unfortunately it turns out that the program doesn't always compute a correct maximum flow. When inspecting the solution you realize that your friend's program implements a simplified variant of the Ford-Fulkerson algorithm. When computing augmenting paths, the program only considers forward edges of the residual graph and it does not consider backward edges at all. We have seen in the lecture that backward edges are necessary to get a correct algorithm. However your friend claims that his algorithm (let's call it the forward-edge-only algorithm) always computes a solution that is within a constant factor of the optimal one. That is, there is an absolute constant $b > 1$ such that the forward-edge-only algorithm computes a flow of value at least $1/b$ times the value of an optimal flow. Is your friend right? If yes, prove it, otherwise show that the ratio of the maximum flow value and the flow computed by the forward-edge-only algorithm can be arbitrarily large. Assume that the forward-edge-only implementation always takes an arbitrary (possibly worst-case) augmenting path of only forward edges as long as such an augmenting path exists. You can also assume that all edge capacities are positive integers.

Solution

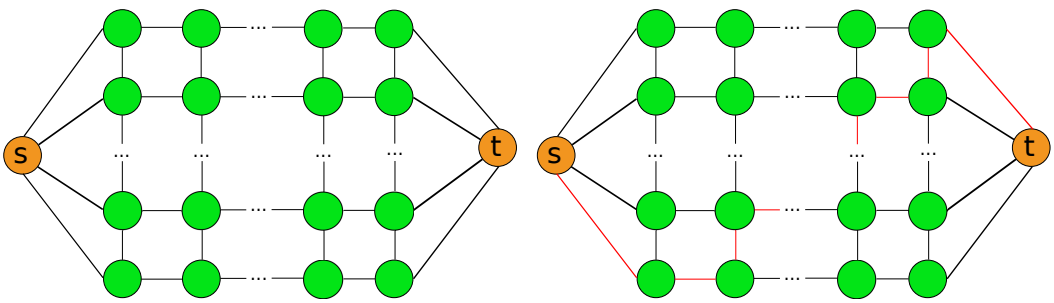


Figure 1: Example of a network (left) and the path (right) to prove the claim is wrong.

Consider a network as shown in Figure 1 on the left. It consists of $N \times N + 2$ vertices - s and t are source resp. sink vertices (shown in orange) and $N \times N$ vertices that are connected only with the direct vertical and horizontal neighbours. Let all edges have capacities 1 and have only left-to-right

and down-to-up directions. The maximum flow value in this network is N (simply use only left-to-right edges when looking for an $s-t$-path). Now consider the *red* path shown in Figure 1 on the right, possibly found by the forward-only algorithm.
If this path is considered first by the algorithm, it blocks all other paths that are considered by the algorithm, as s and t are then disconnected in the "forward-only" residual graph and the returned maximum flow is 1. Since N is arbitrary, the claim that the forward-only algorithm always finds a constant approximation solution is wrong.