# Homework 5

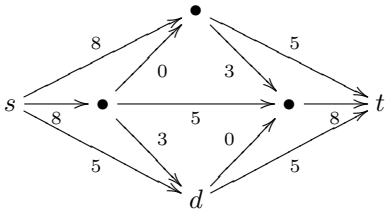
Frederick Robinson

18 February 2010

## 1 Problem 7.3

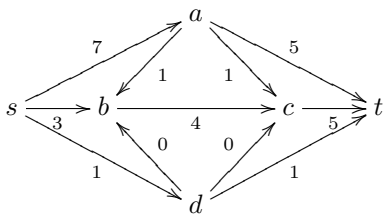
1. It is not entirely clear whether this question refers to Figure 7.26, Figure 7.27 or both. Accordingly I will answer it for bothe 7.26 then 7.27.

   The value of the flow on Figure 7.26 is 18 since this is the total flow into the sink and out of the sorce. Moreover, we observe that it is not the maximum flow. In particular the following flow is larger

   with a flow of 18.

   For Figure 7.27 the value of the flow is just 10, again because this is the amount of flow going out of theo source node, and into the sink node. The current flow is again not maximal. In particular we demonstrate a larger flow as

   This flow has value $11 > 10$.

2. Fortunately since this graph is small we can just check all possibilities. Since in an $s - t$ cut $s$ must be in $A$ and $t \in B$ we can completely specify a cut by whether or not each of the four "middle" vertices is in $A$. In the following table a sequence of 4 numbers either one or zero represents this

in the obvoious order. That is '1010' signifies "$a$ and $c$ are in $A$ while $b$ and $d$ are in $B$.

| Cut | Value |
|---|---|
| 0000 | 14 |
| 0001 | 29 |
| 0010 | 19 |
| 0011 | 31 |
| 0100 | 17 |
| 0101 | 29 |
| 0110 | 16 |
| 0111 | 25 |
| 1000 | 12 |
| 1001 | 27 |
| 1010 | 14 |
| 1011 | 27 |
| 1100 | 14 |
| 1101 | 26 |
| 1110 | 11 |
| 1111 | 20 |

By inspection it is clear that the minimum cut is $\{s, a, b, c\} \cup \{d, t\}$ and that this cut has capacity 11.

# 2 Problem 7.7

## 2.1 Algorithm (Reduction)

I will demonstrate a slightly more general algorithm: it solves the network problem and returns which clients connect to which nodes. It is easy to see however that given a solution in which the most possible clients are connected, if there are disconnected clients then there is no solution in which there are no disconnected clients.

We will reduce this problem to a max-flow problem, then use a max-flow algorithm to solve the problem before converting this solution into a solution of the original problem.

First introduce a source, and sink node. Then, create a directed graph by creating links of capacity 1 from the source to each client. Then, connect each client to every base station in range by links with capacity 1. Finally, connect each base station to the sink with a capacity $L$ edge.

Once the direceted graph has been set up in this manner we run our preexisting max-flow algorithm on it to determine the max flow. To go from the max flow back to a solution for our initial problem we just consider each link from a client to a base station. If the link has flow on it in our maximum flow we add it to the solution. If not, then the link is not used in the optimal solution.

If we don't wish to go back to the soultion we can just check to see if the flow on the graph is equal to the number of clients. If it is then we managed to

connect them all. If not, there is no such connection.

## 2.2 Correctness

A max flow on the graph can be converted to an allowable network solution

*Proof.* Since each client is only connected in the graph to the base stations which are in range it is impossible that a flow on the graph will correspond to a solution of the networking problem in which a client connects to a base station not in range. Moreover, since all the edges have integral weight there is an integral solution to the max flow problem. This, together with the fact that the edges in to each client has value 1 means that the flow coming out of each client (and corresponding the connection that each client makes) goes to precisely one (or perhaps none if the client cannot be connected in an optimal solution) base station.

Finally, we know that no more than the maximum load will connect to a given base station since the flow out (and therefore in, by flow conservation) of a given node is at most $L$ as this is the capacity of the edge going out of each base station. □

Now I prove a short lemma.

**Lemma 2.1.** *The number of clients connected in a solution to the networking problem is precisely the amount of flow on a solution to the flow problem (and conversely the amount of flow on such a solution is the number of clients in the related network problem).*

*Proof.* The amount of flow on a solution to a flow problem is precisely the amount of flow which comes out of the source node in total. However, each edge out of the source node leads into a client, and in the corresponding network problem any client which has flow on it is connected.

Conversely, if a client is connected then he has flow (precisely 1 unit) coming in to him in the corresponding flow problem. Hence, as the only edges into a client come from the source, and the only edges out of the source go to the clients we know that the total number of clients is equal to the amount of flow on the graph. □

Now I claim that a max flow on the graph created as described above is an optimal choice for the network and, conversely that an optimal choicoe on the network is a maximum flow on the graph. If I establish this then my algorithm's correctness follows from the correctness of the max flow algorithm.

I begin by showing ($\Rightarrow$) that a max flow is an optimal solution to the original problem.

*Proof.* Suppose towards a contradiction that there exists a max flow on the graph above say $\mathcal{O}$ which does not correspond to an optimal solution to the network problem. Then, there is some solution to the network problem say $\mathcal{N}$ which causes more clients to be connected than $\mathcal{O}$.

However we can convert this network solution $\mathcal{N}$ to a max flow solution by the procedure given in the algorithm. The corresponding max flow solution must however carry more flow than $\mathcal{O}$ since the number of clients connected and the amount of flow on the network is exactly the same. The is a contradiction since we assumed that $\mathcal{O}$ was a maximal flow. □

I will now demonstrate conversely ($\Leftarrow$) that an optimal solution to the network problem is a maximal flow on the corresponding directed graph.

*Proof.* Suppose that there exists an optimal solution to the network problem which does not correspond to a maximal flow. Then, there exists some flow which has a larger capacity than the number of clients connected in our network solution. This is however, a contradiction since we could just convert this flow to a solution of the network problem which would have more connected clients than our purportedly maximal solution. □

### 2.3 Runtime

The runtime of our algorithm is essentially the time it takes to convert the original problem into a graph, the time it takes to solve the max flow on the graph and the time it takes to convert this solution back to a solution of the original problem.

To convert the original problem to a graph we need at most $n \cdot k$ computations in order to determine which clients are in range of which base stations. (This is the figure we get from naive pairwixe comparison, perhaps it could be improved, but it is polynomial.)

Moreover we know that the network flow algorithm (Ford-Fulkerson) runs in $O(mn)$ time. In this case we have $n = n$ since the max flow cannot be larger than the number of clients. Moreover then numer of edges in our graph cannot be larger than $m = n \cdot k + k + n$ (that is, a connection between each node and each client as well as the connections from the clients to the source and from the nodes to the sink). Thus, Ford-Fulkerson runs in $O(n(n \cdot k + k + n)) = O(n^2k + nk + n^2) = O(n^2k)$ time for our particular problem.

Finally, the procedure to analyze our max flow is either constant time (if all we want is to verify whether or not all clients are connected) or $O(n)$ if we want to record which links are active in our optimal solution.

Thus, together our algorithm runs as $O(nk + n^2k + 1)$ or $O(nk + n^2k + n)$ depending. Both of these reduce to $O(n^2k)$ however.

## 3 Problem 7.10

### 3.1 Algorithm

First we construct a new flow from the old one by reducing the flow we started out with. Explicity, reduce the amount of flow on the reduced edge by one, then traverse the graph following edges with nonzero flow towards the end, reducing

each flow by one. Similarly traverse the graph backwards from the reduced edge: follow edges withnonzer flow reducing the amount of flow on the edge until reach the source.

Intuitively this will result in a graph where all the flow is the same as in the original max flow, but with less on the reduced edge, and corresponding reductions to meet the flow preservation condition.

Now we need only run a search of the new residual graph. If there exists a new path to add we do so and terminate the algorithm. If not we terminate the algorithm.

### 3.2 Correctness

The procedure described above for reducing flow on the original max flow will result in a legal flow which has capacity one less than that of the original max flow.

Since we are guaranteed that the flow is acyclic we know that naively following the flow forwards and backwards will eventually lead to the sorce (sink). Thus, since we have reduced the flow in an entire path from start to end by one we retain a legitimate flow. Moreover since we have reduce by only one we know that the new flow is one less than the original one.

Moreover, since the new graph is the same as the old one, but with reduced capacity its maximum flow is less than or equal to that of the original graph. We have constructed a flow which is one less than that of the original graph. Hence, the new maximum flow will either be equal to the reduced flow from the original graph, or one greater. Therefore if there exists a increase in the flow we will find it on the first breadth first search of the residual graph. If not, there does not exist one.

### 3.3 Runtime

The runtime of our algorithm is $O(2m + n) = O(m + n)$ as claimed since the runtime of breadth first search is $O(m + n)$ and the runtime of the procedure to find a new flow from the original max flow is just $O(m)$.

## 4 Problem 7.13

### 4.1 Algorithm (Reduction)

*Note:* I was not sure whether or not we were to consider edges as having capacities or not. I assumed yes, however if this is incorrect my algorithm is still correct; just set all the edge weights to '$\infty$' (some number we know to be larger than the capacity of our graph) and run the algorithm as described below.

This problem reduces fairly easily to max flow problem we are used to. We simply take the input graph and transform it in the following manner.

For each node $n$ create an additional node say $n'$. Then, create an edge from $n \to n'$ with capacity $c_n$ the same capacity as the original node. Next, connect

any outgoing edges from $n$ to $n'$ instead. That is, if there are any edges going out from $n$ remove them, and create a new edge going from $n'$ to wherever the old edge went, and with the same capacity.

Finally, run the normal max flow algorithm on this new graph. The solution to this problem can be taken back to a solution on the original graph by putting the same amount of flow on the original edges as are on the corresponding edges in the the flow solution we have attained.

## 4.2 Correctness

First we establish

**Lemma 4.1.** *A legal flow on the original graph is legal on the intermediate graph and visa versa*

*Proof.* Suppose we have some flow on the intermediate graph, it is constrained on the edges of the original graph just as before since we have not changed this. However, a flow on this graph which is nominally unconstrained on the vertices is actually limited. In particular note that the flow through one of the vertices can be no more than the limit as, the only flow into (respectively, out of) a node is through the new edges which we have constructed whose capacity corresponds ot that of the node. Thus, since the total flow through a node cannot be more than the total flow in (out) it is limited in the desired manner.

This proves that a legal flow on the intermediate graph is a legal one on the original graph.

Conversely it is easy to see that a legal flow on the original graph is also legal on the intermediate graph. For, on the "real" edges flow is restricted similarly for both, and since flow is restricted through the nodes in the original graph it follows that it is correctly restricted through the netw "artificial" edges in the intermediate graph. □

The lemma proves that a solution obtained in the algorithm described above is a "legal" solution to the problem.

Now I claim that our solution is optimal

*Proof.* Suppose that there were a solution to the problem which we did not find, but which had more flow. Call this $\mathcal{O}$. This would mean that on the intermediate graph there was a flow which was better than the one we found since any legal flow on the original graph is legal on the intermediate graph. This is a contradiction however by correctness of our max flow algorithm. □

## 4.3 Runtime

The algorithm runtime is determined by the time necessary to move from our original graph to the intermediate one, run the Ford-Fulkerson algorithm, then translate this solution back to one of our original problem.

Translating from our original graph to the intermediate on will take no more than $O(n + m)$ time since we only need to operate on each node, or edge a constant number of times.

Similarly, translating back ot our original graph will take only $O(m)$ time since we need only read off the amount of flow on each of our original "real" edges.

Finally running the Ford Fulkerson algorithm will take $O(c(m + n))$ time since the intermedate graph has $m + n$ edges: $m$ "real" ones oand "n" artificial ones.

Thus, all together our algorithm runs in $O(c(m + n) + (n + m) + m) = O(c(m + n))$ time.

## 4.4 Cuts (Max-Flow Min-Cut)

We shall define a cut on such a graph as two sets of edges $A$ and $B$ such that each edge from $s$ is in $A$ and each edge to $t$ is in $B$. We define the capacity of a given cut to be the amount of flow that goes out of edges in $A$ (into edges of $B$)

Now I claim that an analogue of the Max-Flow Min-Cut theorem holds. Namely: the capacity of the maximum flow is the same as that of the minimum cut.

*Proof.* First we establish $MinCut \leq MaxFlow$. This mus be true since, if each cut had a larger capacity than the MaxFlow then we could construct a larger flow in a way similar to that described in the proof of the normal MinCut/MaxFlow Theorem.

Conversely, we have $MinCut \geq MaxFlow$ since if there were a cut with a smaller value than the MaxFlow this would be a contradiction. In particular any flow must cross any cut with capacity at least as large as that of the flow. That is, we must have $c(A, B) \geq Flow$ for any cut. Contradiction. □