Advanced Algorithms and Data Structures
Hand-in 1

Philip Pickering
pgpick@gmx.at

Thomas Bracht Laumann Jespersen
ntl316@alumni.ku.dk

The problem could be formulated and solved as a max-flow problem. There are a few things to observe before that: the network (as in the example) is undirected, in which case we would want to make it directed. We also have several sources that may themselves be connected (such as nodes 2 and 3 in the example).

Let the network be an undirected graph called $N = (V, E)$. We want to construct a max-flow graph $G = (V', E')$ from N. For all nodes $u, v \in V$ and $(u, v) \in E$, we add to our max-flow graph nodes u, v and an extra node w. We add add directed edges $(u, v), (v, w)$ and (w, u). If we let k denote the capacity of the edge $(u, v) \in E$, then we let $c(u, v) = c(v, w) = c(w, u) = k$ in G. Lastly, we add a super source s and an edge (s, v) to all the nodes v that were a source in N.

The number of edges in G is $|V'| = |V| + |E| + 1$ and $|E'| = 3|E| + p$, where p is the number of edges added from the super source s. p is exactly the number of sources in N.

Now we have the original problem stated as a max-flow problem, which can be easily expressed as a linear program

$$\begin{array}{llll}
\text{max.:} & \sum_{v \in V_G} f(s, v) - \sum_{v \in V_G} f(v, s) & & \\
\text{s.t.:} & f(u, v) & \leq & c(u, v), \quad \forall u, v \in V_G \\
& \sum_{v \in V_G} f(v, u) - \sum_{v \in V_G} f(u, v) & = & 0 \quad\quad\quad \forall u, v \in V_G \\
& f(u, v) & = & 0, \quad\quad\quad \forall u, v \in V_G
\end{array}$$

As small example graph can be found in fig.1.

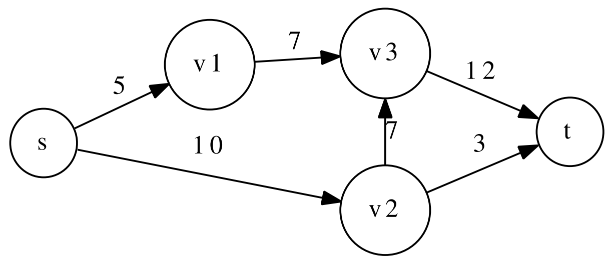

Figure 1: An example max-flow graph

The linear program corresponding to the graph in fig. 1 is the following:

$$
\begin{array}{lrcl}
\text{max.:} & f_{s,v_1} + f_{s,v_2} & & \\
\text{s.t.:} & f_{s,v_1} & \leq & c(s,v_1) = 5 \\
 & f_{s,v_2} & \leq & c(s,v_2) = 10 \\
 & f_{v_1,v_3} & \leq & c(v_1,v_3) = 7 \\
 & f_{v_2,v_3} & \leq & c(v_1,v_3) = 7 \\
 & f_{v_2,t} & \leq & c(v_2,t) = 3 \\
 & f_{v_3,t} & \leq & c(v_3,t) = 12 \\
 & f_{s,v_1} & = & f_{v_1,v_3} \\
 & f_{s,v_2} & = & f_{v_2,v_3} + f_{v_2,t} \\
 & f_{v_1,v_3} + f_{v_2,v_3} & = & f_{v_3,t} \\
 & f_{s,v_1}, f_{s,v_2}, f_{v_1,v_3}, f_{v_2,v_3}, f_{v_2,t}, f_{v_3,t} & \geq & 0
\end{array}
$$

We have minimized the above linear program, such that edges with capacity zero have been omitted. Without showing the work of the simplex algorithm, we arrive at the solution: $f_{s,v_1} = 5, f_{s,v_2} = 10$.

In order to find the cheapest critical connection we have to identify the *minimal cuts*. Intuitively, if we have a cut (S,T) of our graph and a maximum flow f in that same graph, then the flow from S to T must be equal to capacity across that cut, $c(S,T)$. This means in order to increase the overall flow of the graph, we have to increase the capacities on some of the edges of our minimal cuts.

If we only have one such minimal cut, then we simply select the edge with the smallest capacity and increase its capacity. If we have more than one cut, it's not immediately clear which edges to choose. In our example above, we can identify three minimal cuts:

1. $S_1 = \{s\}$, $T_1 = V - \{s\}$
2. $S_2 = \{s, v_2\}$, $T_2 = \{v_1, v_3, t\}$

3. $S_3 = V - \{t\}$, $T_3 = \{t\}$.

In the first cut we can identify the edge (s, v_1), $c(s, v_1) = 5$, as the cheapest to increase, for (S_2, T_2) the cheapest edge is (v_2, t), $c(v_2, t) = 3$, and this is also the cheapest to increase for (S_3, T_3). If we increase the capacities of these two edges by 1, then we have increased the capacities of all the minimal cuts (S_i, T_i) by 1. In this example the above cuts are still minimal cuts, so the max-flow min-cut theorem tells us that we have increased the maximum flow by 1 (otherwise we should have provoked another minimal cut).

Recaptured briefly, in order to find the "cheapest critical connection", we need to inspect the minimal cuts and for each of them, select the edge with the smallest capacity to increase. In the end, by the max-flow min-cut property, we will have increased the maximum flow in the entire graph.

The dual of the above stated problem can be stated by restating the above program in standard form $Ax \leq b$ along with a coefficent vector $c = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}^T$. The dual is then given by $A^T y \geq c$, where we minimize $b^T y$. y is a vector of size proportional to the number of constraints, in our case 12, because each of the equality constraints give rise to two inequalities. Now we can state the dual:

$$
\begin{aligned}
\text{min.:} \quad & 5y_1 + 10y_2 + 7y_3 + 7y_4 + 3y_5 + 12y_6 \\
\text{s.t.:} \quad & \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 1
\end{bmatrix} y \geq \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}
\end{aligned}
$$

A solution to the dual problem is $y_1 = 1, y_2 = 1$ and the rest are zero.

In order to model that some of the relay stations aren't owned by the company, we can rephrase the problem as a *minimum-cost flow* problem (29.51), where the cost function:

$$
a(u, v) = \begin{cases} 1, & \text{if } u \text{ or } v \text{ is not owned by the company} \\ 0, & \text{otherwise.} \end{cases}
$$

Then we can directly apply the definition from the book, because we can require the value of the flow to be exactly d.