CS 522: Data Structures and Algorithms II
Homework 2

Dustin Ingram

February 11, 2013

1. **Solution:** We split the edge (u, v):

$$\begin{tikzcd} {} \arrow[r] & u \arrow[r, "c(u,v)"] & v \arrow[r] & {} \end{tikzcd}$$

Into edges (u, x) and (x, v):

$$\begin{tikzcd} {} \arrow[r] & u \arrow[r, "c(u,x)"] & x \arrow[r, "c(x,v)"] & v \arrow[r] & {} \end{tikzcd}$$

Such that:

$$c(u, x) = c(x, v) = c(u, v)$$

Since the amount of potential flow into and out of x is the same, one can show that it is equivalent with the edge (u, v). There are three cases to consider:

(a) $f(s, u) = 0$: If there is no flow into u, there will be no flow between u and x, and between x and v, and the maximum flow in G' remains the same.

(b) $f(s, u) < c(u, v)$: If the flow to u is less than the potential maximum flow on (u, x) and (x, v), since there is no other source of flow for the path, the maximum flow in G' remains the same.

(c) $f(s, u) = c(u, v)$: If the flow to u is equivalent to the potential maximum flow of (u, v), and thus (u, x) and (x, v), the flows will also be maximized across these edges and the maximum flow in G' remains the same.

Essentially, any consecutive sets of edges which constitute a single path, which do not share any vertex with any other edges, and which have the same maximum flow can be simplified into a single edge and the maximum flow of the graph will remain the same.

2. **Solution:** We extend the flow properties and definitions as follows:

   (a) **Capacity constraint:** Does not change, edges still cannot exceed their maximum potential flow.

   (b) **Flow conservation:** This property does not change, and is still true for all $u \in V - s, t$ (including super-sources and super-sinks).

   (c) **Flow value:** In the original graph, the flow value is defined as:

   $$f = \sum_{v \in V} f(s, v)$$

   The flow from the super-source s_s to the regular sources $S \subset V$, and from the regular sinks $T \subset V$ to the super-sink t_s, is the same as the flow across the graph. Therefore:

   $$f = \sum_{v \in V} f(s, v) = \sum_{s \in S} f(s_s, s) = \sum_{t \in T} f(t, t_s)$$

   Thus:

   $$f = \sum_{v \in V} f(s, v) = \sum_{v \in V'} f(s_s, t_s)$$

3. **Solution:** If there is a vertex u on the graph G such that there is no path $s \rightsquigarrow u \rightsquigarrow t$, the maximum flow of G must have $f(u, v) = f(v, u) = 0$ for all vertices $v \in V$. If there is not a path $s \rightsquigarrow u \rightsquigarrow t$, this means that for any path $s \rightsquigarrow v \rightsquigarrow u$, there is no path $s \rightsquigarrow v \rightsquigarrow u \rightsquigarrow t$. Thus any flow along this path would not arrive at the sink t, and thus:

   $$\sum_{v \in V} f(s, v) \neq \sum_{v \in V} f(v, t)$$

   Specifically because:

   $$f(s, u) \neq f(u, t)$$

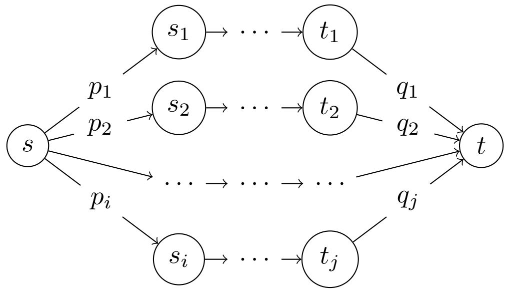
   Where u is the violating vertex. The only case where this would not violate this maximum-flow property is that in which:

   $$f(u, v) = f(v, u) = 0, v \in V$$

   Which makes the flow $f(s, v)$ equivalent to the (non-existent) flow $f(v, t)$, and satisfies:

   $$\sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$$

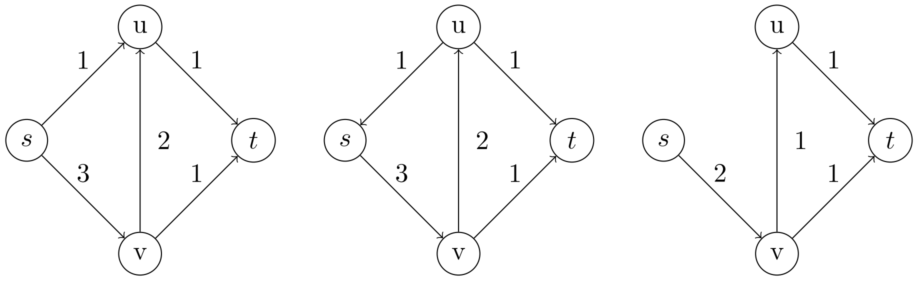
4. **Solution:** The graph G of the flow network with multiple sources s_i and sinks t_j is as follows:



For every source, there is a unit of flow p_i which is delivered to the graph, and for every sink, there is a unit of flow q_j which leaves the graph. Since these values are guaranteed for each source or sink vertex, we can reduce the problem of finding a flow f in G to a single-source, single-sink problem by converting all s_i and t_j to regular vertices on the graph, and converting the edges from $s \to s_i$ to regular edges, such that $c(s, s_i) = p_i$ and similarly the edges from $t_j \to t$ to edges such that $c(t_j, t) = q_j$.

5. **Solution:** Using the Max-Flow algorithm, we can determine the edge connectivity of the undirected graph G with vertices V and edges E as follows. First, we select any two vertices $s, t \in V$ as the source and sink. Then, for every undirected edge $(u, v) \in E$, we create the directed edges (u, v) and (v, u) and make their capacity such that $c(u, v) = c(v, u) = 1$. We then use $|V|$ applications of the Max-Flow algorithm to determine the maximum total flow of the graph, which is equivalent to the edge connectivity of the original undirected graph G.

6. **Solution:** An edge entering the source on a flow network G represents a cycle between the source s and any number of vertices $v \in V$. The figures below provide an example for G:



Here, the first graph is G with all of it's original edges and their maximum capacities. The second graph is an example flow f of G in which the max flow $f = 2$, and $f'(v, s) = 1$. The third graph is another flow f' with the cycle removed, such that $f' = 2$ is still true, however $f'(v, s) = 0$.

A cycle $s \rightsquigarrow s$ represents "wasted" flow in the flow network of G, as the flow is not entering the sink t but instead returning to the vertex s. An algorithm to compute f' from f removes this cycle:

(a) Identify a path p_{cycle} in G such that $s \rightsquigarrow s$;

(b) Determine the minimum capacity c_{cycle} of all $e \in p_{cycle}$;

(c) For every $e \in p_{cycle}$, reduce the flow of e by c_{cycle}.

Since we are removing the entire capacity from at least one edge in the cycle, we are breaking the cycle and eliminating the "wasted" flow. Since we are traversing at most E edges, the complexity of this algorithm would be $O(E)$.

In the figures above, the path of the cycle is $p_{cycle} = s \to u \to v$, and the maximum capacity of the cycle is $c_{cycle} = 1$.

7. **Solution:**

   (a) In a flow network in which vertices *and* edges have capacities, it is trivial to reduce the network to an ordinary maximum flow problem on a flow network of a comparable size. Each vertex which has a capacity can be converted into a pair of vertices with a single edge between them with a capacity equal to the capacity of that vertex. Each edge entering the original vertex would enter one node, and each edge exiting the original vertex would exit the second. This would result in an increase of vertices and edges from $G(V, E)$ to at most $G(V + V, E + V)$.

   (b) To solve the escape problem, we can reduce it to a form of a flow network and solve it via MAX-FLOW. To convert the undirected $n \times n$ escape grid into a flow network, we must take the following into consideration:

      i. Paths may not pass through unoccupied vertices more than once;
      ii. Paths may not pass through edges more than once;
      iii. Every occupied vertex must be able to escape;
      iv. Every unoccupied border vertex is a goal destination.

      To satisfy these restraints, we modify the grid as follows:

      i. Every unoccupied vertex is given a capacity of $c = 1$, and reduced as described in (a);
      ii. Every undirected edge is given has a bi-directional capacity $c = 1$;
      iii. A new vertex, s is added to the graph, and given an edge to every occupied vertex in the grid for which $c = \infty$, which will be the source;

iv. A new vertex, t is added to the graph, and given an edge to every $4n-4$ border vertices for which $c=\infty$, which will be the sink.

One can then use MAX-FLOW to find the maximum flow through the new graph. A graph which is fully "escapable" will have a maximum flow equal to the number of starting points within the graph.

Complexity: In the original grid, there were n^2 nodes and $2n^2-2n$ edges. After converting the grid to a simple flow network, we have increased the vertices by at most n^2 (by splitting capacitive vertices) and the edges by n^2 (by adding edges to the source) plus $4n-4$ (by adding edges to the border vertices) plus an additional $2n^2-2n$ (by splitting all undirected edges). Thus, the number of vertices and edges in the flow network are:

$$V = (n^2) + (n^2) = 2n^2 = O(n^2)$$
$$E = (2n^2 - 2n) + (n^2) + (4n-4) + (2n^2 - 2n) = 5n^2 - 4 = O(n^2)$$

Thus if we were to use an algorithm such as the FORD-FULKERSON algorithm, the total runtime would be:

$$O(E|f^*|) = O(n^2 \cdot 4n - 4) = O(n^3)$$

Where $f^* = 4n-4$ is the maximum flow, which is limited by the number of possible escape vertices on the border of the grid.

8. **Solution:**

(a) We turn G into a bipartite graph $G' = (V', E')$ such that:

$$V' = \{x_0, x_1, \ldots, x_n\} \cup \{y_0, y_1, \ldots, y_n\}$$
$$E' = \{(x_0, x_i) : i \in V\} \cup \{(y_i, y_0) : i \in V\} \cup \{(x_i, y_j) : (i,j) \in E\}$$

We then assign a capacity $c=1$ to every edge in the graph, and use MAX-FLOW to determine the maximum flow of G', using x_0 as the source and y_0 as the sink. The edges which cross the bipartite graph (that is edges of the form (x_i, y_j)) correspond to edges in the optimal path cover of G.

(b) The algorithm will only work on directed acyclic graphs. This is because the algorithm will complete every cycle in the resulting graph to increase the flow, even though cycles do not exist in an optimal path covering (they have one more edge than is necessary).