CS 599: Algorithms **(Due: November 24th, 2020)**

Submission Homework 8

Instructor: Prof. Prasad Tadepalli *Name:* Rashmi Jadhav, *Student ID:* 934-069-574

1: Ford-Fulkerson algorithm for maximum flow

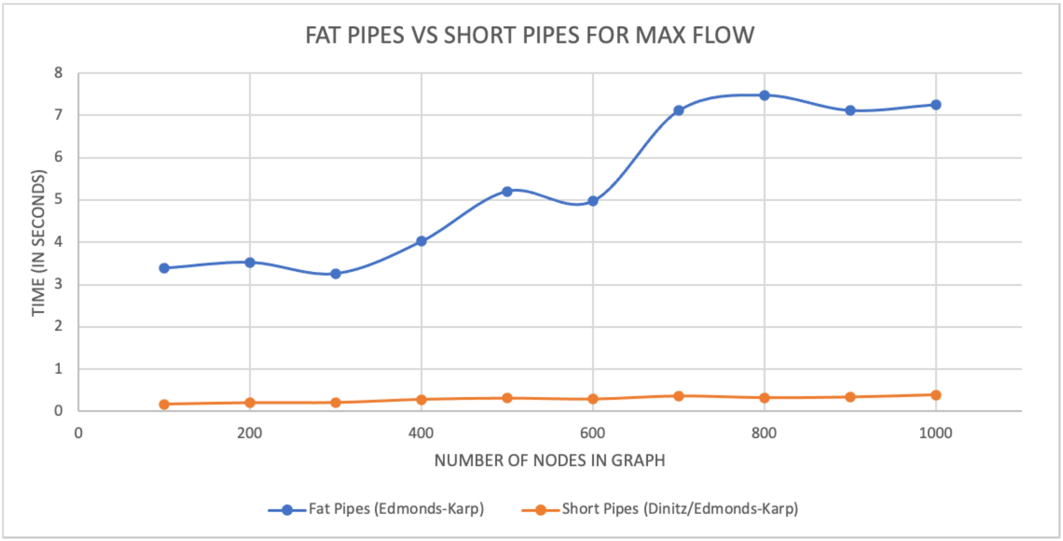
Ford-Fulkerson algorithm for maximum flow was implemented with two approaches, ”fat pipes” and ”short pipes”. To verify if the algorithms were working fine, the given 2 examples were run which gave the expected output.

```
---------------------------Example 1: Fat Pipes:---------------------------
(3, [(0, 1, 1), (0, 2, 2), (1, 3, 1), (2, 3, 2)], 0.0004241466522216797)
---------------------------Example 2: Fat Pipes:---------------------------
(8, [(0, 1, 2), (0, 3, 6), (1, 4, 2), (3, 4, 6)], 0.00023865699768066406)
---------------------------Example 1: Short Pipes:---------------------------
(3, [(0, 1, 1), (0, 2, 2), (1, 3, 1), (2, 3, 2)], 0.00011086463928222656)
---------------------------Example 2: Short Pipes:---------------------------
(8, [(0, 1, 2), (0, 3, 6), (1, 4, 2), (3, 4, 6)], 0.0001220703125)
```

10 random graphs having nodes 100, 200, ..., 1000 were generated as specified to compare the performance of the two algorithms. We observe that both algorithms always give same max flow for the same input graph. Following is the output for the same:

```
   ALGORITHM| NODES IN GRAPH|  MAX FLOW|   TIME TAKEN (in s)
   Fat pipes|            100|       406| 0.15137696266174316
 Short pipes|            100|       406|0.009048938751220703
   Fat pipes|            200|       328|  0.5750589370727539
 Short pipes|            200|       328|0.017666101455688477
   Fat pipes|            300|       351|  0.2107248306274414
 Short pipes|            300|       351|0.029652118682861328
   Fat pipes|            400|       532|  0.8209347724914551
 Short pipes|            400|       532|0.041352033615112305
   Fat pipes|            500|       763|  1.3333539962768555
 Short pipes|            500|       763| 0.07894706726074219
   Fat pipes|            600|       627|  1.6474571228027344
 Short pipes|            600|       627| 0.08412528038024902
   Fat pipes|            700|       596|  1.5601661205291748
 Short pipes|            700|       596| 0.14328408241271973
   Fat pipes|            800|       592|  3.4482600688934326
 Short pipes|            800|       592| 0.12245702743530273
   Fat pipes|            900|       497|  1.4972071647644043
 Short pipes|            900|       497| 0.11134099960327148
   Fat pipes|           1000|       683|   2.760190963745117
 Short pipes|           1000|       683| 0.14847683906555176
```

Plotting the time taken by both algorithms as a function of number of nodes, we observe that fat pipes is performing slower than short pipe. The obvious winner is short pipes. While fat pipes chooses the augmenting path with largest bottleneck value, short pipes chooses the augmenting path with fewest edges. For the fraction of the times the actual flows along each edge are also the same for both approaches for graphs of each size, I couldn't really observe any trend and the actual flows along each edge were almost never the same for both the approaches.



2: Conference Paper Reviewing Problem

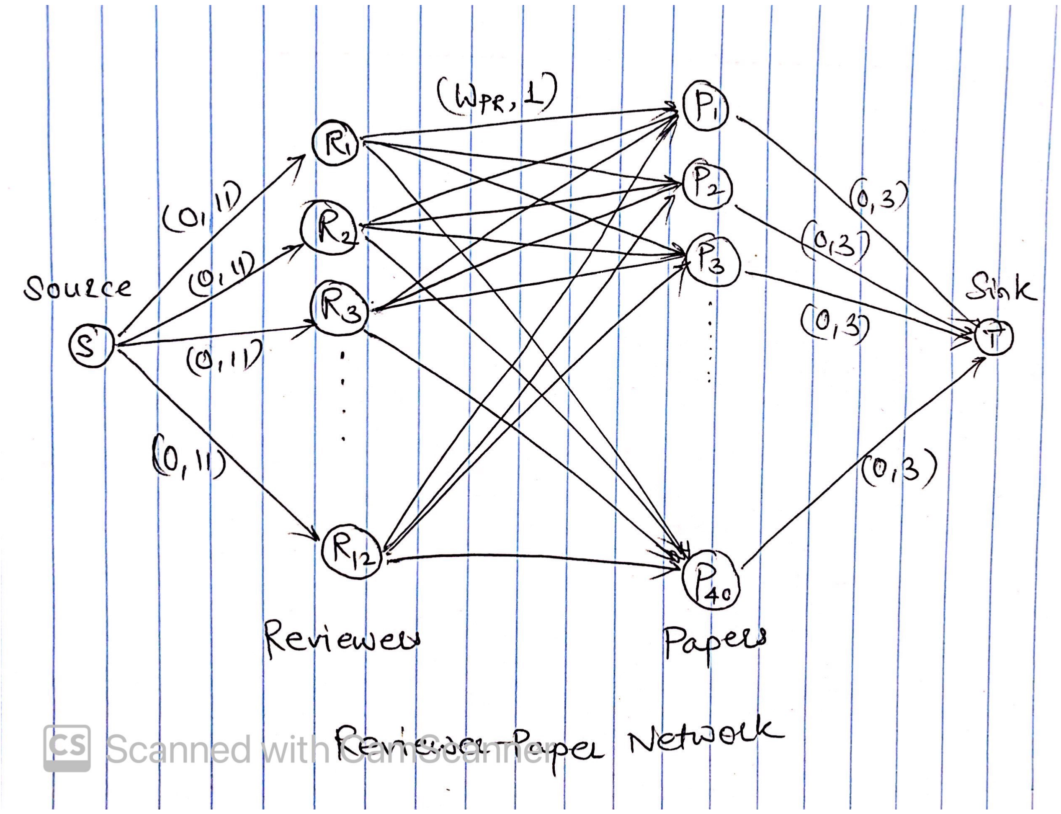
Conference with 40 papers, 12 reviewers, one reviewer's load 11 papers, each paper should be reviewed by 3 reviewers, each reviewer bids for 20 papers. We can formulate this as a network flow problem:

First off, there would be a set of nodes with one node per reviewer, thus there would be 12 such nodes. Then we would have a set of nodes with one node per paper, which means there would be 40 such nodes. Finally, there would be a source node and a sink node, leading to a total of 40 + 12 + 2 = 54 nodes.

There would be three different kinds of edges between these nodes.

1. A set of edges with capacity that link each reviewer node to each paper node. (12 x 40)
2. A set of edges with capacity 11 that link source node to each of the 12 reviewer nodes.
3. A set of edges with capacity 3 linking each of the 40 paper nodes to the sink node.

We see that the capacity constraints between the paper layer and the sink node enforces a constraint of exactly 3 reviews for each submitted paper (40). The expected maximum flow in this network design would thus be 40 x 3 = 120.



$\boxed{\textbf{3: Hall's Theorem}}$

G is a bipartite graph $(Boys, Girls, E)$ where $|Boys| = N, |Girls| = N$. G has a perfect match if and only if $\forall S \subseteq Boys, |S| \leq |N(S)|$.
Proof : If we have subset $S \subseteq Boys$ with $|S| > N(S)$ then there would be no way to match up all of the $|S|$ boys of S along edges with girls in $Girls$ without connecting some boys more than once.

To prove that the condition is sufficient, we use max-flow min-cut theorem. Take given graph G and add in a source vertex s and a sink vertex t. Connect s to every boy of $Boys$ by adding N edges and similarly connect t to every girl by adding another N edges. The capacities between nodes are:
$c_{s,b} = 1$ for any $b \in Boys$
$c_{g,t} = 1$ for any $g \in Girls$
$c_{b,g} =$ inf for any $b \in Boys, g \in Girls$, such that $\{b, g\} \in E(G)$
$c_{g,b} = 0$ for any $b \in Boys, g \in Girls$, such that $\{b, g\} \in E(G)$ (disallow flow from Girls to Boys)
$c_{x,y} = 0$ for any $\{x, y\} \notin E(G)$

By Ford-Fulkerson, there is a maximal flow f and minimal cut S on this graph, with $c(S) = f(s)$. f only has value 0 or 1 on any edge. Now, let M be collection of all edges $\{b, g\}$ on which $f_{b,g} == 1$. We can observe that this is a matching. Because the net flow into any $g \in Girls$, there is at most one $\{b, g\} \in M$ with $f_{b,g}$.

We now claim that M is a perfect matching. $|M| = N$. Note that $|M| = f(s)$. We also know that $f(s) = c(S)$ where S is minimal cut.

Therefore, if we can prove that the capacity of any cut in our graph is at least N, then we are done, as this will prove that $|M| = f(s) = c(S) \geq N$!

Let $X = S \cap Boys$ and $Y = S \cap Girls$. We know that the initial capacities in G are infinite; consequently, if $x \in X$ has a neighbor $Y \in Girls$ such that $y \notin Y$, the capacity of this cut would be infinite.

But we know that there is a finite cut. $\{s\}$ has capacity N as s is connected to N other vertices by edges of capacity 1. Therefore if S is a minimal cut, then if $x \in X$ has $y \in N(x)$, we must have $y \in Y$. In other words, $N(X) \subseteq Y$ and thus no edge in $E(G)$ has $b \in S, g \notin S$. Our edge must either be $\{s, b\}$ for some $b \in Boys/X$ or it could be $\{y, t\}$ for some $y \in Y$!.

We look at c(S), we observe that, $c(S) = \sum c_{v,w}$
$\implies c(S) = \sum c(s, v) + \sum c(y, t)$
$\implies c(S) = \sum 1 + \sum 1$
$\implies c(S) = |Boys/X| + |Y|$
$\implies c(S) = |Boys - X| + |N(X)|$
$\implies c(S) \leq N - |X| + |N(X)|$
$\implies c(S) \leq N - |X| + |X|$
$\implies c(S) \leq N$
Thus, any cut has a capacity at least N and we have proven the claim.

4: Shortest path problem as a linear program

We have a directed graph $G(V, E)$ where each edge(u, v) has $l[u, vl > 0$, a source vertex s and a destination vertex t. To formulate shortest distance problem, we need to calculate $d[t]$ which is the weight of the shortest path from s to t. Now we need to identify a set of variables and constraints that define when we have a shortest path from s to t. The Bellman-Ford algorithm computes this for us. When the algorithm terminates, it has computed $d[v]$ for each vertex v such that for each $edge(u, v) \in E$, we have $d[v] \leq d[u] + l[u, v]$. The source vertex initially has $d[s] = 0$, which never changes. Thus we obtain the following linear program to compute shortest path from s to t in G:

$minimize \;\; d[t]$
$subject \;\; to$
$d[v] \leq d[u] + l[u, v]$ for each $edge(u, v) \in E$,
$d[s] = 0$

In this linear program, there are $|V|$ variables, one for each vertex $v \in V$. There are $|E| + 1$ constraints, one per edge and the additional constraint of source vertex being 0.

The dual of this shortest-path linear program is equivalent to the following:
maximize $X(s) - X(t)$
subject to $X[u] - X[v] \leq l[u, v]$ for all $edges(u, v) \in E$

There is one variable $X(u)$ for each variable u. We can interpret this program as the graph being embedded on a line where $X(u)$ represents the position of vertex u on the line subject to the constraints that for every edge u, v vertex v is prevented from being put more than $l[u, v]$ units farther along the line than vertex u.

Alternate interpretation: Replace each edge in the graph by a rope of appropriate length. Then, starting with all vertices in a pile on the table, gradually lift up vertex s, letting other vertices be pulled down towards the table. As s lifts, the rope will cause the other vertices to lift too. A vertex u will start lifting when the strings along the shortest path from s to u are all *tight*.