Average Runtimes of Max-Flow Algorithms
Math566 – Fall 2016—Project

Altansuren Tumurbaatar, altaa.tumurbaatar@wsu.edu, Washington State University

1 Random graph

Input: n, m, l, U, p, U_{min}
Output: A random graph in FStar format
Where:
n-Number of nodes
m-Number of arcs
l - Number of layers
U - Maximum capacity
p - probability distribution for different types of arcs
U_{min} - Minimum capacity, default $U_{min} = 1$
Given the input arguments, we do the following steps:

1. Node distribution for l layers.
   Layer0 has the source node, $s = 1$, the last layers has the sink node $t = n$.
   All the other nodes are distributed randomly into $l - 2$ layers. We use a dictionary for the layer and node correspondence.

2. Arc distribution for l layers.
   Layer0 is the top most layer and Layer-$(l - 1)$ is the bottom layer. Firstly, the number of arcs in each layer is chosen randomly (besides the last layer) depending on number of nodes in each layer and the number of layers.
   Secondly, we loop over the layers from the bottom to the top while assigning the correct number of arcs in each iteration.
   To do that, we have a policy to allocate some of arcs in m_i with less randomness such that it is guaranteed to be there for connectivity and the remaining arcs to be chosen randomly with random tails and random heads, furthermore with random arc-types. The arc-types are chosen randomly following the probability distribution given as p.
   Layer0 has out-arcs to every node in Layer1. The remaining arcs in Layer0 are out-arcs with type-2, type-3 with random heads.
   Arctypes:

   - 1 : An arc going 1 layer down
   - 2 : An arc going 2 layers down
   - 3 : An arc going 3 layers down
   - -1 : An arc going up. How many layers up(either 1,2,3) is decided randomly.

The last layer has in-arcs from every node in the previous layer.
All the other layers in between the first and the last layers has at least one in-arc and at least one out-arc coming from the layer below or connecting to the layer below respectively. The remaining arcs (if there is any) have random arc-types with random heads and random tails. The arc-types are decided by the arc-type probability, p given. Out of random arcs, type-1, type-2, type-3, type-(-1) arcs have $0.1, 0.3, 0.2, 0.4$ probabilities.
The algorithm start from the lowest layer and run till the top layer.
Special mechanisms used for the implementation:

(a) FIFO LIST We use FIFO LIST to add at least one in-arcs and at least one out-arc connecting to the layer below itself for layers except the first and the last.
LIST0 = "Nodes in Layer-i"
LIST1 = "Nodes in Layer-$(i+1)$"
We pop the first node in LIST0 as i-node and append it at the end of LIST0. We pop the first node in LIST1 as j-node and append it at the end of LIST1. The arc (i-node, j-node) is added to the graph as well as the reverse arc (j-node, i-node). As a result, every node in Layer-i has at-least one in-arc and at least one out-arc connecting to Layer-$(i+1)$.

(b) Policy to choose zero-in-degree or zero-out-degree nodes first.
We keep track of the in-degrees and the out-degrees for all nodes. So that, it is used for reinforcing the random arcs to choose heads and tails out of zero-in-degree and zero-out-degree nodes first (respectively, if there are any) to keep the graph more practical rather than having too much of randomness. If there is no zero-in-degree or zero-out-degree nodes, the candidates have the equal probability to be chosen.

3. Capacities
After we get the random graph, we generate m the random capacities within the given U_{min} and U where m is the number of arcs.

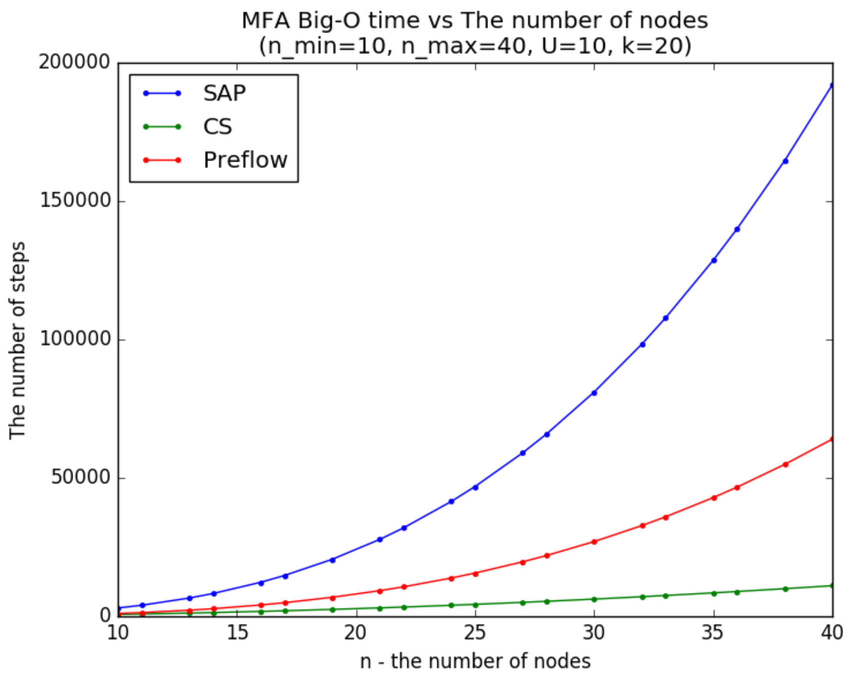
The random graphs work suitably for the context of network optimization because the structures of network flow problems usually have a layer by a layer pattern with a flow running through them. There is a consideration that the graphs can't be a very dense or complete graph since we don't add arcs between nodes in the same layer. High number of arcs sometimes work, but it is not suitable for the analysis since there could be a situation in which there is a layer with a fewer nodes, but a large number of arcs is allocated to the layer because of the randomness. Overall, the random graphs are good enough to analyze the max flow algorithms.

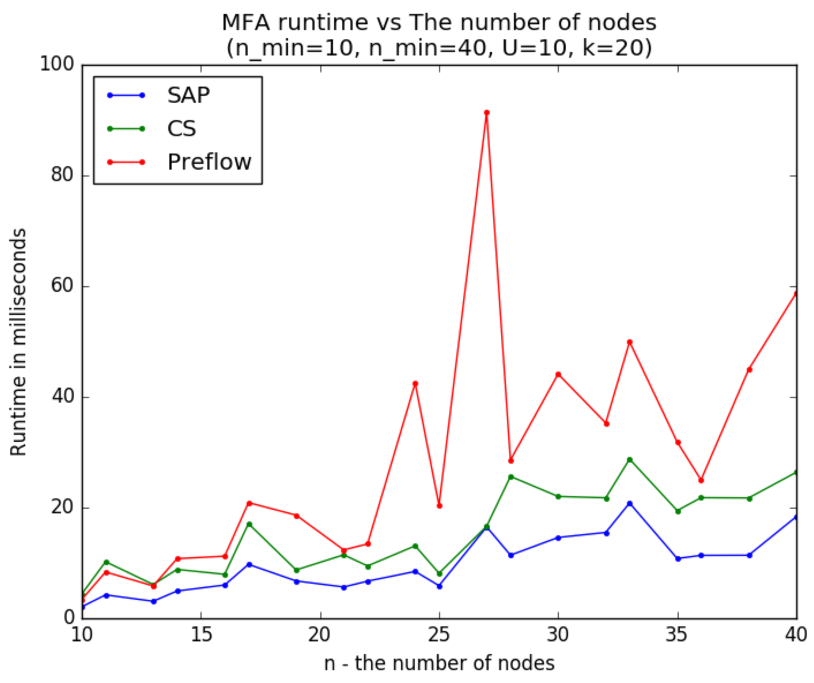
2 Runtime Analysis in terms of Number of nodes

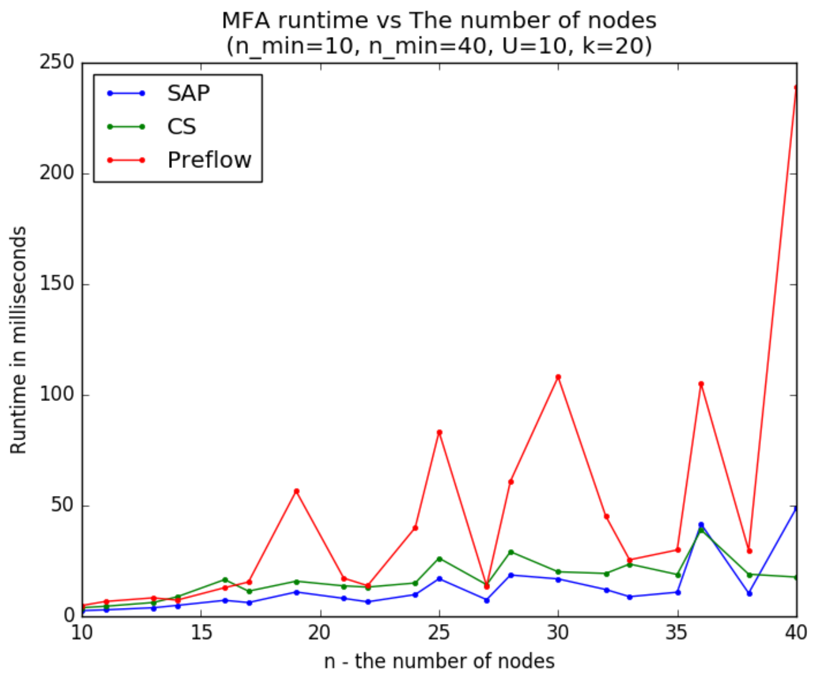
In practice, SAP-MaxFlow algorithm usually over-performs the other two. FIFOPreflow-MaxFlow algorithm performs slower in most cases, and sometimes deviates high with longer runtime from the other two. Please see the Big O times of the algorithms and the typical runtime patterns observed from the experiment in the following figures where k is the number of random graph. Each figure is for a different set of random graphs.

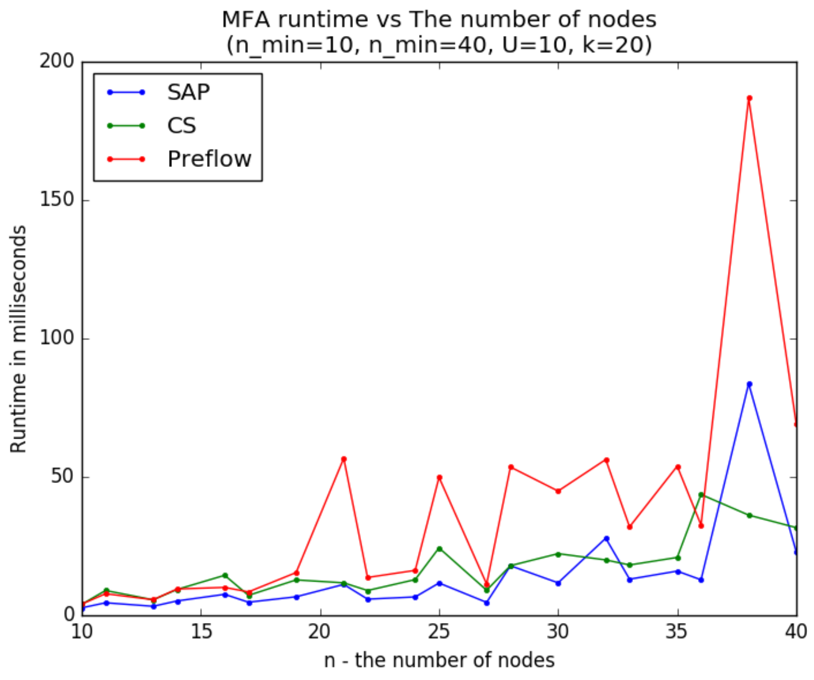
The experiment settings:

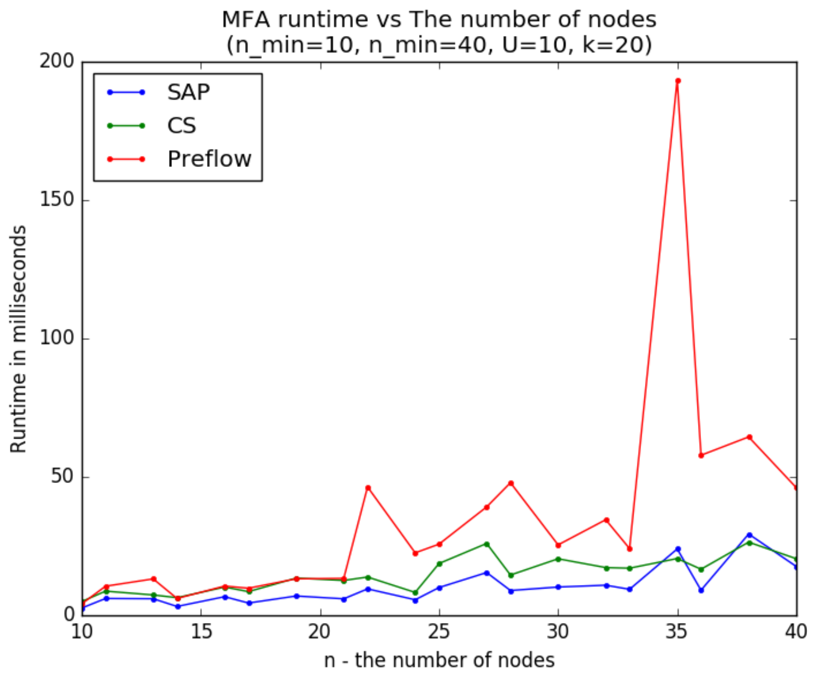
We increase the number of nodes while fixing the maximum capacity constant and the number of arcs to be scaled the same depending on n such that $m = 2n = m_{scale} \times n$. The m_{scale} can be changed.

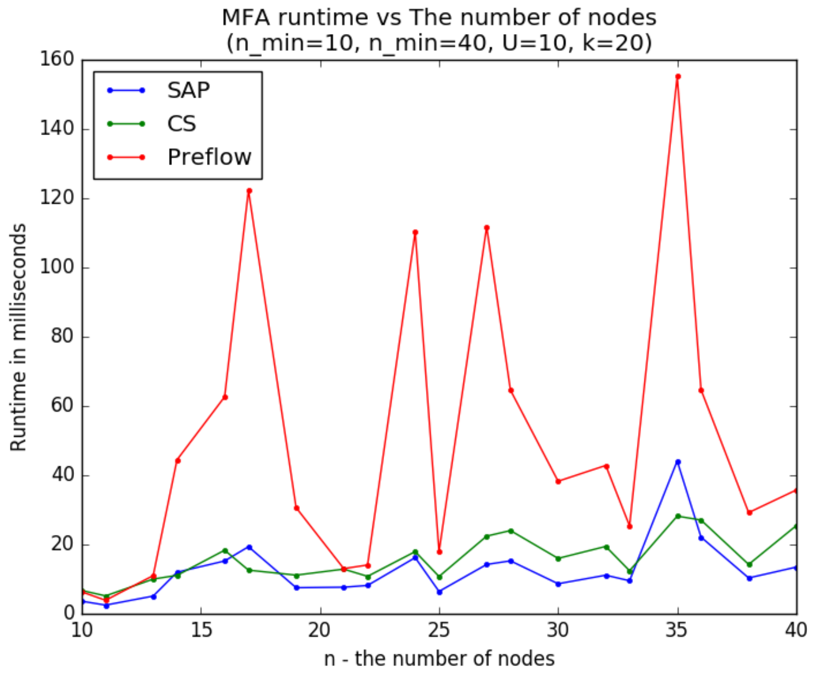


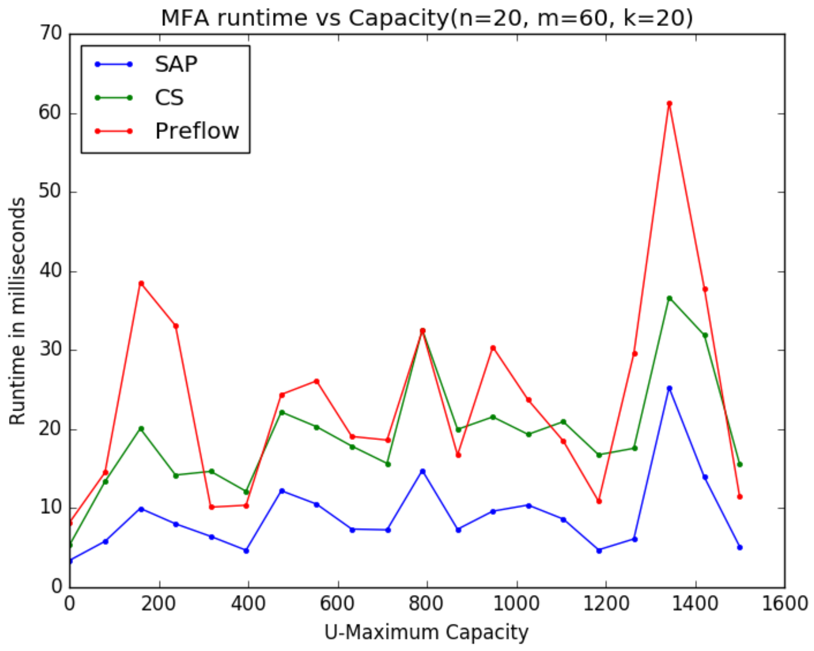

MFA runtime vs The number of nodes
(n_min=10, n_min=40, U=10, k=20)
SAP
CS
Preflow
Runtime in milliseconds
n - the number of nodes
0
20
40
60
80
100
10
15
20
25
30
35
40


MFA runtime vs The number of nodes
(n_min=10, n_min=40, U=10, k=20)
SAP
CS
Preflow
Runtime in milliseconds
n - the number of nodes
0
50
100
150
200
250
10
15
20
25
30
35
40


MFA runtime vs The number of nodes
(n_min=10, n_min=40, U=10, k=20)
SAP
CS
Preflow
Runtime in milliseconds
n - the number of nodes
0
50
100
150
200
10
15
20
25
30
35
40

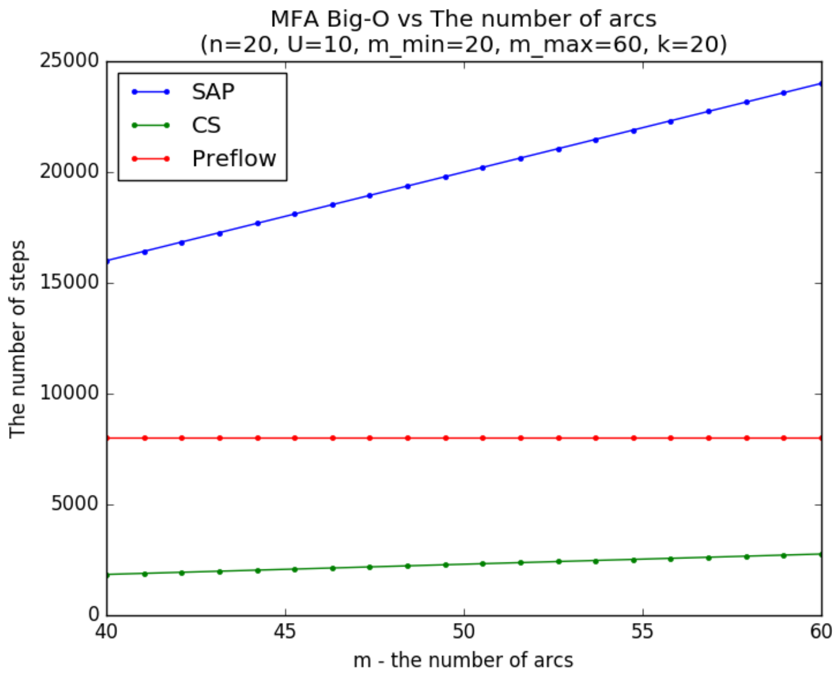

MFA runtime vs The number of nodes
(n_min=10, n_min=40, U=10, k=20)
SAP
CS
Preflow
Runtime in milliseconds
n - the number of nodes
0
50
100
150
200
10
15
20
25
30
35
40

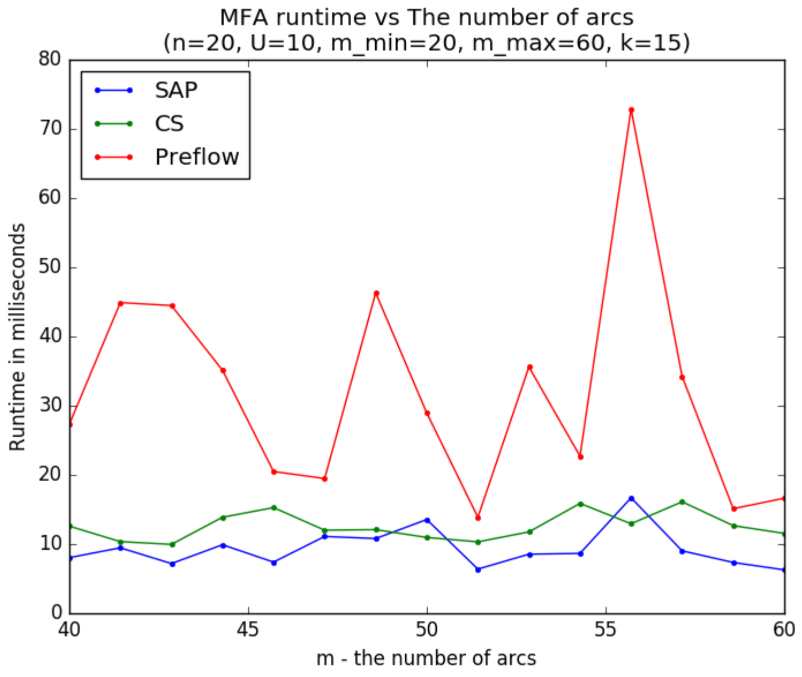

MFA runtime vs The number of nodes
(n_min=10, n_min=40, U=10, k=20)
SAP
CS
Preflow
Runtime in milliseconds
n - the number of nodes
0
20
40
60
80
100
120
140
160
10
15
20
25
30
35
40

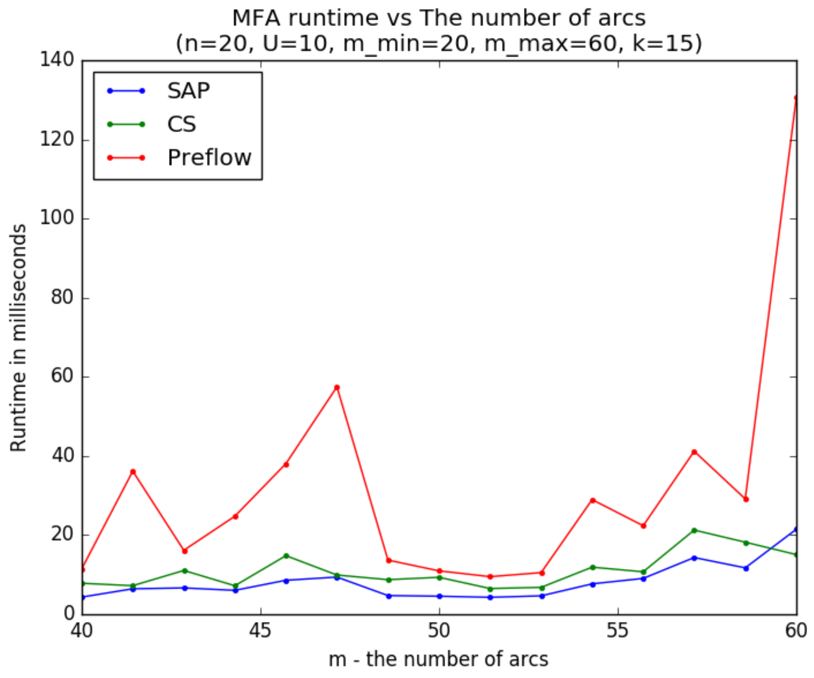

MFA runtime vs Capacity(n=20, m=60, k=20)
SAP
CS
Preflow
Runtime in milliseconds
U-Maximum Capacity
0
10
20
30
40
50
60
70
0
200
400
600
800
1000
1200
1400
1600

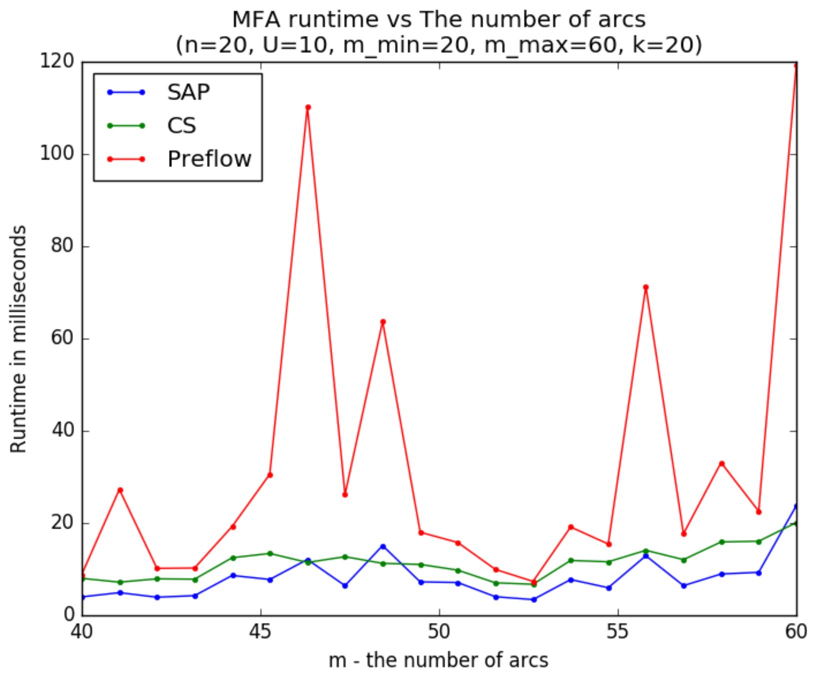
3 Runtime Analysis in terms of Number of arcs

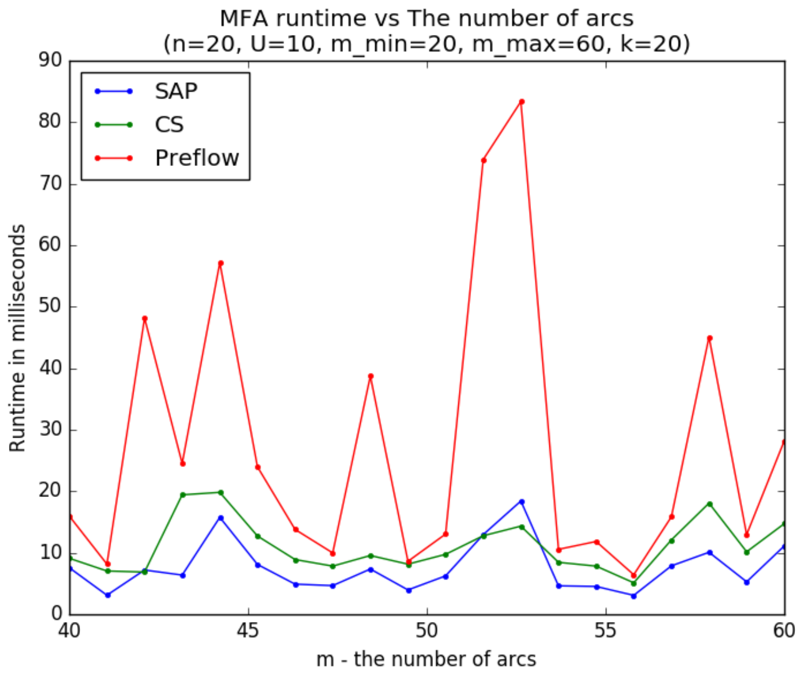
Average runtime in practice:
The best: SAP-MaxFlow algorithm
The second best: Capacity scaling MaxFlow algorithm
The slowest: The FIFOPreflow-MaxFlow algorithm. Though sometimes it performs the same as the previous two.
Please see the Big O times of the algorithms and the typical runtime patterns observed from the experiment in the following figures where k is the number of random graph. Each figure is for a different set of random graphs.
The experiment settings:
We fix the number of nodes large enough and the maximum capacity constant. Then we vary the number of arcs upto $m = 3n$.

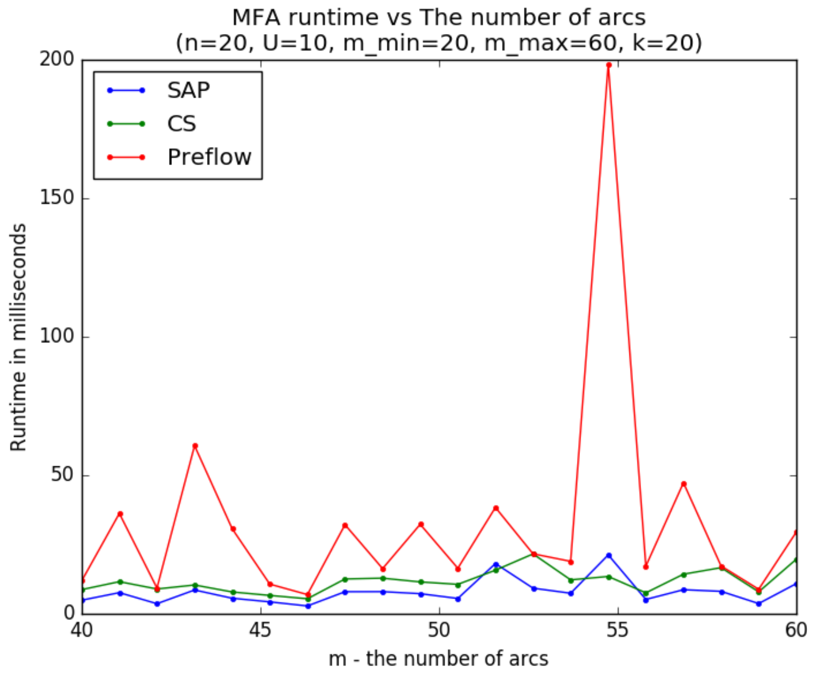



MFA runtime vs The number of arcs
(n=20, U=10, m_min=20, m_max=60, k=15)
SAP
CS
Preflow
Runtime in milliseconds
m - the number of arcs
0
10
20
30
40
50
60
70
80
40
45
50
55
60


MFA runtime vs The number of arcs
(n=20, U=10, m_min=20, m_max=60, k=15)
SAP
CS
Preflow
Runtime in milliseconds
m - the number of arcs
0
20
40
60
80
100
120
140
40
45
50
55
60

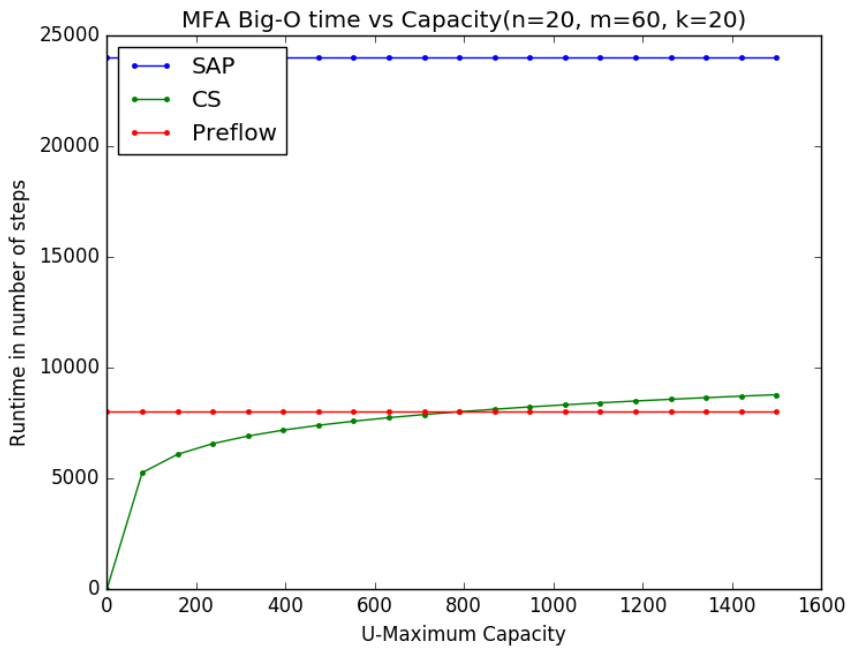

MFA runtime vs The number of arcs
(n=20, U=10, m_min=20, m_max=60, k=20)
SAP
CS
Preflow
Runtime in milliseconds
m - the number of arcs
0
20
40
60
80
100
120
40
45
50
55
60

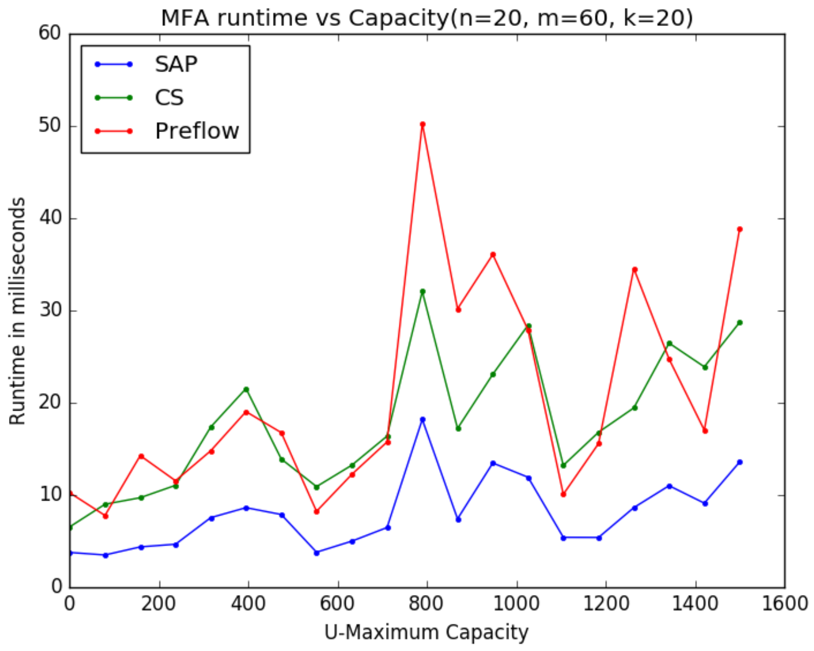

MFA runtime vs The number of arcs
(n=20, U=10, m_min=20, m_max=60, k=20)
SAP
CS
Preflow
Runtime in milliseconds
m - the number of arcs
0
10
20
30
40
50
60
70
80
90
40
45
50
55
60

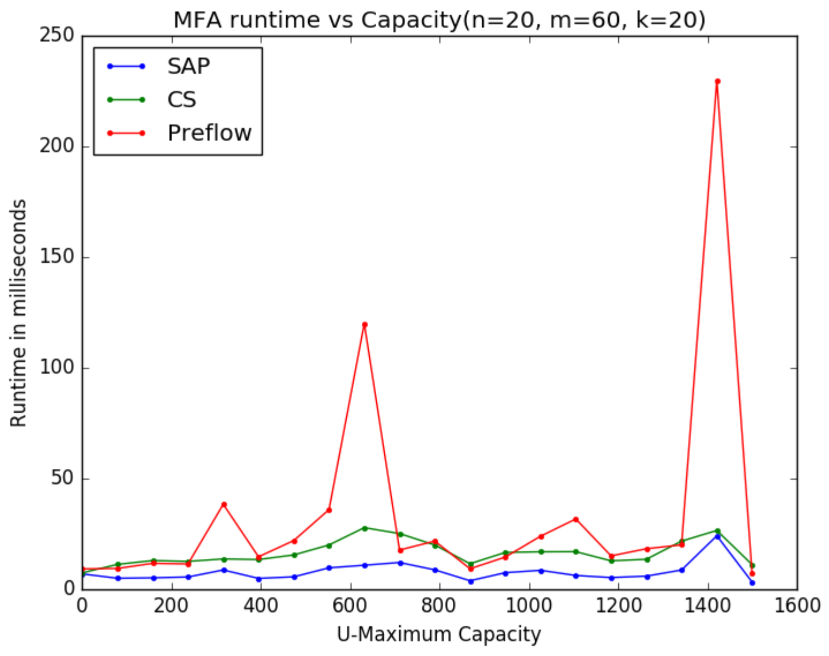


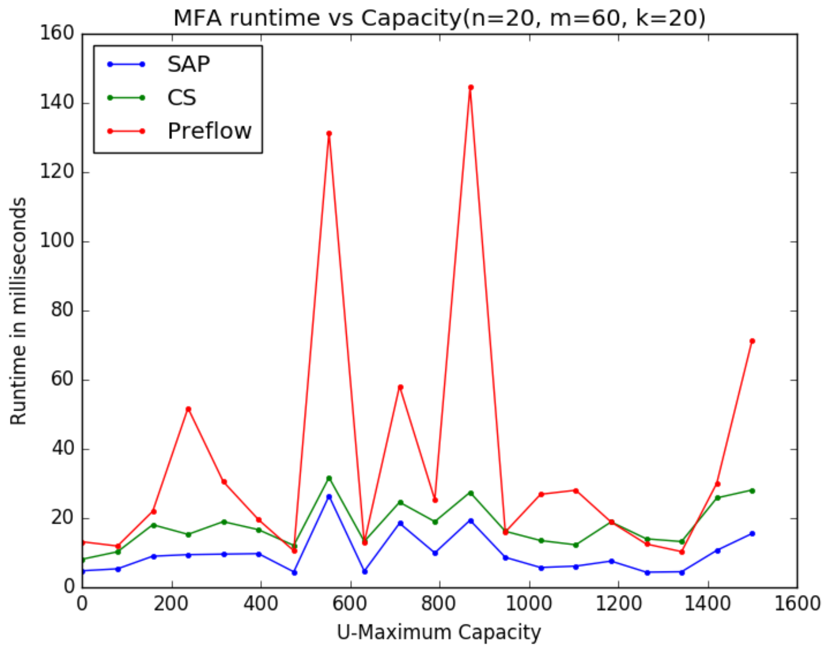
4 Runtime Analysis in terms of Maximum Capacity

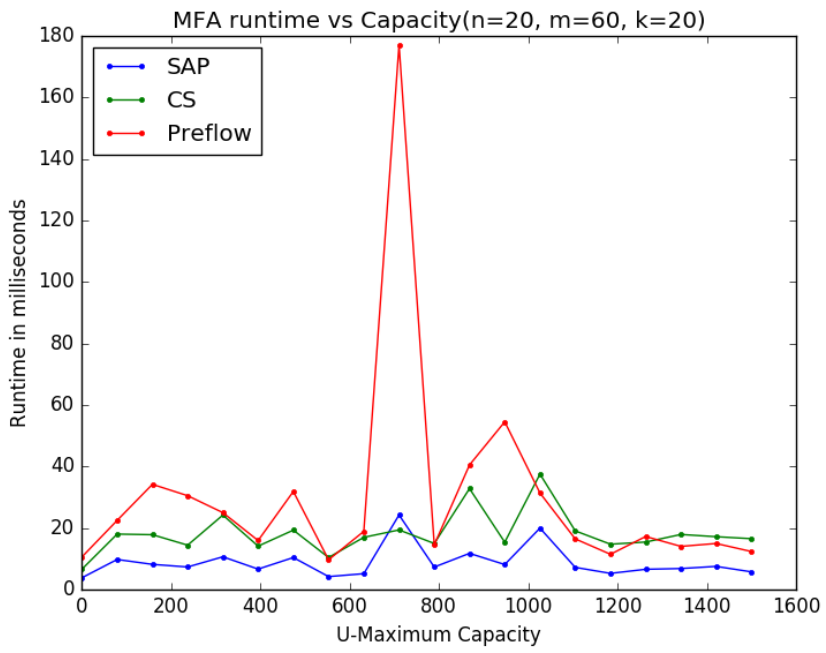
Average runtime in practice:
The best: SAP-MaxFlow algorithm
The second best: Capacity scaling MaxFlow algorithm
The slowest: The FIFOPreflow-MaxFlow algorithm. Though sometimes it performs the same as the previous two.
Please see the Big O times of the algorithms and the typical runtime patterns observed from the experiment in the following figures where k is the number of random graph. Each figure is for a different set of random graphs.
The experiment settings:
We fix the number of nodes and the number of arcs. Then we vary the maximum capacities of the random graphs.. For each maximum capacity, the random graph is different, but each random graph has the same n,m.


MFA Big-O time vs Capacity(n=20, m=60, k=20)
SAP
CS
Preflow
Runtime in number of steps
U-Maximum Capacity
0
5000
10000
15000
20000
25000
0
200
400
600
800
1000
1200
1400
1600


MFA runtime vs Capacity(n=20, m=60, k=20)
SAP
CS
Preflow
Runtime in milliseconds
U-Maximum Capacity
0
10
20
30
40
50
60
0
200
400
600
800
1000
1200
1400
1600


MFA runtime vs Capacity(n=20, m=60, k=20)
SAP
CS
Preflow
Runtime in milliseconds
U-Maximum Capacity
0
50
100
150
200
250
0
200
400
600
800
1000
1200
1400
1600


MFA runtime vs Capacity(n=20, m=60, k=20)
SAP
CS
Preflow
Runtime in milliseconds
U-Maximum Capacity
0
20
40
60
80
100
120
140
160
0
200
400
600
800
1000
1200
1400
1600



5 Summary

The average runtime in practice is different than the worst-case time, Big-O, depending on the experiment setting. In practice, SAP-MaxFlow algorithm performs the best although it has the largest Big-O time. The capacity Scaling Algorithm is in the second place with a little slower runtime than SAP-MaxFlow algorithm. In practice, the Preflow push algorithm has the slowest average runtime with high deviation depending on the input graph while its Big-O time is usually above Big-O time of Capacity scaling and below the Big-O time of SAP-MaxFlow algorithm.

6 How to run the codes for the experiments

```
>python NAnalysis.py
>python MAnalysis.py
>python UAnalysis.py
```

If the code hangs, click Ctrl + C to interrupt and run again. Sometimes, FIFOPreflow algorithm hangs because the input random graph doesn't have enough connectivity to pull back the preflow. I need to debug it more and figure it out.