Note:

We have placed all our source in the folder 'Group_D_Final_code'.

The input graphs (Bipartite, Mesh, Fixed Degree and Random) that are generated for our testing are placed in the folder 'Group_D_Input_Graphs'.

1.Ford Fulkerson Algorithm

The ford Fulkerson algorithm computes the maximum flow in the flow network. The code implemented follows the recipe which is explained here. To compute the maximum flow in a flow network using Ford Fulkerson algorithm, we need a residual graph G_f. Residual Graph of a flow network is a graph which indicates additional possible flow. If there is a path from source to sink in residual graph, then it is possible to add flow. Every edge of a residual graph has a value called residual capacity which is equal to original capacity of the edge minus current flow. Residual capacity is basically the current capacity of the edge. Residual capacity is 0 if there is no edge between two vertices of residual graph. We can initialize the residual graph as original graph as there is no initial flow and initially residual capacity is equal to original capacity. To find an augmenting path, we can either do a BFS or DFS of the residual graph. Here in the implantation of Ford Fulkerson, we used BFS. Then we have to find the minimum capacity of the path which is called as bottleneck. and increase the flow by bottleneck in the original graph for each augmenting path, update the original graph and residual graph as for forward edge of the augmenting path in original graph, increase the flow by bottleneck. For backward edge of the augmenting path in residual graph, decrease the flow by bottleneck. We have to repeat the procedure, till there is no augmenting path. When we reached the stage that there is no augmenting path in the residual graph then maximum flow is reported in the flow network.

Structure:

Following are the files used for implementing the algorithm:

1.FordFulkersonAlgorithm – Main class for the ford Fulkerson algorithm. It returns the maximum flow. This method takes in a graph as input and returns the max flow of the input graph using F-F algorithm. This method also measures the time taken in milliseconds to compute the max flow

2.computeBottleneck-This method computes the bottleneck capacity available in the augmenting path. This is additional amount of flow that can be augmented to the
network flow

3.addFlowToAugmentPath-This methods adds the additional amount of flow through the network and updates the available capacity of each edge along the augmenting path

4.FindForwardEdge-This method finds the forward edge of the residual graph for a given backward edge.

5.computeEdgeData-This method is called to initialize the edge data of the graph with edge capacity and edge flow

6.createResidualGraph-Creates the residual graph based on the flow network

7.hasPathFromSourceToSink-This method perform BFS to find the augmenting path in the residual graph

Other class used by the algorithm:

Vertex.java, Edge.java, EdgeData.java, InputEdge.java and SimpleGraph.java are used without modifications from the Graph code.

InputEdge.java class – it contains the name of firstVertex, secondVertex and capacity of the edge as given on each line of the input file. The input file is converted to ArrayList of InputEdge and pass to the algorithm. The algorithm then constructs and initializes the graph using internal datastructures.

Other datastructures used: Queue, LinkedList, HashMap and ArrayList

Description of Routines:

1.FordFulkersonAlgorithm

Constructor:

1. Public double calculateMaxFlow(ArrayList<InputEdge> inputEdges

Takes in the ArrayList of inputEdges and constructs the SimpleGraph and initializes the edge and vertex data.

2.SimpleGraph ResidualGraph =create ResidualGraph(G)

This method computes the initial residual graph

3.double bottleneck = computeBottleneck(residualGraph ,source, sink, nodetoparent)

This method computes the bottleneck

4.double maxflow=ffalgm.CalculatemaxFlow(G)

Calculate the max flow in the graph

2.Scaling Ford Fulkerson:

1.ScalingFordFulkerson – Main class for the algorithm. It contains the core code for the algorithm.

Private Routines:

1.Function to determine if there is an augmenting s-t path in the residual graph -

private static boolean bfs (HashMap<String, LinkedList<Edge>> adjGraph, HashMap<String, Edge> child_parent_map, String s, String t, double delta) throws Exception

2.To calculate the max capacity flowing out of source

private static double maxCapacityFromSource (HashMap<String, LinkedList<Edge>> graph, String s) throws Exception

3.To calculate the scaling factor delta where delta is the maximum power of 2 less than equal to capacity.

private static double calculateDelta (double n)

4.To print adjacency list graph's vertices and edges**.**
private static void printGraph (HashMap<String, LinkedList<Edge>> graph)

5.Function to build an adjacency list graph based on vertices and edges provided by Simple Graph
private static HashMap<String, LinkedList<Edge>> buildGraph (SimpleGraph G, Boolean Isresidual)

Public Routines:
1.The following ScalingFordFulkersonDriver function is to build SimpleGraph based on input file and executing scalingFordFulkerson method
public static double scalingFordFulkersonDriver (String filePath) throws Exception

3.PreFlow-Push Algorithm:

1.Structure
Following are the files used for implementing the algorithm:

1.PushRelabelAlgorithm – Main class for the algorithm. It contains the core code for the algorithm.
2.PushRelabel.PRVertexData– Class which contains height and excessFlow for each vertex.
3.PushRelabel.PREdgeType– An enum to denote whether the edge is contained in the Graph or Residual Graph.
4.PushRelabel.PREdgeData– Contains the capacity, flow and the EdgeType of the edge.
Other class used by the algorithm:
5.Vertex.java, Edge.java and SimpleGraph.java are used without modifications from the Graph code.
InputEdge.java class – it contains the name of firstVertex, secondVertex and capacity of the edge as given on each line of the input file. The input file is converted to ArrayList of InputEdge and pass to the algorithm. The algorithm then constructs and initializes the graph using internal datastructures.
Other datastructures used: Queue, LinkedList, HashMap and ArrayList

2.Description of Routines:

1.PushRelabelAlgorithm:
Constructor:
public PushRelabelAlgorithm(ArrayList<InputEdge> inputEdges)

Takes in the ArrayList of inputEdges and constructs the SimpleGraph and initializes the edge and vertex data.

2.Public routines:

public int getMaxFlow()

If the maxflow for the graph was already calculated for the graph, it returns the maxflow that was already calculated.

Otherwise, it initializes the graph before running the Push-Relabel algorithm. Then it runs the algorithm, calculates the maxflow and saves the result in instance variable.

3.Private routines:

private void initialize()

Initializes the vertex and edge data, and makes saturation push for all out edges from source.

It also adds the vertices with excessFlow to the relabelVertexQueue.

4.private void calculateMaxFlow()

This routine actually runs the algorithm calculates the maxflow and saves it in an instance variable.

It runs a while loop till there are vertices in relabelVertexQueue.

The loop dequeues the first vertex which has excessFlow form the relabelVertexQueue and executes push operations if possible. If the vertex has excessFlow even after executing all possible push operations, it is relabeled and added back to the queue.

5.private boolean tryToPushForExcessFlow(Vertex vertexWithOverflow)

Checks if push operation is applicable and pushes excessFlow to connected vertices. Returns true if relabel operation is applicable and false otherwise.

6.private void push(Edge edge)

Executes the push operation on the edge.

7.private void relabel(Vertex vertex)

Executes the relabel operation on the vertex.

8.private void constructGraph(ArrayList<InputEdge> inputEdges)

this routine is used by the constructor to create a SimpleGraph from the List of inputEdges.

Output for the input testing files and the results plotted as Graph:

1.Bipartite Graph:

Test 1: Varying nodes Vs average running time

		Bipartite Graph - Varying Nodes Vs Average Running Time (with constant capacity min_capacity = 50 and max_capacity = 1500 and Probability = 0.5)			
		Average Running Time (milliseconds)			max_flow
m	n	Ford Fulkerson	Scaling Ford Fulkerson	Preflow Push	
20	20	16	4	1	15188
50	50	68	18	29	35639
100	100	243	47	7	76034
150	150	706	100	15	110331
200	200	1737	202	1878	149663

Table 1: Varying nodes vs average running time(ms)

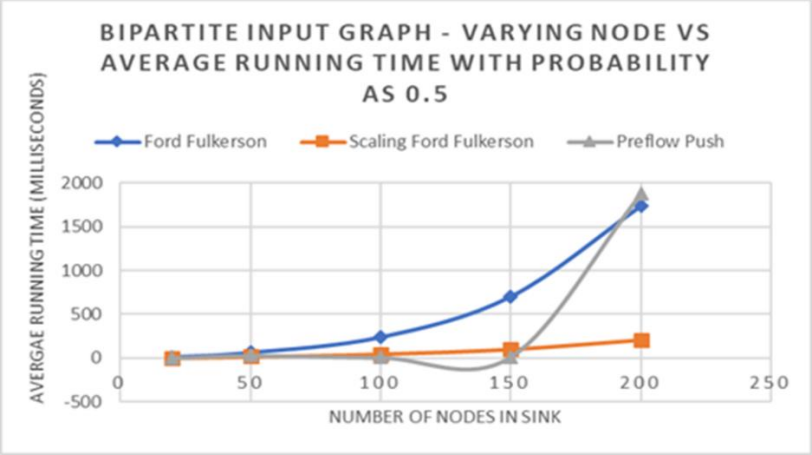

Figure 1: Bipartite Input Graph-Varying Nodes vs average running time(ms)

Test 2: Capacities ranges Vs Running time

Varying capacities with min capacity ranging from 0 to 500 and max capacity from 250 to 2000 with probability 0.5					
min capacity	max capacity	Average Running Time in milliseconds			max flow
		Ford Fulkerson	Scaling Ford Fulkerson	Preflow-push	
0	250	109	34	5	7205
50	250	88	37	7	8782
100	250	81	34	5	9294
150	250	70	35	58	12148
200	250	79	34	54	13250
500	1000	73	37	54	45847
500	2000	102	53	72	78512

Table 2: Varying Capacities ranges vs average running time(ms)

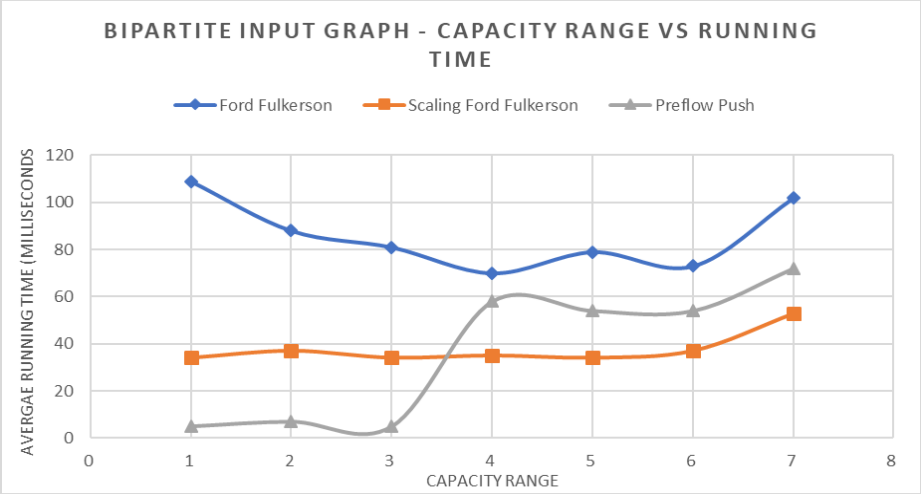

Figure 2: Bipartite Input Graph- Varying Capacities ranges vs average running time(ms)

Test 3: Varying Nodes Vs Average Running Time:

	Bipartite Graph - Varying Nodes Vs Average Running Time (with constant capacity min_capacity = 100 and max_capacity = 500 and Probability = 1.0)				
		Time (milli seconds)			max_flow
m	n	Ford Fulkerson	Scaling Ford Fulkerson	Preflow Push	
20	20	13	5	8	4992
50	50	73	21	41	14991
100	100	294	53	481	30360
150	150	946	139	60	42653
200	200	2797	313	5430	59093

Table 3: Varying Nodes Vs Average Running Time(ms)

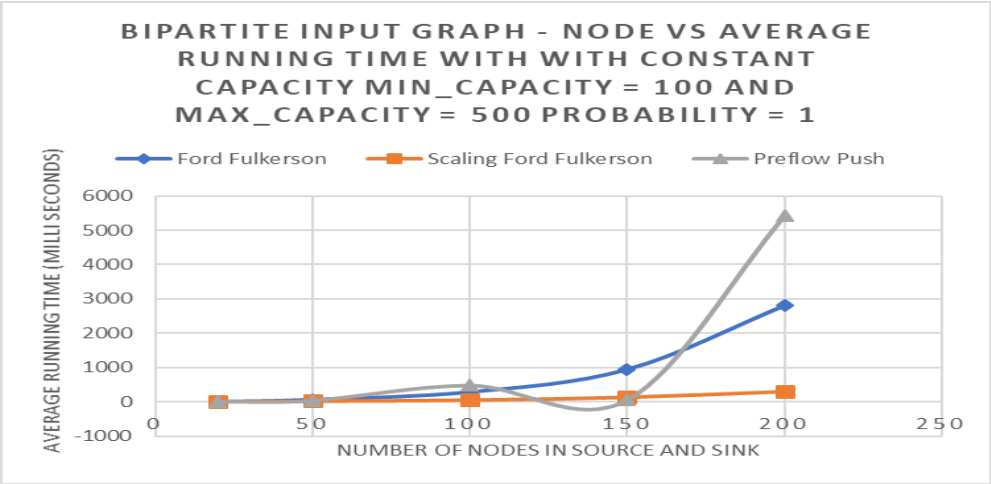

Figure 3: Varying Nodes Vs Average Running Time(ms)

2.Random Input graph:

Test 1: Varying min capacity Vs Average running time

Random Graph with varying the capacity range with other parameter are constant				
	constant (vertics=30,dense=30,maxcapacity=250)			
MinCapacity	Running Time(ms)			MaxFlow
	Ford Fulkerson	Preflow push	caling Ford Fulkerson	
50	6	3	3	659
100	9	1	4	1082
150	14	14	9	1531
200	11	9	7	1428
250	5	4	2	1750

Table 4: Varying min capacity Vs Average running time(ms)

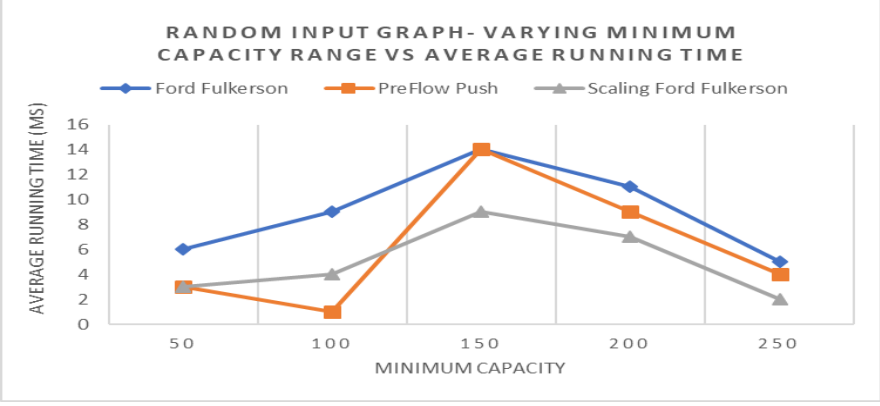

Figure 4: Random Input Graph-Varying min capacity Vs Average running time(ms)

Test 2: Varying dense range Vs Running time

Random Graph with varying the dense range with other parameter are constant				
constant (vertices=100,maxcapacity=500,MinCapacity=250)				
Dense Range	Average Running Time(ms)			MaxFlow
	Ford Fulkerson	Preflow push	Scaling Ford Fulkerson	
50	135	72	32	16210
100	925	7	109	37336
150	935	1	118	36623
200	943	138	117	37804
250	876	165	117	36172

Table 5: Varying dense range Vs average running time(ms)

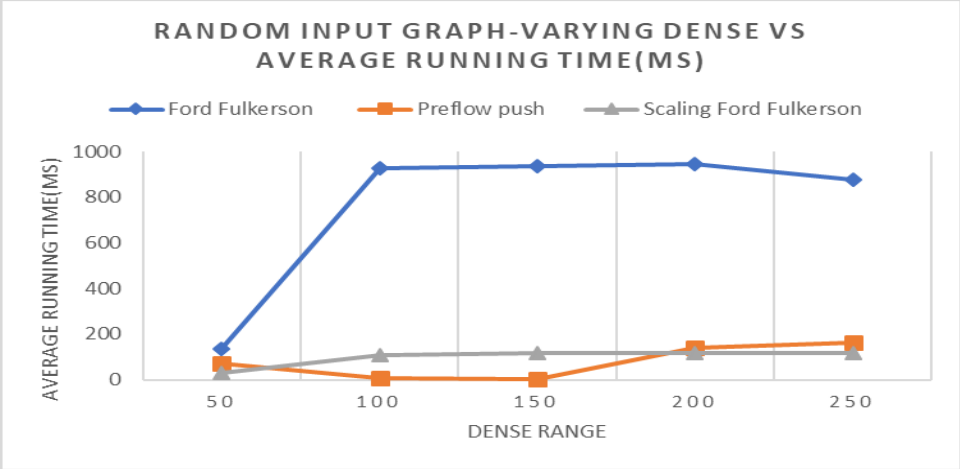

Figure 5: Random Input Graph- Varying dense range Vs average running time(ms)

Test 3: Varying the Vertices Vs Running Time

Random Graph with varying the Vertices range with other parameter are constant				
constant (dense=100,mincapacity=250,MaxCapacity=500)				
Vertices Range	Average Running Time(ms)			MaxFlow
	Ford Fulkerson	Preflow push	Scaling Ford Fulkerson	
50	46	15	12	17895
100	884	1	108	36470
150	4174	1015	426	54782
200	9767	3017	1109	73445
250	20720	6805	3057	93373

Table 6: Varying Vertices range vs average running time(ms)

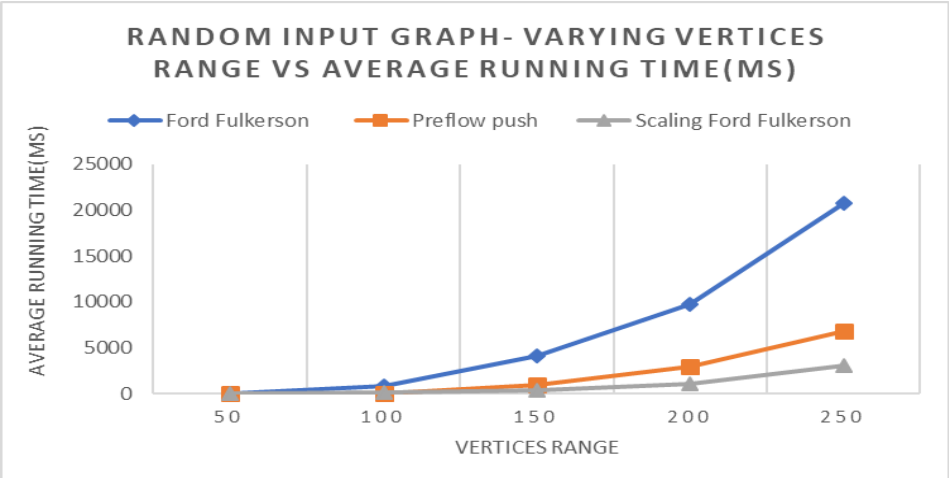

Figure 6: Varying vertices range Vs average running time(ms)

Test 4: Varying the Max capacity Vs Running Time

This is the addition testing which is not in the report. The Graph represent the Graph for varying maximum capacity range vs average running time. Graph 7 contains the data for the various tests that we have done by giving random graph as input. In this testing, we consider other parameter as constant throughout the process and keep varying the max capacity from 50 to 250 at a step size of 50. The constant parameters are (dense=100, vertices=100 and min capacity=50). For random input graph with varying max capacity, scaling ford Fulkerson perform and Preflow push perform better than ford Fulkerson algorithm

Random Graph with Varying Maxcapacity range with other parameter				
are constant (vertices=100,dense=100,mincapacity=50)				
Max Capacity	Running Time(ms)			MaxFlow
	Ford Fulkerson	Preflow Push	Scaling Ford Fulkerson	
50	459	1	23	4900
100	867	153	89	7490
150	839	1	101	9831
200	851	6	87	12369
250	830	175	96	14332

Table 7: Varying max capacity Vs average running time(ms)

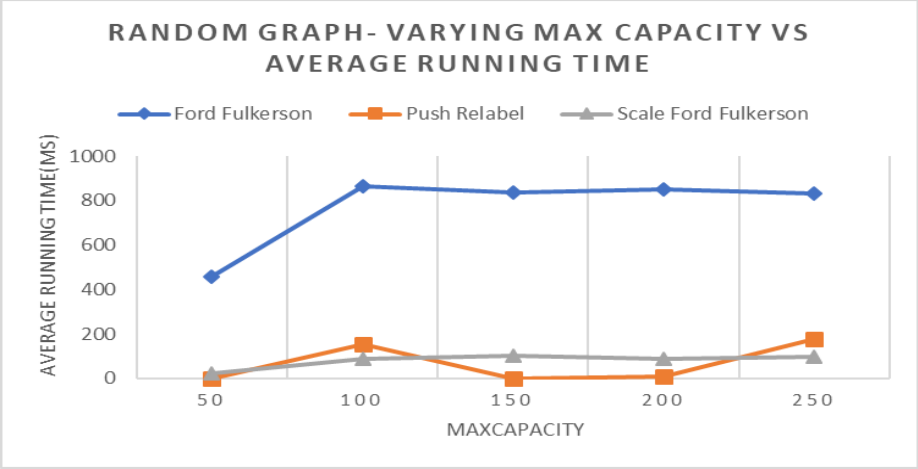

Figure 7: Varying max capacity vs average running time(ms)

3.Fixed Degree Graph:

Test 1: Varying the capacity ranges Vs Average running time(ms)

FixedDegree Graph - Vary the capacity ranges on edges (with constant vertices v = 30 and edges e = 25)						
input_file_name	min_capacity	max_capacity	Avaerage Running Time (milli seconds)			max_flow
			Ford Fulkerson	Scaling Ford Fulkerson	Preflow Push	
FixedDegree_input1.txt	0	50	28	11	2	527
FixedDegree_input2.txt	0	100	30	13	2	1210
FixedDegree_input3.txt	0	250	26	15	14	2695
FixedDegree_input4.txt	0	500	25	14	2	5334
FixedDegree_input5.txt	0	1000	42	16	2	12052
FixedDegree_input6a.txt	0	1500	25	17	14	16621
FixedDegree_input7a.txt	0	2000	24	16	15	20993

Table 8: Fixed Degree Graph as Input Graph: Varying the range of capacities on edges keeping the number of vertices and number of edges as constant

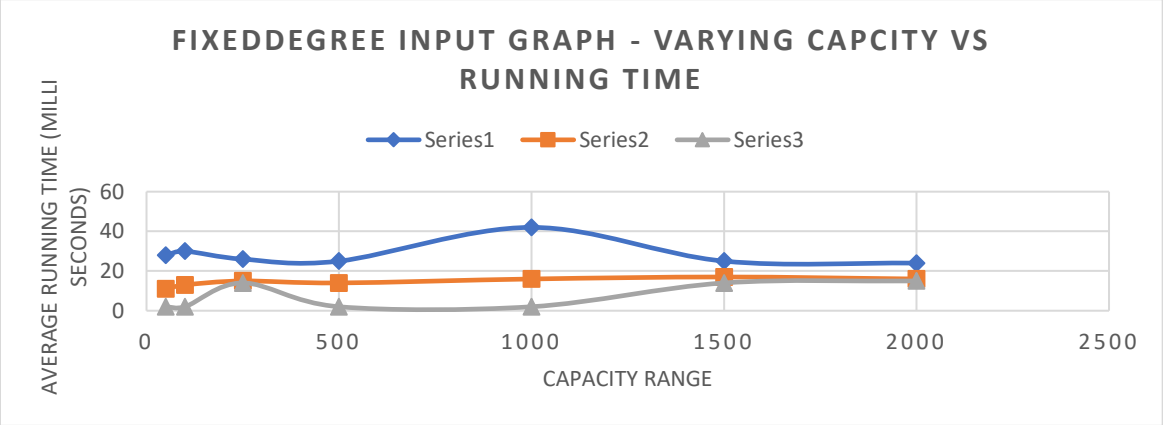

Figure8: Fixed Degree Graph as Input Graph: The capacity range on X-axis and average running time in Y-axis keeping the number of vertices and number of edges as constant (v=30 and e=25)

Test 2: Varying the vertices and edges vs average running time(ms)

FixedDegree Graph - Vary the vertices and edges (with constant capacity min_capacity = 10 and max_capacity = 100)						
input_file_name	vertices	edges	Time (milli seconds)			max_flow
			Ford Fulkerson	Scaling Ford Fulkerson	Preflow Push	
FixedDegree_input6.txt	10	9	3	1	1	353
FixedDegree_input7.txt	50	49	47	21	4	2576
FixedDegree_input8.txt	75	74	119	43	120	3770
FixedDegree_input9.txt	100	99	264	81	399	4839
FixedDegree_input10.txt	125	124	536	173	30	6212
FixedDegree_input13.txt	150	149	1061	308	1337	7768
FixedDegree_input14.txt	200	199	2858	880	35	11122
FixedDegree_input15.txt	250	249	6024	2389	7623	13310

Table9: Fixed Degree Graph as Input Graph: Varying the number of vertices and edges keeping the range of capacities as constant

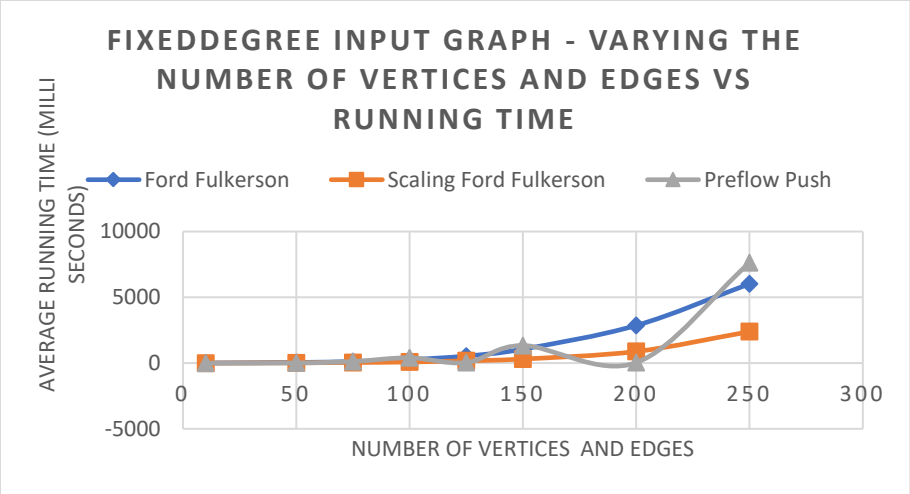

Figure 9: Fixed Degree Graph as Input Graph: Number of vertices/Edges on X-axis and Average running time on Y-axis keeping the capacities as constant with min_capacity = 10 and max_capacity = 100

Test 3: Vary the Vertices Vs average running time(ms)

FixedDegree Graph - Vary only vertices (with constant capacity min_capacity = 0 and max_capacity = 250 and edges = 50)					
		Time (milli seconds)			max_flow
input_file_name	Vertices	Ford Fulkerson	Scaling Ford Ful	Preflow Push	
FixedDegree_input16.txt	60	59	25	4	5877
FixedDegree_input17.txt	100	89	34	124	6315
FixedDegree_input18.txt	150	165	47	10	6244
FixedDegree_input19.txt	200	190	58	732	5488
FixedDegree_input20.txt	250	259	105	1203	5965

Table 10: Fixed Degree Graph as Input Graph: Varying the number of vertices keeping the range of capacities and number of rows as constant

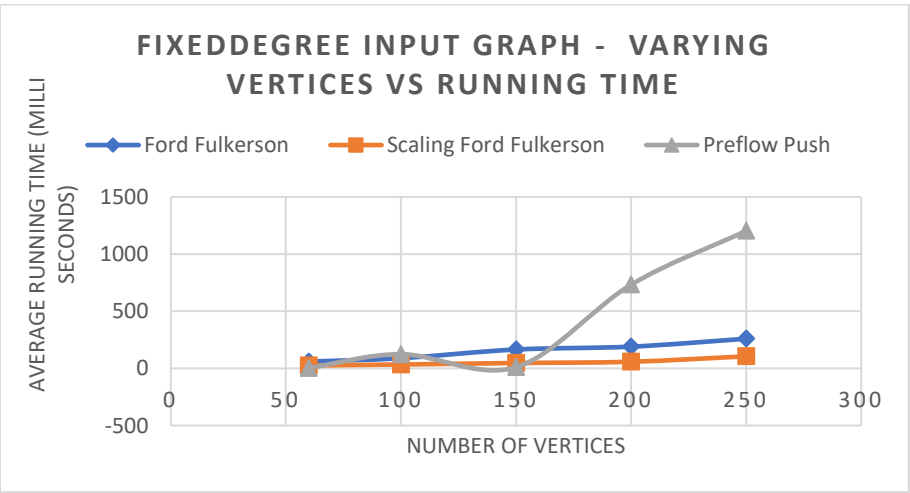

Figure 10: Fixed Degree Graph as Input Graph: The number of vertices on X-axis and average running time on Y-axis with constant range of capacities (min_capacity = 0, max_capacity =250) and number of edges as constant (e=50)

Test 4: Vary edges vs average running time(ms)

FixedDegree Graph - Vary only edges (with constant capacity min_capacity = 0 and max_capacity = 250 and vertices = 250)					
		Time (milli seconds)			max_flow
input_file_name	edges	Ford Fulkerson	Scaling Ford Ful	Preflow Push	
FixedDegree_input20.txt	50	259	105	27	5965
FixedDegree_input21.txt	90	774	221	4136	11068
FixedDegree_input22.txt	140	1928	676	6397	16305
FixedDegree_input23.txt	190	3535	1334	4100	22585
FixedDegree_input24.txt	240	6162	2654	4136	29401

Table 11: Fixed Degree Graph as Input Graph: Varying the number of edges keeping the range of capacities and number of vertices as constant

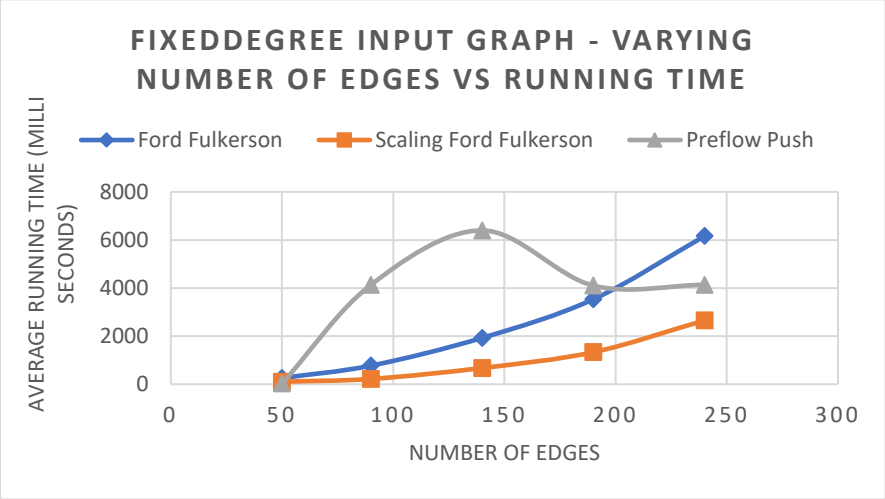

Figure 11: Fixed Degree Graph as Input Graph: The number of edges on X-axis and average running time on Y-axis with constant range of capacities (min_capacity = 0, max_capacity =250) and number of vertices as constant (v=250)

4.Mesh Input Graph:

Test 1: Vary the capacity range on edges vs average running time(ms)

Mesh Graph - Vary the capacity range on edges (other parameters constant - rows(m) = 30 and columns n=35)					
		Time (milli seconds)			max_flow
input_file_name	capcity	Ford Fulkerson	Scaling Ford Fulkerson	Preflow Push	
mesh_input1.txt	50	38	28	2	1500
mesh_input2.txt	100	52	24	1	3000
mesh_input3.txt	200	45	30	2	6000
mesh_input4.txt	300	41	31	2	9000
mesh_input5.txt	500	40	23	1	15000
mesh_input6.txt	1000	41	23	2	30000

Table 12: Mesh Input Graph – Varying the range of capacities by keeping the number of rows and columns as constant

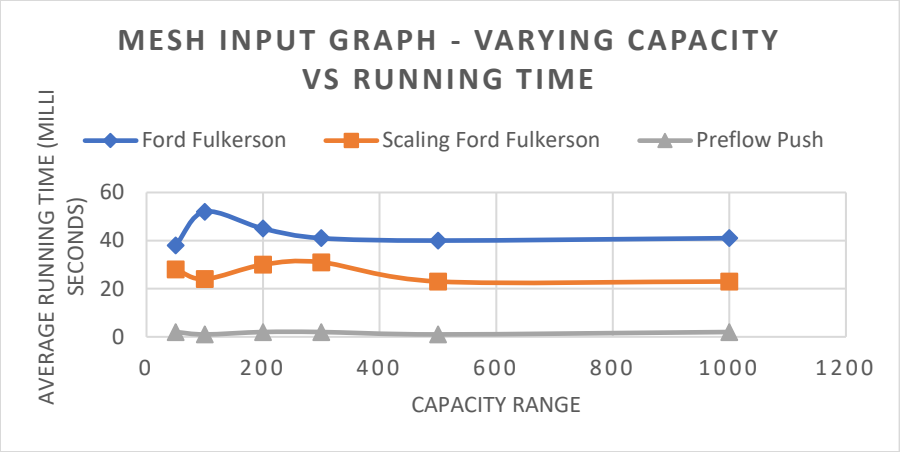

Figure 12: Mesh Input Graph – Capacity range on X-axis and Average running time on Y-axis with constant value for rows = 30 and columns = 35

Test 2: Vary the row and column vs average running time(ms)

Mesh Graph - Vary rows and columns (Capacity as constant = 100)					
		Time (milli seconds)			max_flow
input_file_name	m/n	Ford Fulkerson	Scaling Ford Fulkerson	Preflow Push	
mesh_input7.txt	10	6	3	3	1000
mesh_input8.txt	50	124	64	3	5000
mesh_input9.txt	75	371	220	6	7500
mesh_input10.txt	100	922	630	7	10000
mesh_input11.txt	125	1944	1399	9	12500
mesh_input15.txt	150	3941	2495	20	15000
mesh_input16.txt	200	10156	6336	31	20000

Table 13: Mesh Input Graph – Varying the number of rows and columns by keeping the capacity range as constant

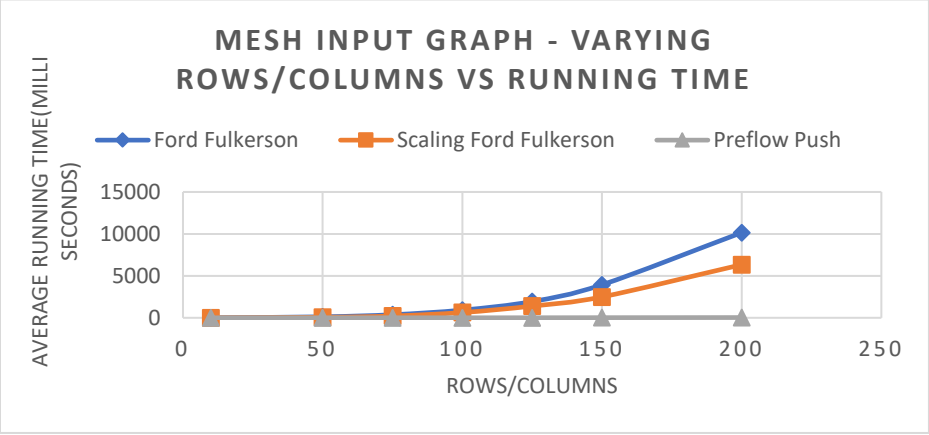

Figure 13: Mesh Input Graph – Number of rows/columns on X-axis and Average running time on Y-axis with constant value for capacity = 100

Test 3: Vary the columns vs average running time(ms)

Mesh Graph - Vary columns (Constant parameters, capacity = 50 and rows (m) = 50)					
		Time (milli seconds)			max_flow
input_file_name	n	Ford Fulkerson	Scaling Ford Fulkerson	Preflow Push	
mesh_input17.txt	10	39	22	1	2500
mesh_input18.txt	50	112	63	3	2500
mesh_input19.txt	100	231	140	6	2500
mesh_input20.txt	150	417	256	7	2500
mesh_input21.txt	200	558	333	19	2500

Table 14: Mesh Input Graph – Varying the number of columns by keeping the capacity range and number of rows as constant

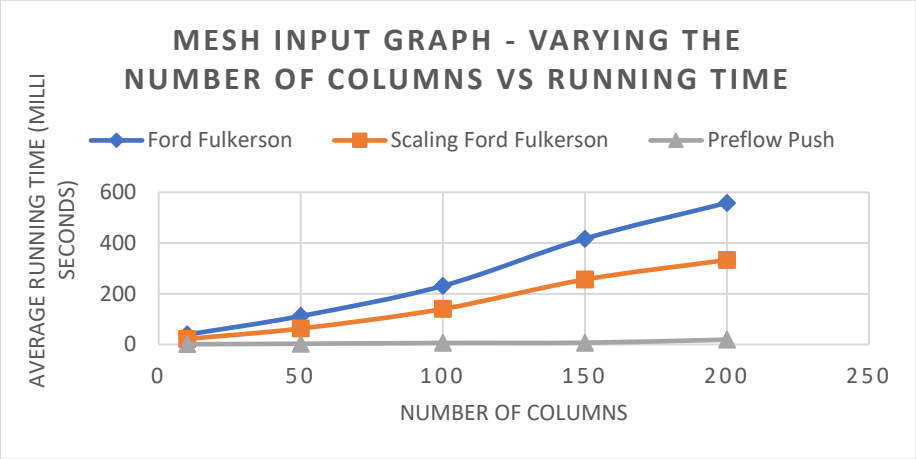

Figure 14: Mesh Input Graph – Number of columns on X-axis and Average running time on Y-axis with constant value for capacity = 50 and number of rows = 50