DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY
COMP 6651 Algorithm Design Techniques Fall 2023
Project Report
Shalvi Saxena (40220846)

---

1 PROBLEM DESCRIPTION

The project aims to apply the Ford-Fulkerson algorithm to source-sink networks by creating and contrasting four augmenting route methods. Examining these algorithms' performance on arbitrary source-sink graphs is the goal. The project involves developing random source-sink networks, putting the Ford-Fulkerson algorithm into practice, running simulations, and using versions of Dijkstra's algorithm for finding augmenting paths.

2 IMPLEMENTATION DETAILS

2.1 Random Source-Sink Graph Generator

- The generator creates a source-sink graph with n nodes.
- Each node is assigned random (x, y) coordinates between 0 and 1.
- Edges are randomly added between nodes within a distance r.
- The capacity of each edge is assigned a random integer value in the range [1..upperCap].
- The graph is stored in an ASCII readable format adjacency list in a text file.

Pseudocode for the generator:

```
GENERATE_SINK_SOURCE_GRAPH(n, r, upperCap):
    Create a set of vertices V such that |V| = n
    For each vertex u in V:
        u.x ← a uniform random number in [0..1]
        u.y ← a uniform random number in [0..1]
    For each vertex u in V:
        For each vertex v in V:
            If (u  v) && ((u.x - v.x)^2 + (u.y - v.y)^2  r^2):
                rand ← a uniform random number in [0..1]
                If rand < 0.5:
                    If (u, v)  E and (v, u)  E:
                        E ← E  {(u, v)}
                Else:
                    If (u, v)  E and (v, u)  E:
                        E ← E  {(v, u)}
```

```
For each edge (u, v) in E:
    (u, v).cap ← a uniform random number in [1..upperCap]
Randomly select a node as the source s
Apply breadth-first search from s to find a longest acyclic path
and set t as the end node
Store the graph and edge capacities in a readable format
```

2.2 Augmenting Path Algorithms

Implement Ford-Fulkerson and three augmenting path algorithms:

- **Shortest Augmenting Path (SAP):** Run Dijkstra's algorithm treating edge capacities as unit capacities therefore, the key in the priority queue will be 0 for all the vertices.
- **DFS-like:** Use a decreasing counter as the key value when inserting a node into the priority queue therefore, the key in the priority queue will be (-len(path)) for all the vertices making it suitable for min heap.
- **Maximum Capacity (MaxCap):** Modify Dijkstra's algorithm to find the augmenting path with the maximum capacity therefore, the key in the priority queue will be (- capacity of the edge) for all the vertices.
- **Random:** Use a random value as the key value when inserting a node into the priority queue therefore, the key in the priority queue will be any random value for all the vertices.

3 IMPLEMENTATION CORRECTNESS

A thorough testing procedure has been used to confirm the accuracy of the Ford-Fulkerson algorithm's augmenting route variants and implementation. The correctness of the augmenting pathways produced by each approach was confirmed manually by cross-validation of the findings and by using sample graphs. The phases in the validation process are as follows:

3.1 Manual Verification on small graphs

- The augmenting pathways were identified by meticulous examination after a set of small graphs were manually constructed to verify the accuracy of the augmenting path techniques.
- The pathways produced by the relevant algorithm were compared with the manually recognised augmenting paths for every approach.
- Confirming that the algorithms successfully identified enhancing pathways in different contexts was made possible through the process of human verification.

3.2 Cross-Examination of the Findings

- Cross-verification was done between the outcomes of the implemented algorithms and the manually determined augmenting pathways on small networks.
- The number of pathways, their lengths, and the particular edges that are part of each path were all compared.
- The accuracy of the implementations was confirmed by the consistent matching of the augmenting path approaches' findings with the manually validated pathways.
- I logged the max flow in the table to verify the maximum flow detected by all the algorithms.

3.3 Examining using exemplar graphs

- The solutions were evaluated on a series of example graphs that represented various contexts in addition to small personally constructed graphs.
- The online code functioned as a point of reference, and the anticipated outputs were verified with the outcomes of the algorithms applied to the example graphs [2] [1].

4 RESULTS

4.1 Simulation and Results

I conducted simulations on various graph configurations and recorded the following metrics:

- **Paths:** The number of augmenting paths required until Ford-Fulkerson completes.
- **Mean Length (ML):** The average length (number of edges) of the augmenting paths.
- **Mean Proportional Length (MPL):** The average length of the augmenting path as a fraction of the longest acyclic path from s to t.
- **Total Edges:** The total number of edges in the graph.
- **Flow:** The flow achieved by the Ford-Fulkerson algorithm.

Due to the limitations of computational resources I had to skip the graphs with a higher number of edges. The simulations conducted with the required graph configurations and results are recorded in Table [1].

I selected four graphs for comparison based on the number of edges, as illustrated in the figure [1]. Notably, the DFS algorithm exhibits the highest mean length, indicating its relatively lower efficiency in terms of path length. The mean proportional length (MPL) represents the average length of the augmenting paths as a fraction of the longest acyclic path from the source (s) to the sink (t). Despite DFS detecting longer augmenting paths, its MPL is not high.

Table 1: Results of Augmenting Path Algorithms

Algorithm	n	r	upperCap	Paths	ML	MPL	total Edges	Flow
SAP	100	0.2	2	3	20.3333	0.924242	477	3
DFS	100	0.2	2	3	45	0.833333	477	3
MaxCap	100	0.2	2	4	18.75	0.585938	477	3
Random	100	0.2	2	3	17.6667	0.929825	477	3
SAP	200	0.2	2	2	20.5	0.97619	2067	2
DFS	200	0.2	2	2	120.5	0.919847	2067	2
MaxCap	200	0.2	2	3	13.6667	0.621212	2067	2
Random	200	0.2	2	2	22	0.733333	2067	2
SAP	100	0.3	2	10	13	0.619048	1031	10
DFS	100	0.3	2	10	27.2	0.523077	1031	10
MaxCap	100	0.3	2	8	13.75	0.528846	1031	10
Random	100	0.3	2	10	12.2	0.677778	1031	10
SAP	200	0.3	2	9	17.2222	0.717593	4118	9
DFS	200	0.3	2	9	107.333	0.730159	4118	9
MaxCap	200	0.3	2	7	10.4286	0.579365	4118	9
Random	200	0.3	2	9	13.8889	0.434028	4118	9
SAP	100	0.2	50	33	22.9697	0.588967	556	73
DFS	100	0.2	50	57	40.3158	0.610845	556	73
MaxCap	100	0.2	50	8	17.75	0.71	556	73
Random	100	0.2	50	43	19.7442	0.54845	556	73
SAP	200	0.2	50	66	20.9848	0.723615	2096	145
DFS	200	0.2	50	145	109.324	0.687573	2096	145
MaxCap	200	0.2	50	8	21.375	0.689516	2096	145
Random	200	0.2	50	100	20.15	0.671667	2096	145
SAP	100	0.3	50	97	14.6082	0.635141	1066	211
DFS	100	0.3	50	196	54.1531	0.608461	1066	211
MaxCap	100	0.3	50	9	12.3333	0.72549	1066	211
Random	100	0.3	50	102	14.8529	0.495098	1066	211

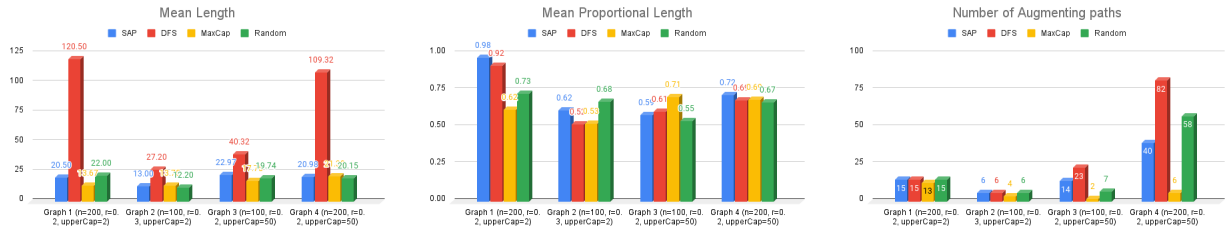

Figure 1: Graph Comparision Metrics

The 'number of paths' metric signifies the count of augmenting paths required until the Ford-Fulkerson algorithm completes for each algorithm on different graphs. This metric provides insights into how quickly the algorithm converges to the maximum flow. Examining the charts, the MaxCap algorithm stands out by taking the least number of paths. This behaviour can be attributed to MaxCap's strategy of detecting paths based on maximum flow, leading to fewer iterations and higher efficiency in achieving the maximum flow.

4.2 Choice of Experimental Parameters

In selecting the experimental parameters for our study, careful consideration has been given to obtaining a comprehensive understanding of the augmenting path algorithms' behaviour. The chosen parameters aim to shed light on algorithm performance under varying graph characteristics.

I am choosing to set the number of vertices (n) to 200 based on an analysis of the graphical representations. It is observed that key metrics such as the number of paths, mean length, mean proportional length, total edges, and flow exhibit consistent behaviour for different values of n. To scrutinize the algorithms under more challenging scenarios and to closely examine potential differences, a higher value of n=200 has been selected. Similar to the choice of n, the analysis of graphs indicates that varying the upper limit of edge capacities (upperCap) does not result in significant changes in the observed metrics. To thoroughly investigate the algorithms' responses to larger capacity networks, I decided to set upperCap to 30.

The parameter r, representing the density of the graph, emerges as a critical factor influencing algorithmic outcomes. Inconsistent results are observed for both r=0.2 and r=0.3 due to the significant difference in number of edges, indicating sensitivity to graph density. To explore the algorithms' behaviour under varying levels of network connectivity, both r=0.2 and r=0.25 values are selected.

5 CONCLUSION

Graph 1 comprises 200 nodes, the upperCap of 30 and the edge probability (r) of 0.2. While Graph 2 and Graph 1 are similar, Graph 2 has a greater edge probability (25%) that is r=0.25. The simulations conducted with the proposed graph configurations and results are recorded in Table [2].

Table 2: Results of Augmenting Path Algorithms with custom values

Algorithm	n	r	upperCap	Paths	ML	MPL	total Edges	Flow
SAP	200	0.2	30	14	18.8571	0.650246	1996	24
DFS	200	0.2	30	24	135.458	0.873925	1996	24
MaxCap	200	0.2	30	4	15.25	0.693182	1996	24
Random	200	0.2	30	11	20.6364	0.687879	1996	24
SAP	200	0.25	30	128	19.3672	0.537977	2996	188
DFS	200	0.25	30	184	102.723	0.567529	2996	188
MaxCap	200	0.25	30	22	17.9545	0.544077	2996	188
Random	200	0.25	30	145	19.3724	0.472498	2996	188

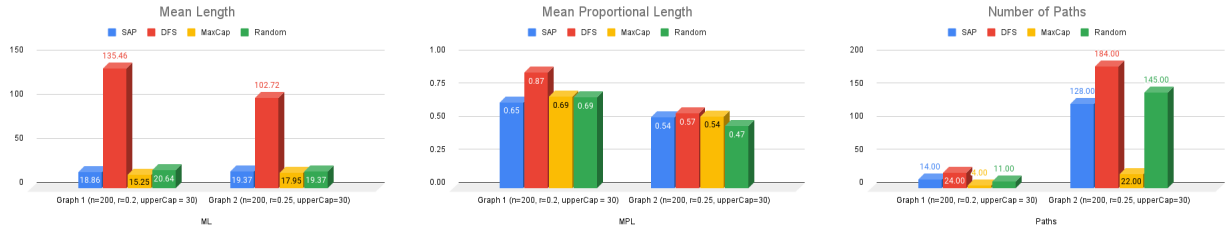

Figure 2: Graph Comparision Metrics for custom values

As stated in the figure [2], the average length of the augmenting paths as a percentage of the longest acyclic path from the source to the sink is found to be relatively consistent when the mean proportional length (MPL) for different algorithms is analysed across different graphs. This suggests that the algorithms' efficiency is similar.

These findings suggest that the Ford-Fulkerson algorithm's efficiency is highly influenced by the augmenting route method selection. In terms of mean length and mean proportional length, the Shortest Augmenting Path and Max Capacity outperform each other; nevertheless, the number of pathways produced by each method differs, indicating potential trade-offs in computing efficiency. When choosing an augmenting route approach for the Ford-Fulkerson algorithm, it is important to take into account the properties of the particular graph as well as the needs of the issue.

Based on the conducted analysis, the MaxCap algorithm emerges as the preferred choice among the considered algorithms. The selection is grounded in several key observations:

1. **Efficiency Across Diverse Graph Characteristics:** The MaxCap algorithm consistently demonstrates competitive performance across various graph sizes and densities, maintaining efficiency even in scenarios with higher numbers of nodes and edges.

2. **Reduced Iterations:** The algorithm exhibits a notable capability to achieve the desired results with a reduced number of iterations compared to other algorithms. This is crucial as it implies faster convergence towards the maximum flow in the network.

3. **Optimized Time Complexity:** The MaxCap algorithm not only achieves fewer iterations but also manages the time complexity associated with finding augmenting paths

and creating residual graphs more efficiently. This contributes to overall computational speed and resource utilization.

4. **Robustness in Denser Graphs:** In denser graphs, where the number of edges is higher, the MaxCap algorithm maintains its efficiency, indicating robustness in scenarios where network complexity increases.

5. **Consideration of Time Complexity:** The decision is not solely based on the number of iterations but also takes into account the time complexity associated with each iteration. The MaxCap algorithm strikes a balance between the two, resulting in an overall efficient solution.

In conclusion, the MaxCap algorithm emerges as a well-rounded choice, offering efficient performance in terms of both iterations and time complexity. Its ability to handle larger and denser graphs with reduced computational overhead makes it a suitable candidate for the given problem of finding maximum flows in source-sink networks.

REFERENCES

[1] Dijkstra's algorithm for adjacency list representation. 3

[2] How to find shortest paths from source to all vertices using dijkstra's algorithm. 3