UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA

CARLOS WESTERMANN

Maximum s-t flow and Ford-Fulkerson algorithms

Porto Alegre
April 2025

1 Introduction

The following work presents an implementation and performance analysis of three algorithms from the Ford-Fulkerson family: Edmonds-Karp, Fattest-Path, and Random-DFS. The objective is to evaluate their practical efficiency, compare theoretical complexities, and test scalability across various graph structures and sizes.

2 Testing

2.1 Testing Environment

All test cases were run on a machine with the following specifications:

- Chip: Apple M3 Pro
- Total Number of Cores: 12 (6 performance and 6 efficiency)
- Memory: 18 GB

2.2 Testing Methodology

The evaluation of the algorithm and data structure evaluation was carried out in three phases, each evaluating different aspects of performance and scalability.

1. **Test instances:** All three algorithms were run on the same set of graph instances described with graphs of types described in the following table:

Name	Description
Mesh	A regular mesh $M = \{s\} \cup [r] \times [c] \cup \{t\}$ with r rows and c columns.
Random Level	Similar to a Mesh, but with $\delta \in_U \binom{[r]}{3}$.
Basic Line	A line $L = \{s\} \cup [nm] \cup \{t\}$.
Exponential Line	A Line where weights are exponentially smaller with increasing distance.
Dinic Bad Case	A line $L = [n]$ with arcs $(i, i+1)$, $i \in [n-1]$, and direct arcs (i, n), $i \in [n-1]$.

The test instances are the following:

Name	Arg1	Arg2	Arg3	Arg4
Mesh Graphs (Rows r, Columns c, Max Capacity C)				
mesh_100x100-300	100	100	300	
mesh_80x80-300	80	80	300	
mesh_120x120-500	120	120	500	
mesh_130x130-500	130	130	500	
mesh_140x140-500	140	140	500	
mesh_150x150-500	150	150	500	
Random Level Graphs (Rows r, Columns c, Max Capacity C)				
rlevel_60x60-300	60	60	300	
rlevel_80x80-400	80	80	400	
rlevel_100x100-500	100	100	500	
rlevel_110x110-500	110	110	500	
rlevel_120x120-500	120	120	500	
rlevel_130x130-500	130	130	500	
Basic Line Graphs (Groups n, Subgroups m, Degree d, Max Capacity C)				
basic_3.6k_nodes_deg6-300	60	60	6	300
basic_4.9k_nodes_deg7-400	70	70	7	400
basic_6.4k_nodes_deg8-500	80	80	8	500
basic_8.1k_nodes_deg8-500	90	90	8	500
basic_10k_nodes_deg8-500	100	100	8	500
basic_12.1k_nodes_deg8-500	110	110	8	500
Exponential Line Graphs (Groups n, Subgroups m, Degree d, Max Capacity C)				
exp_3.6k_nodes_deg6-300	60	60	6	300
exp_4.9k_nodes_deg7-400	70	70	7	400
exp_6.4k_nodes_deg8-500	80	80	8	500
exp_8.1k_nodes_deg8-500	90	90	8	500
exp_10k_nodes_deg8-500	100	100	8	500
exp_12.1k_nodes_deg8-500	110	110	8	500
Dinic Bad Case Graphs (Vertices n)				
dinic_bad_1k	1000			
dinic_bad_2.5k	2500			
dinic_bad_5k	5000			
dinic_bad_6k	6000			
dinic_bad_7k	7000			
dinic_bad_8k	8000			

3 Experimentation results and analysis

3.1 Operation Counting

3.1.1 Number of iterations

- **Edmonds-Karp and Fattest-Path:**
  As shown in the first two figures, both Edmonds-Karp and Fattest-Path consistently show ratios smaller than 1 across all graph types and sizes. This is to be expected, as the bounds for these algorithms ($nm/2$ for Edmonds-Karp and $m \log C$ for an upper limit C of maximum flow iterations for Fattest-Path) are pessimistic. This shows that both algorithms are generally efficient in practical scenarios, rarely approaching their theoretical worst-case iteration counts.
- **Randomized DFS:**
  The third figure shows that the randomized DFS algorithm almost always achieves a ratio $r \approx 1$, meaning that the number of iterations closely matches the theoretical maximum (which, in this case is an upper limit C of maximum flow iterations). This behavior is explained by the nature of the randomized DFS approach, as it selects augmenting paths without regard to their capacity or length. As a result, the algorithm may require as many iterations as the value of the maximum flow. This inefficiency becomes more apparent in graphs with high maximum flow values, as the algorithm is forced to perform a large number of augmentations, each contributing minimally to the total flow. This will be seen in time comparisons where the randomized DFS is stronlgy outperformed by other algorithms.

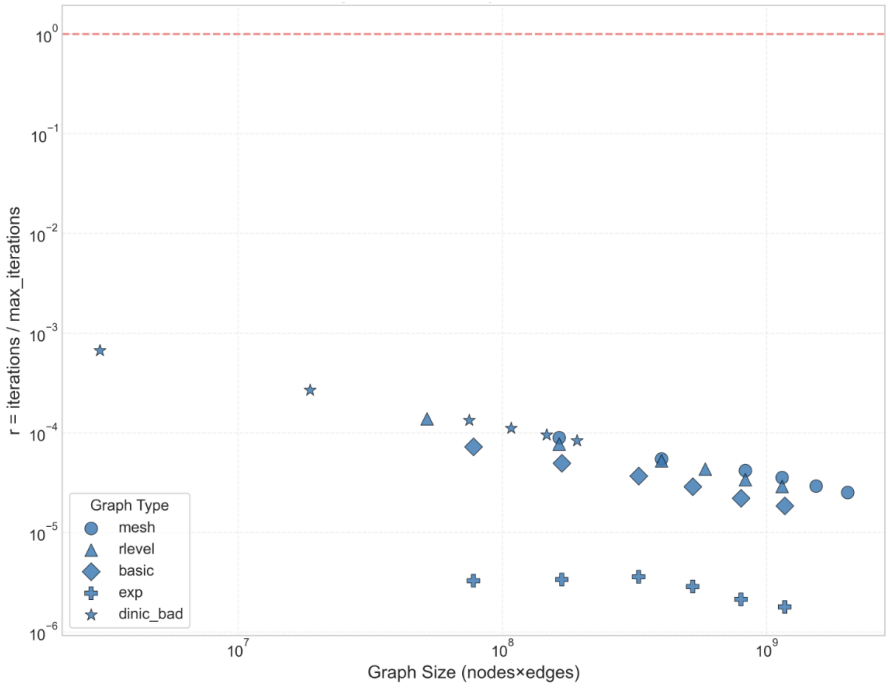

Figure 1 – Distribution of the Ratio $r = \frac{\text{Number of Iterations}}{\text{Maximum Theoretical Iterations}}$ over Graph Size for Each Test Instance for the Edmond Karp algorithm

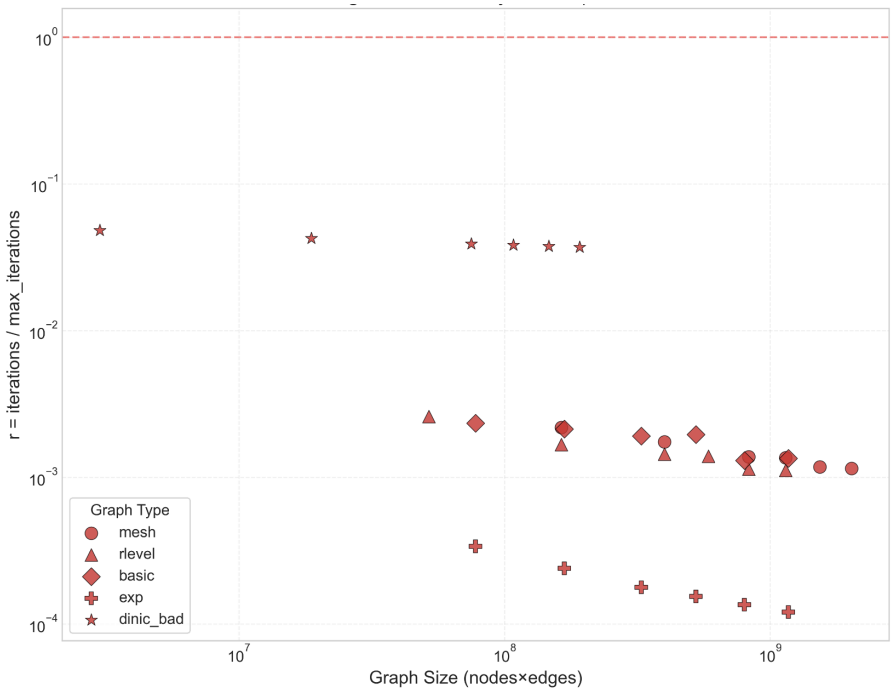

Figure 2 – Distribution of the Ratio $r = \frac{\text{Number of Iterations}}{\text{Maximum Theoretical Iterations}}$ over Graph Size for Each Test Instance for the Fattest Path algorithm

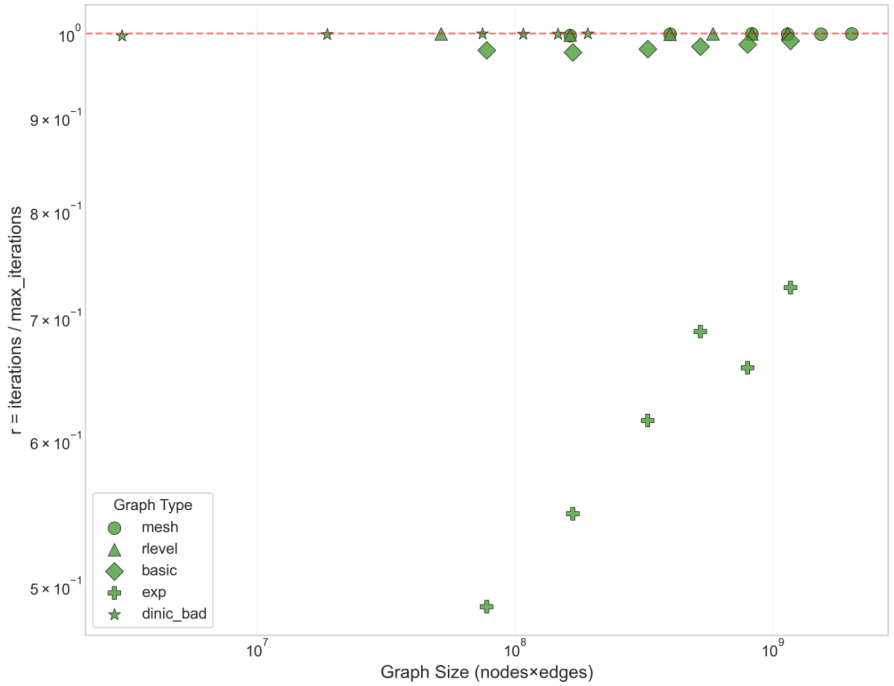

Figure 3 – Distribution of the Ratio $r = \frac{\text{Number of Iterations}}{\text{Maximum Theoretical Iterations}}$ over Graph Size for Each Test Instance for the randomized DFS algorithm

3.1.2 Critical arcs for Edmonds-Karp

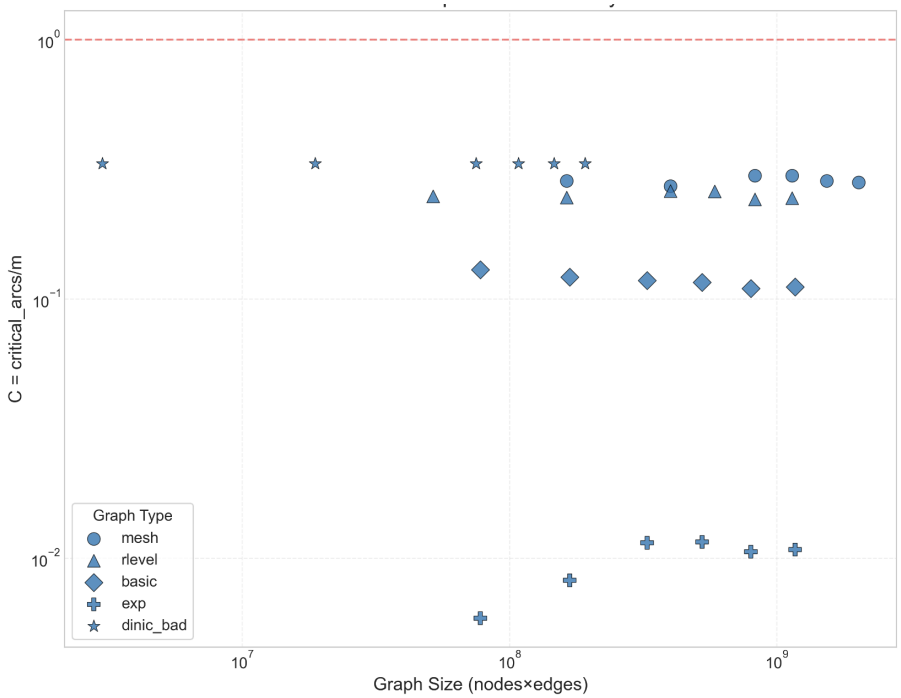

Figure 4 – Fraction of critical arcs $C = \frac{|\{a \in A : C_a > 0\}|}{m}$ for Edmonds-Karp, as a function of graph size for each test instance.

The plot shows that, for all tested graphs, only a small amount of the arcs become critical during the execution. This fraction C is always less than 1, as expected from the

theoretical analysis, confirming the upper bound for this implementation.

3.1.3 Visited arcs and vertices during search

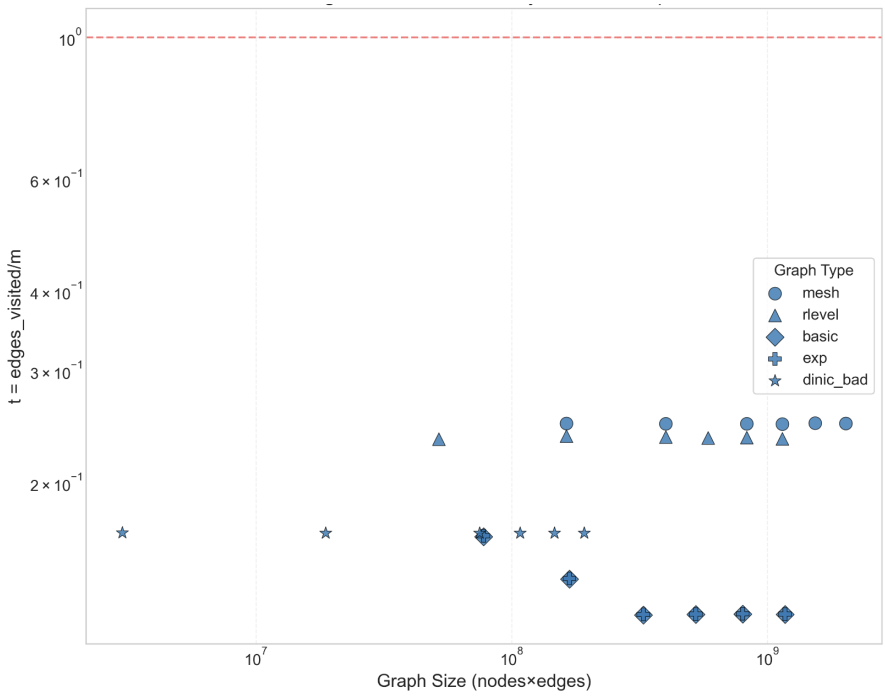

Figure 5 – Fraction of edges visited per iteration, $t = \frac{\text{edges\_visited}}{m}$, as a function of graph size for Edmonds-Karp.

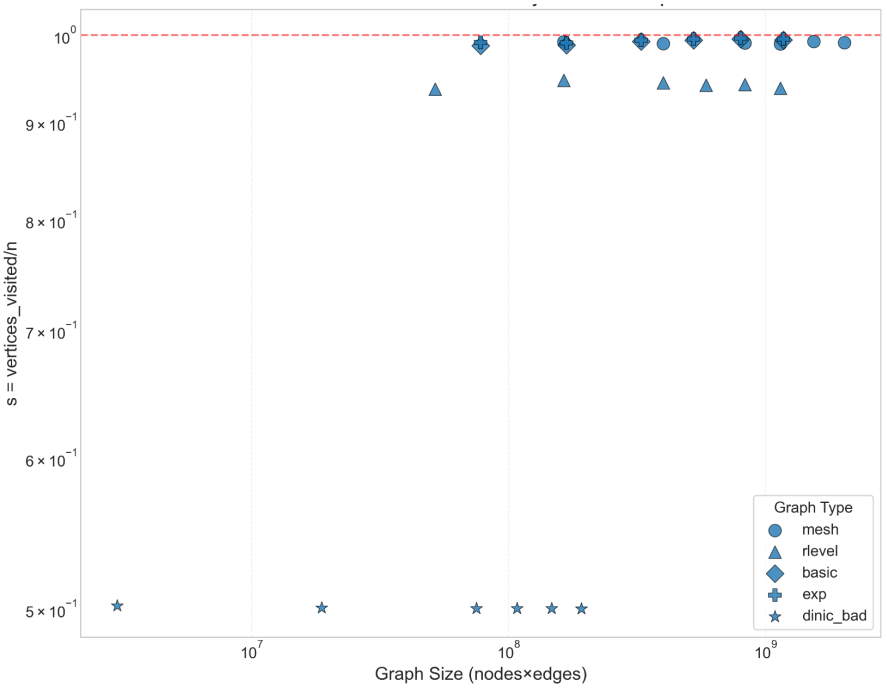

Figure 6 – Fraction of vertices visited per iteration, $s = \frac{\text{vertices\_visited}}{n}$, as a function of graph size for Edmonds-Karp.

From figures 5 and 6, we can confirm that the implementation for Edmonds-Karp respects the theoretical upper-bound of 1 for both the number of vertices and edges visited. In the case of Edmonds-Karp, we notice that during the search part of the algorithm, most vertices are visited for almost all instances, this makes sense since we are using BFS in this case. An interesting outliar in this case are the instances that are bad for dinic's algorithm, that consistently visited a smaller amount of the vertices. This may happen because of the way the graph is constructed, as a line with each node connected to the sink, we will have n iterations, the first being $1 => n$, the second $1 => 2 => n$ and so on, that results in:

$$\frac{\sum_{k=1}^{n} k}{n} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2}$$

This reasonably explains the data seen in the plot.

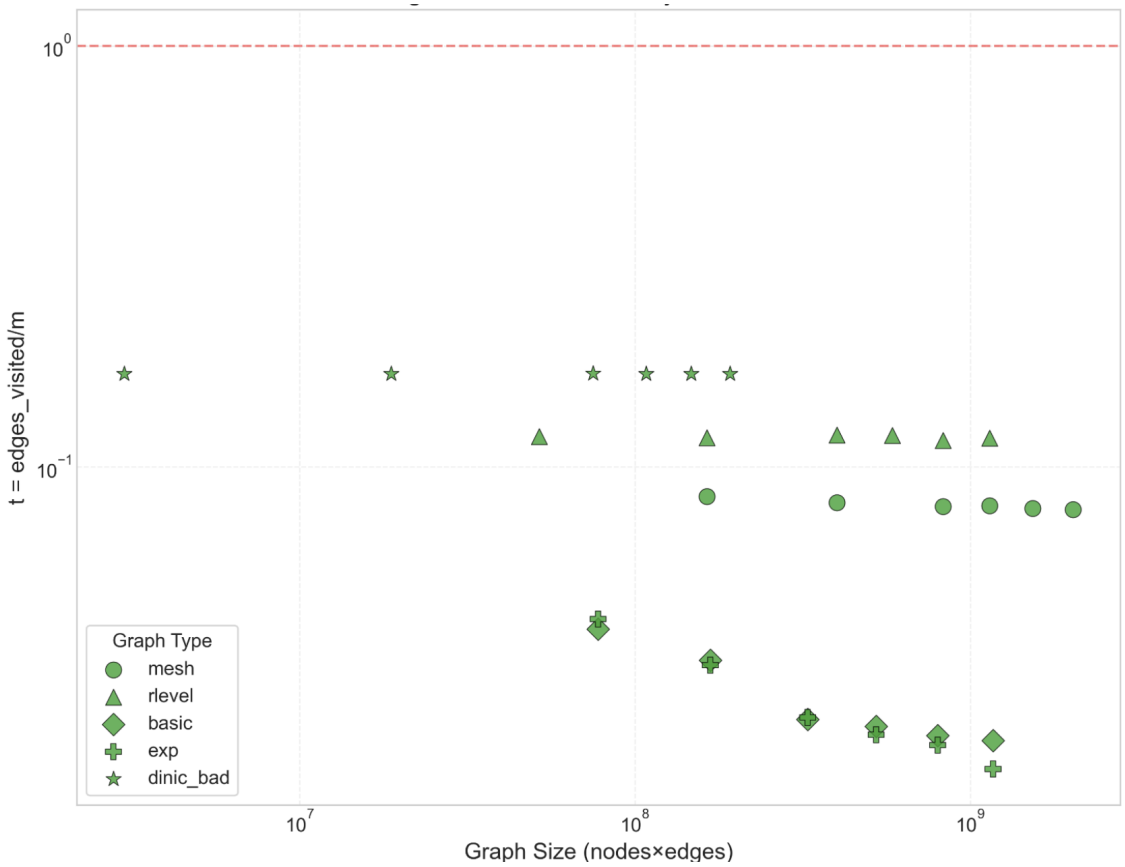

Figure 7 – Fraction of edges visited per iteration, $t = \frac{\text{edges\_visited}}{m}$, as a function of graph size for randomized DFS.

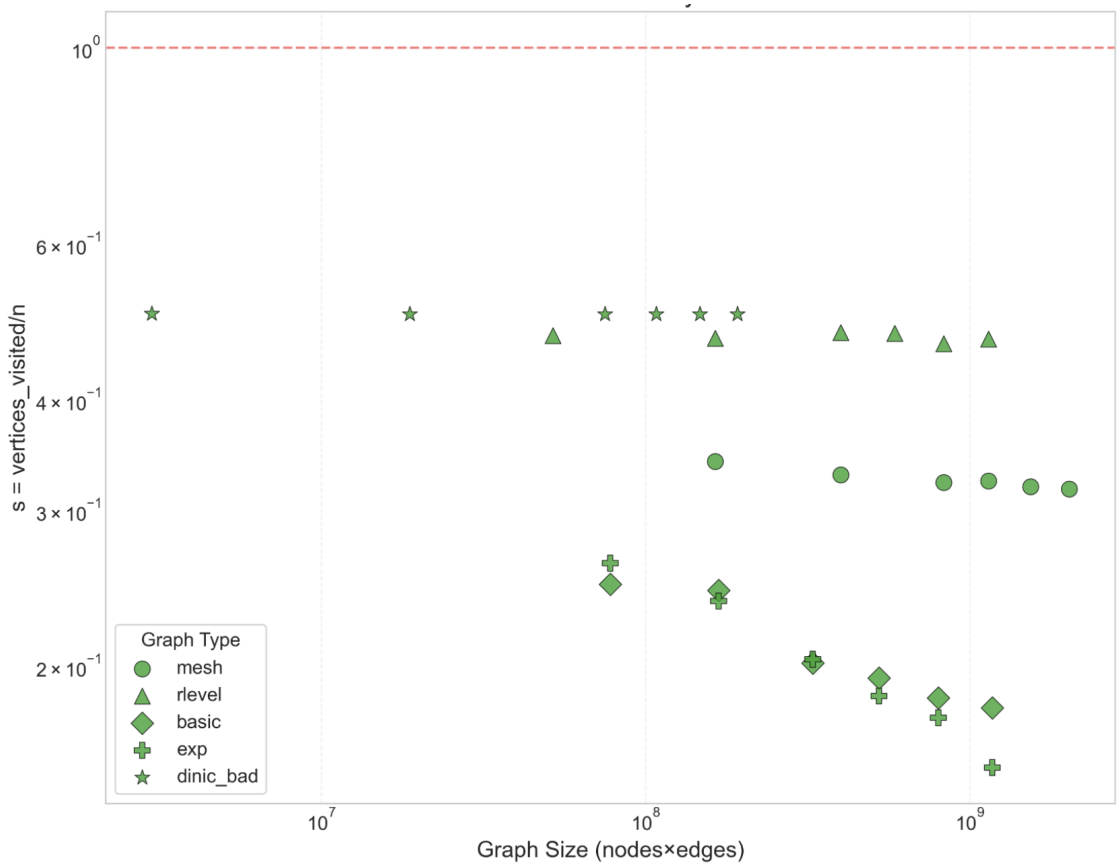

Figure 8 – Fraction of vertices visited per iteration, $s = \frac{\text{vertices\_visited}}{n}$, as a function of graph size for randomized DFS.

From figures 7 and 8, we observe the expected behavior for the randomized DFS implementation, that is to have both ratios $s, t < 1$. For the visited vertices, we can also see a different behavior in the visited vertices. This may be explained by the differences in BFS and DFS. This will also result in DFS needing on average for the test instances more iterations, as it may be looking for suboptimal paths for the problem.

3.1.4 Heap operations for Fattest Path

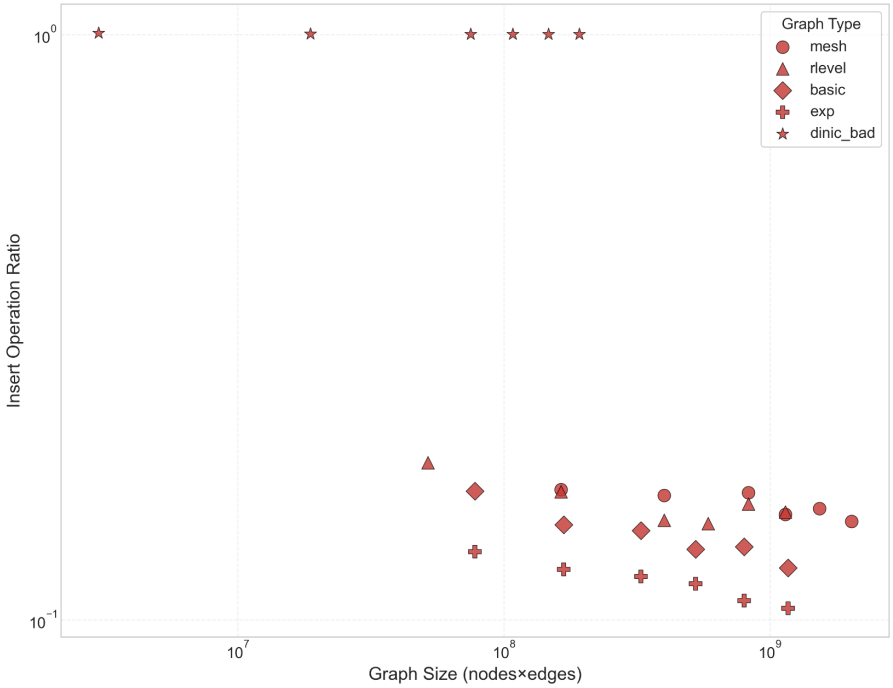

Figure 9 – Average ratio of `insert` operations per iteration for Fattest-Path, as a function of graph size.

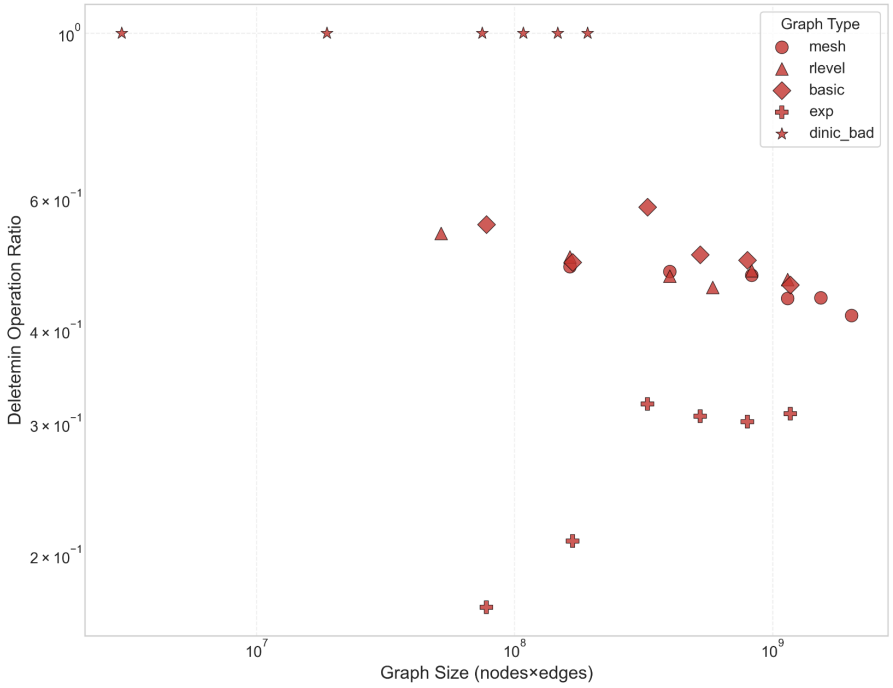

Figure 10 – Average ratio of `deletemin` operations per iteration for Fattest-Path, as a function of graph size.

From figure 9 and 10 we can also understand that the implementation for Fattest Path is correct in regards to the heap operations used for the priority queue. Both ratios are consistently below 1 for all graph types. This is expected because, in each iteration,

usually not all vertices are inserted into or removed from the heap. In practice, the heap should only have a subset of the whole graph stored at each iterations. We also see an interesting behavior for Dinic bad cases in this metric as well, similar to what happens in the Edmonds-Karp. This is because of the structure of the graph that forces the algorithm to process all vertices in each augmentin path.

3.2 Time Complexity Measurement

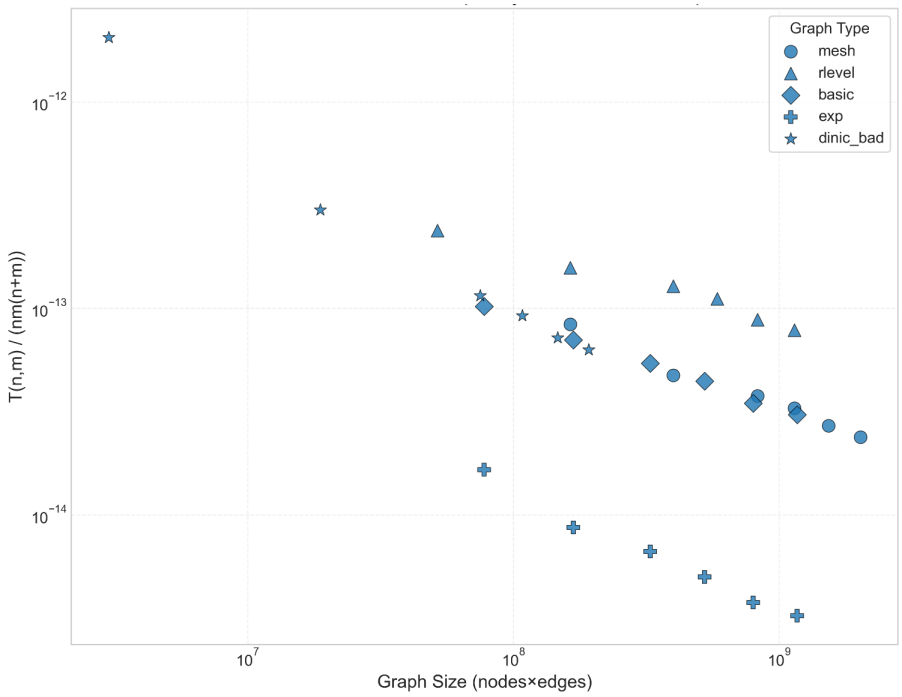

Figure 11 – Ratio of execution time to pessimistic complexity, $\frac{T(n,m)}{nm(n+m)}$, for the Edmonds-Karp algorithm, as a function of graph size. Each point represents a test instance.

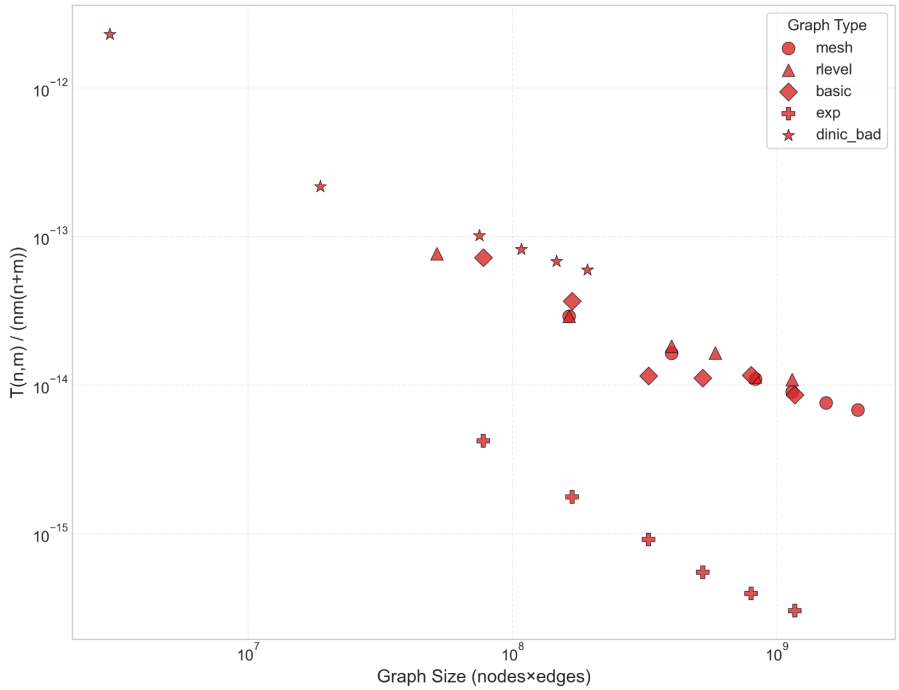

Figure 12 – Ratio of execution time to pessimistic complexity, $\frac{T(n,m)}{nm(n+m)}$, for the Fattest-Path algorithm, as a function of graph size. Each point represents a test instance.

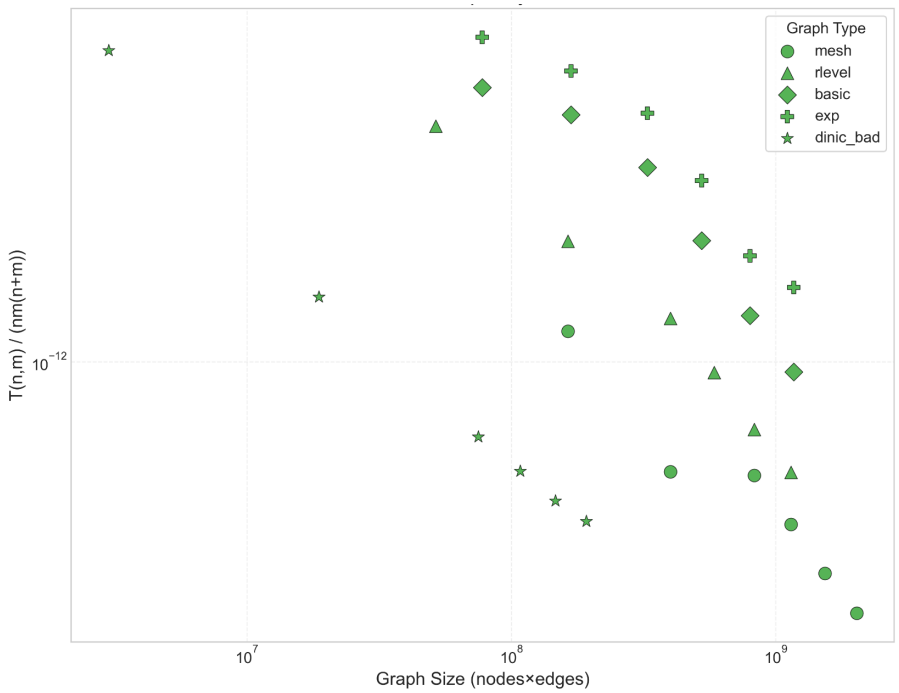

Figure 13 – Ratio of execution time to pessimistic complexity, $\frac{T(n,m)}{nm(n+m)}$, for the Randomized DFS algorithm, as a function of graph size. Each point represents a test instance.

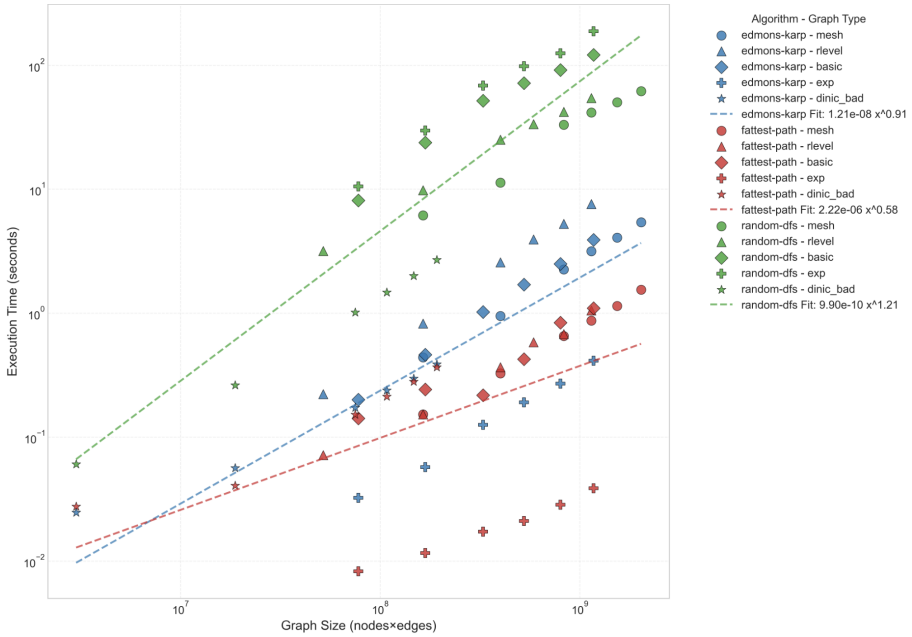

Figure 14 – Comparison of execution time of all algorithms on all instances

In regards to the execution time, we see that all implementations fall in a similar order of magnitude, however, Edmonds-Karp and Fattest Path being consistently faster then the DFS, with the second taking slight advantage on average. An observation to be made is that, while the theoretical pessimistic time complexity for Ford-Fulkerson algorithms is $O(nm(n+m))$, the empirical execution time for all tested algorithms is much faster, as shown by figures 11, 12 and 13. The dashed lines in figure 14 represent power-law fits $T(\text{size}) \sim a \cdot \text{size}^b$ for each algorithm, indicating that the empirical complexity scales with graph size raised to a power less than the pessimistic bound. This demonstrates that all implementations, on average, perform better than the theoretical worst case.

4 Conclusion

The experimentation and analysis presented in this work provide an evaluation of three Ford-Fulkerson variants: Edmonds-Karp, Fattest-Path, and Random-DFS. Operation counting reveals that all implementations align with their theoretical complexity bounds, with both Edmonds-Karp and Fattest-Path consistently performing far below their pessimistic upper limits. The fraction of critical arcs and visited vertices/edges confirms the theoretical analysis across all graph types. Time complexity measurements demonstrate that all algorithms execute significantly faster than their worst-case bounds of $O(nm(n+m))$, with Fattest-Path showing slight advantages in most test instances. These findings highlight the notable gap between theoretical and practical performance

in maximum flow algorithms, with Random-DFS demonstrating why theoretical bounds matter in practice.