Comparing the Efficiency of the Edmonds-Karp Algorithm to the Dinic's Algorithm

Ethan Bown and Rick Kabuto

Problem Statement

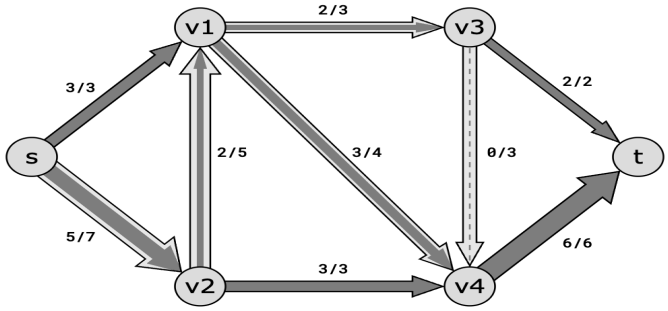
Many algorithms exist to solve an important type of graphing/networking problem: maximum flow. Maximum flow, and therefore algorithms to handle this issue, are used to determine the greatest flow rate from a source vertex to a sink vertex. Commonly the source node is denoted as S and the sink node as T; additionally, a flow from S and T can be referred to as S-T flow, where the maximum S-T flow is the solution to the maximum flow problem. The largest S-T flow is directly equivalent to the smallest S-T cut by the max-flow min-cut theorem, where a cut carves out two disjoint subsets inside the graph, separating some vertices from others. While there exists a multitude of algorithms to tackle this problem, two contrasting examples are the Edmonds-Karp Algorithm and the Dinic's Algorithm. The goal when specifically choosing the Edmonds-Karp Algorithm and the Dinic's Algorithm is to compare their efficiency–each performs better in certain cases where the other struggles. What this highlights is the importance of understanding how different algorithms, while solving the same problem, can perform significantly differently. The best method to demonstrate these qualities is by testing both algorithms on a real world problem. Traditional applications include transportation problems, where the optimization of traffic flow for cars or other automobiles increases the effectiveness of said vehicles. For this project, we chose two problem scenarios to test the implementation of Edmonds-Karp and Dinic's Algorithm on: a sample problem from GeeksforGeeks and a real-world example of traffic engineers exploring traffic congestion from a central parking area to a stadium. In the real-world problem, the goal is to find the maximum flow from a source (the central parking area) to the sink (the stadium) of cars, representing this relationship using a network. From these problem sets, we aim to both verify the output that each algorithm produces while also comparing the actual running times. That way, both the theoretical and the practical aspects of maximum flow can fully be appreciated.

Algorithm Description

Before proceeding to the application of these algorithms, a proper analysis of their rationale remains crucial. During the processing of the Edmonds-Karp algorithm, flow is sent through the network only once per iteration. Breadth-first search (BFS) is the main backend algorithm behind Edmonds-Karp. BFS's primary goal is to find an S-T path where its capacity is available–meaning that flow can be sent through that path. Such paths are referred to as augmenting paths, and BFS will send the maximum amount of flow through them once discovered. If no augmenting path exists, then that means that the maximum flow has been found and the algorithm should return its result. Otherwise, the algorithm must determine something called the bottleneck capacity for a path, which is equivalent to the minimum residual capacity for said path. Once this is done, each forward edge along the path will have its edge capacity decreased by the bottleneck and every backwards edge will have its edge capacity increased by

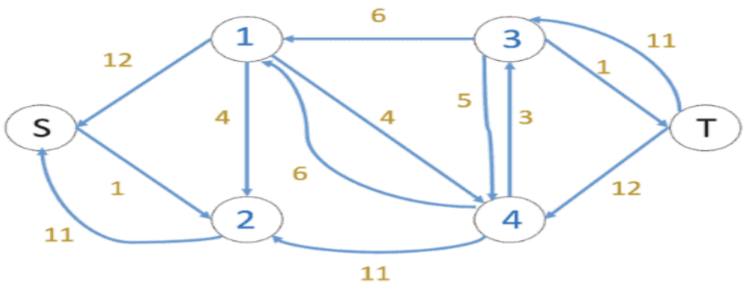
the bottleneck. This process overall will update the residual graph, allowing for the next iteration of the algorithm to commence until no augmenting paths exist. The bottleneck value is returned once the algorithm terminates.

Figure 1: Example of a Residual Graph for Edmonds-Karp Where No More Possible Flow Can Be Sent Through the Graph, Resulting in a Maximum Flow of 8 (W3Schools, n.d.)



Dinic's algorithm, while operating on many of the same key principles, takes a different approach to solving maximum flow. In addition to BFS to find if an S-T path exists, Dinic's algorithm uses Depth-first Search (DFS) to push flow. Dinic's algorithm first uses BFS to build a level graph, where each node is designated with a level corresponding to the shortest distance from said node to S. Edges with a residual capacity of 0 are removed from the graph and if the sink is not reachable, the algorithm returns so. To find the blocking flow, a flow counter is set to 0 and DFS pushes flow from the S to T and each edge starts with initially 0 flow. For each level's edge, the path's minimum residual capacity directly relates to the flow. Forward and reverse edges are augmented similarly to Edmonds-Karp, but the total flow is summed into the flow counter at the end of the iteration. Once this process completes, the previous graph flow is augmented by the new graph flow and a new DFS iteration begins. Once DFS is no longer able to find an augmenting path, the algorithm terminates and returns the maximum flow.

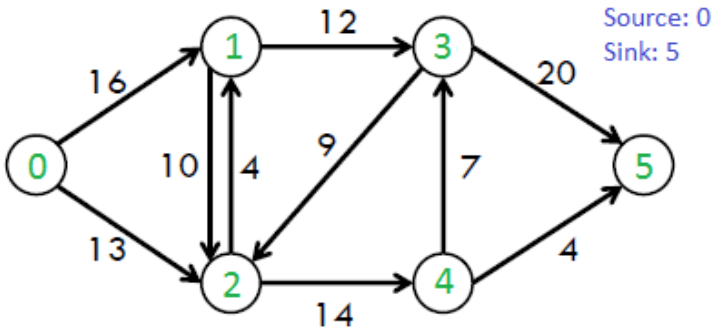
Figure 2: Example of a Residual Graph for Dinic's Algorithm Where No More Possible Flow Can Be Sent Through the Graph, Resulting in a Maximum Flow of 23 (Bouguezzi, 2023)



Results and Discussion

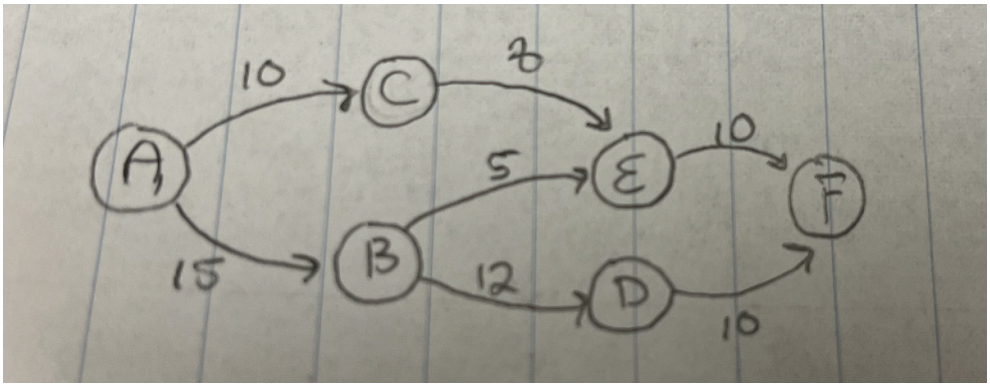
To validate the implementation of both algorithms, a standard test case provided by GeeksforGeeks was used. This test case involves a graph with a source node (0), a sink node (5), and an expected maximum flow output of 23. Both algorithms were applied to this graph to verify the maximum flow value, creating an ideal benchmark for the correctness of the implementations. Dinic's and Edmonds-Karp computed the outputted maximum flow of 23, consistent with the expected result. This agreement confirms that the logic and arithmetic of both implementations are accurate. By successfully matching the expected output, the experiment demonstrated that both implementations are accurately augmenting the paths and the residual graphs are updated properly, demonstrating controlled test cases.

Figure 3: Graphical Representation of GeeksforGeeks Maximum Flow Test Case for Dinic's and Edmonds-Karp (GeeksForGeeks, 2023)



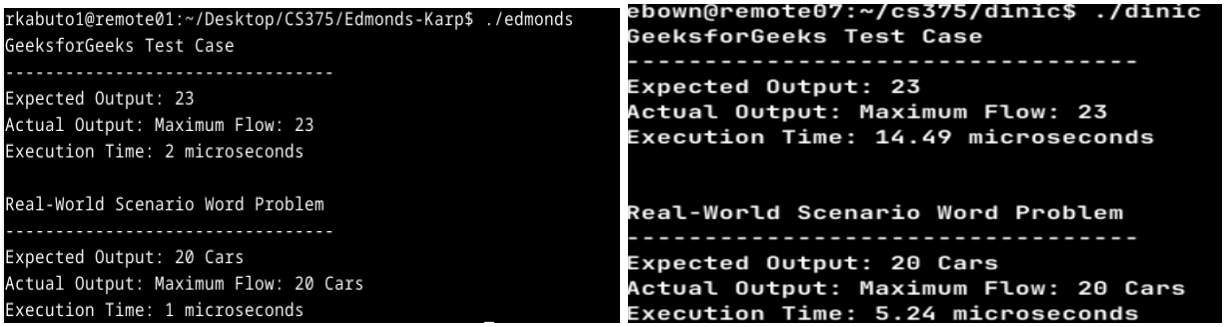
The experiment proceeded by applying the algorithms to a real-world scenario. The problem models a network of roads connecting intersections, where each road has a specific capacity representing the number of cars it can handle per unit time. The task was to determine the maximum number of cars that could travel from a central parking area (source) to a stadium (sink) during an event without causing congestion. The graph, shown in Figure 4, is initialized with various road capacities, where the expected output was a maximum flow of 20 cars.

Figure 4: Graphical Representation of Real World Problem Maximum Flow Test Case for Dinic's and Edmonds-Karp



Both algorithms successfully computed the maximum flow of 20 cars, confirming the ability to handle real-world applications. However, the methods by which the algorithms achieved this result differ, possessing distinct approaches. Edmonds-Karp relied on repeated breadth-first searches (BFS) to find augmenting paths. This requires less iterations, leading to lower running times. On the other hand, Dinic's algorithm demonstrated worse efficiency by constructing a level graph and utilizing depth-first searches (DFS) to push flow. The choice between Edmonds-Karp and Dinic's algorithms depends on the specific characteristics of the problem. Edmonds-Karp is best suited for small to medium-sized graphs, offering simplicity and ease of implementation. Edmonds-Karp has a time complexity of $O(VE^2)$. Dinic's algorithm is preferred for larger or more complex graphs with many nodes and edges. Dinic's algorithm has a time complexity of $O(V^2E)$. Thus sparse graphs or varying path capacities present better performance due to its complexity.

Figure 5: Running Time & Outputs For Both Algorithms



Conclusion

The comparison of Edmonds-Karp and Dinic's algorithms emphasizes their shared ability to accurately solve maximum flow problems while demonstrating differences in efficiency based on graph structure and problem size. By validating the algorithms against a known test case and applying them to a real-world scenario, the experiment confirmed their correctness and practicality. In conclusion, both algorithms find the optimal solution for maximum flow problems, but their arithmetic varies with context. Edmonds-Karp is ideal for smaller, simpler graphs, whereas Dinic's algorithm excels in more complex, real-world applications. Understanding these distinctions ensures that the right algorithm is chosen for the task at hand, optimizing computational efficiency and resource utilization.

References

Bouguezzi, S. (2023, May 31). *Network flow: Dinic's algorithm*. Baeldung. Retrieved November 23, 2024, from https://www.baeldung.com/cs/dinics

Erickson, J. (2019). *Applications of flows and cuts*. University of Illinois at Urbana-Champaign. Retrieved from https://jeffe.cs.illinois.edu/teaching/algorithms/book/11-maxflowapps.pdf

GeeksforGeeks. (2023, June 1). *Ford-Fulkerson algorithm for maximum flow problem*. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/

Udacity. (2015, February 23). *Analysis of Edmonds-Karp - Georgia Tech - Computability, Complexity, Theory: Algorithms* [Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=FlIB73vSl4s

Wikipedia contributors. (n.d.). *Cut (graph theory)*. In *Wikipedia, The Free Encyclopedia*. Retrieved December 1, 2024, from https://en.wikipedia.org/wiki/Cut_(graph_theory)

Wikipedia contributors. (n.d.). *Dinic's algorithm*. In *Wikipedia, The Free Encyclopedia*. Retrieved November 23, 2024, from https://en.wikipedia.org/wiki/Dinic%27s_algorithm

Wikipedia contributors. (n.d.). *Edmonds-Karp Algorithm*. In *Wikipedia, The Free Encyclopedia*. Retrieved November 23, 2024, from https://en.wikipedia.org/wiki/Edmonds%E2%80%93Karp_algorithm

Wikipedia contributors. (n.d.). *Maximum flow problem*. In *Wikipedia, The Free Encyclopedia*. Retrieved December 1, 2024, from https://en.wikipedia.org/wiki/Maximum_flow_problem

WilliamFiset. (2018, November 7). *Dinic's Algorithm | Network Flow | Graph Theory* [Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=M6cm8UeeziI

W3Schools. (n.d.). *Edmonds-Karp algorithm*. W3Schools. Retrieved November 23, 2024, from https://www.w3schools.com/dsa/dsa_algo_graphs_edmondskarp.php