**Minimum-Cost Flow Problem**

# **1.** Problem Description

The main objective of this project is to study and find the solution to the Minimum-Cost Flow Problem – a generalization of the more basic Maximum Flow problem. This is not only a problem of finding the maximum flow from a source node to a sink node of a graph but also the minimization of the total flow is the cost incurred by edge costs. Every edge has its maximum flow capacity, cost and each edge flow makes an effort to deliver the total flow demand (d). The project emphasizes the implementation and comparative analysis of four algorithms: Capacity Scaling (CS), Successive Shortest Path (SSP), Successive Shortest Path with Capacity Scaling (SSPCS), and Cycle-Canceling (CC).

This project postulates also aims at developing a random graph generator which can generate directed Euclidean graphs with specific configurable attributes like density, capacities and costs. The generated graphs are evaluated and the algorithms are applied on them in order to measure efficiency in terms of total flow, minimum cost and time consumed. Findings are presented in tabular forms for analysis at your convenience.

# **2. Implementation Details**

## **Graph Generation**

To generate controllers for the test phase, a graph generator was developed to produce random Euclidean graphs where nodes are located randomly in the unit square [0,1]×[0,1]. Connecting nodes are created based on the defined distance criterion to best block the edges ‘ formation. The capacities and costs of every edge are then generated randomly between two predetermined maximum amounts. To facilitate computing, the derived graphs are maintained in an adjacency list form. These graphs are used later for simulations and also for performing the related algorithms analysis.

## **Largest Connected Component (LCC) identification**

In order to detect the largest connected component of each graph, a Depth-First Search (DFS) is implemented. The source node is chosen as the starting node of the DFS, and the sink node is selected as the node which is most far from the source node, so the meaningful source-sink pairs for flow computation can be obtained.

## **Algorithm Implementations**

1. **Capacity Scaling Algorithm (CS):** This algorithm begins with using the scaling factor Δ equal to the maximum of the capacity of at least one of the edges of the graph. It limits the increase in the path search to edges with residual capacities not less than Δ, reducing the scaling factor by half in each step down to 1. The approach can be viewed as realizing the optimal balance between augmenting path identification and computational load as well as flow differentiation.
2. **Successive Shortest Path Algorithm (SSP):** The whole procedure of finding the minimum cost path from source to the sink is carried out by running a shortest path algorithm like Bellman-Ford in SSP algorithm. Capacity in the network is incremented along these paths until the necessary demand is achieved, or no more augmenting paths are present. Peculiarities of this algorithm: it is efficient in terms of cost minimization, but may be slow for large graphs.
3. **Hybrid Algorithm (SSPCS):** Extending the work on CS and SSP, this algorithm employs scaling to limit paths to higher capacity edges first, then gradually descends to the finer MSS. This hybridization effectively saves computational costs while limiting the level of error during the determination of flow optimality.
4. **Cycle-Canceling Algorithm (CC):** The CC algorithm also detects and removes cost-reducing cycles in the residual graph for the subsequent purchase. Based on these priorities arranged in a queue the algorithm cycles with the potential of reducing the total flow cost are gradually enhanced. This method is highly computational but efficient in terms of avoiding excess costs in complex graphs.

# **3. Implementation Correctness**

## **Software Testing**

A comprehensive testing process was carried out to reach the conclusion of equivalency of the implemented functions. This paper examined the performance of the random graph generator under different boundary conditions. For instance:

* For the edge density parameter when the value of r was 0, implied that their was no connectivity at all and the graph generated had no edges.
* As an indication of the density of the graph, when r was set to 1, almost every pair of nodes was connected by edges other than self-loops.
* Randomly assigned capacities and costs were checked to surely be randomly generated within the uppercap and uppercost limits.
* Whenever a flow augmentation occurred in the algorithms' execution, the correctness of the residual graph updates was checked. It was also confirmed that all edges of the residual graph strictly adhered to the flow conservation constraints for every analyzed component.

## **Algorithm Validation**

Algorithm correctness was verified using small, hand-crafted graphs with known solutions. For instance, on a node graph with specified capacities and costs, all four algorithms produced identical flows and minimum costs, matching manually computed results. Additionally, discrepancies observed during simulations were debugged by inspecting augmenting paths and analyzing residual graphs.

## **Simulation Accuracy**

Simulations were validated by comparing results across algorithms for identical graphs. For example, the total flow (f) and cost (MC) computed by different algorithms were consistent for each graph, confirming implementation accuracy.

# **4. Simulation Results**

## **Graph Characteristics**

#### **Graph Characteristics**

The generated graphs varied significantly in size, edge density, and edge weights, as reflected in their structural properties. Each graph was analyzed to calculate the following key metrics.

These characteristics are summarized in **Table 1**, providing insights into the diversity of the graph structures used in the simulation.

**Table 1: Graph Characteristics**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Graph** | **Nodes (V)** | **Edges (E)** | **Delta\_in** | **Delta\_out** | **Avg Degree** |
| 1 | 100 | 2000 | 30 | 29 | 20.00 |
| 2 | 200 | 7988 | 53 | 54 | 39.94 |
| 3 | 100 | 2950 | 43 | 44 | 29.50 |
| 4 | 200 | 12014 | 81 | 83 | 60.07 |
| 5 | 100 | 1960 | 33 | 29 | 19.60 |
| 6 | 200 | 7876 | 58 | 54 | 39.38 |
| 7 | 100 | 2961 | 38 | 40 | 29.61 |
| 8 | 200 | 11926 | 73 | 77 | 59.63 |

## **Algorithm Performance**

The performance of the four algorithms was evaluated based on several metrics.Table 2 provides a detailed comparison of these metrics across algorithms for each graph.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Graph** | **Algorithm** | **f** | **MC** | **Paths** | **Execution Time** | **ML** | **MPL** |
| 1 | SSP | 21 | 342 | 19 | 0.004534 | 7 | 0.777778 |
| 1 | CS | 16 | 260 | 7 | 0.007828 | 5.928571 | 0.846939 |
| 1 | CC | 21.85 | 352.85 | 30 | 0.005084 | 6.071429 | 0.758929 |
| 1 | HYBRID | 21.85 | 342.15 | 18 | 0.005877 | 6.33294 | 0.875579 |
| 2 | SSP | 48 | 1208 | 55 | 0.068336 | 8.676471 | 0.78877 |
| 2 | CS | 34 | 839 | 16 | 0.074138 | 7.794118 | 0.779412 |
| 2 | CC | 50.35 | 1260.2 | 69 | 0.090758 | 7.647059 | 0.695187 |
| 2 | HYBRID | 50.35 | 1125.8 | 43 | 0.09467 | 7.492824 | 0.847341 |
| 3 | SSP | 74 | 864 | 68 | 0.032316 | 5.363636 | 0.766234 |
| 3 | CS | 48 | 556 | 25 | 0.034364 | 4.113636 | 0.822727 |
| 3 | CC | 81 | 931 | 94 | 0.032125 | 4.181818 | 0.69697 |
| 3 | HYBRID | 85.5 | 895 | 58 | 0.04432 | 3.99046 | 0.77 |
| 4 | SSP | 144 | 1072 | 125 | 0.296222 | 4.121951 | 0.58885 |
| 4 | CS | 91 | 710 | 43 | 0.186555 | 2.890244 | 0.578049 |
| 4 | CC | 177 | 1296 | 172 | 0.281148 | 2.914634 | 0.582927 |
| 4 | HYBRID | 185.525 | 1292.575 | 110 | 0.27322 | 2.69111 | 0.634479 |
| 5 | SSP | 113 | 5680 | 62 | 0.01674 | 5.930233 | 0.658915 |
| 5 | CS | 74 | 2638 | 22 | 0.017841 | 4.744186 | 0.677741 |
| 5 | CC | 182 | 9091 | 83 | 0.016516 | 4.906977 | 0.613372 |
| 5 | HYBRID | 188 | 8292 | 86 | 0.03597 | 5.370364 | 0.6567 |
| 6 | SSP | 178 | 7059 | 99 | 0.129075 | 5.30303 | 0.662879 |
| 6 | CS | 120 | 4453 | 41 | 0.264085 | 4.242424 | 0.707071 |
| 6 | CC | 258 | 10349 | 135 | 0.156036 | 4.287879 | 0.612554 |
| 6 | HYBRID | 286 | 11177 | 128 | 0.2688 | 4.695897 | 0.739899 |
| 7 | SSP | 130 | 4951 | 70 | 0.038241 | 4.934783 | 0.704969 |
| 7 | CS | 78 | 2982 | 24 | 0.041194 | 3.934783 | 0.786957 |
| 7 | CC | 206 | 7931 | 88 | 0.035698 | 3.76087 | 0.626812 |
| 7 | HYBRID | 193 | 7445 | 84 | 0.076677 | 4.046987 | 0.745963 |
| 8 | SSP | 323 | 14165 | 198 | 0.831141 | 5.5 | 0.6875 |
| 8 | CS | 158 | 6158 | 70 | 0.738733 | 4.40625 | 0.734375 |
| 8 | CC | 483 | 20338 | 256 | 0.746398 | 4.1875 | 0.598214 |
| 8 | HYBRID | 517 | 21745 | 245 | 2.263062 | 4.85588 | 0.709473 |

The Capacity Scaling Algorithm (CS) did very well when tested on graphs with large edge capacities. As it will be shown here, by only considering paths with residual capacities larger than or equal to the current scaling factor Δ, it effectively steers clear from any computations that would involve paths with small flows. By doing this, there were few iterations and hence relatively low execution times were registered. Nevertheless, the total flow (f) CS obtained was for some cases lower than that of the SSP or Hybrid algorithms especially where graphs with small residual capacities were important in creating the necessary demand.

Comparing the average MC results, SSP remained the most efficient in finding the lowest MC in most of the graphs. This result points to its capability of preventing flow costs by repeatedly identifying the lowest cost these augmenting paths. This precision, however, greatly increased computational times with SSP, using many more iterations and more time for larger ‘denser’ graphs. The substantial search required to find alternate paths that constitute shortest paths in such graphs added to the computational complexity.

The proposed Cycle-Canceling Algorithm (CC) was particularly successful in minimizing flow cost due to the identification of costly cycles in the residual graph. This approach made it possible to obtain the flow achieved at a very minimum cost. However, the algorithm was not very efficient in terms of the computations required especially where dense graphs with many cycles could be found and canceled. Interestingly, in many cases, CC resulted in the highest total flow (f) while, at the same time, requiring more time for completion because of the iterative nature of the cycle-canceling steps.

Hybrid Algorithm was developed to achieve adequate results from both the SSP and CS, offering average elapsed time and cost-effective solutions. It was particularly successful with all graph types rating high in both total flow and cost. Thus, analysing the results of the solving with the Hybrid algorithm which applies the elements of the successive shortest path approach after capacity scaling in the initial stage, I concluded that the execution time remained moderate and the outcome was close to the optimal in most cases.

## **Evaluation of Results**

The observations above are consistent with theoretical expectations and reflect the inherent trade-offs between computational efficiency, flow maximization, and cost minimization for the algorithms.

* Capacity Scaling is efficient for networks with high-capacity edges due to its ability to avoid unnecessary computations.
* SSP achieves the most cost-efficient flows but can struggle with scalability for larger graphs.
* Cycle-Canceling provides the most comprehensive cost reduction but is impractical for dense networks due to its high computational demand.
* Hybrid balances these trade-offs effectively, making it a reliable choice for a wide range of graph types.

# **5. Conclusion**

This work points out the spare capacity, flow and cost approaches when implementing 4 minimum cost flow algorithms. Whereas the Capacity Scaling and Hybrid algorithms dealt fairly well with the performance aspects of the problem, the cost-driven Successive Shortest Path algorithm outperformed the others in terms of cost, while the Cycle-Canceling algorithm provided the highest flow rates at the cost of time. These results imply that selection of the algorithms needs to be made based on the characteristics of a given graph and the expected performance.

# **6. Teamwork Distribution**

* **Graph Generator and LCC Identification**: [Name]
* **Algorithm Implementation**: [Name]
  + CS, SSP: [Name]
  + SSPCS, CC: [Name]
* **Simulation Design and Testing**: [Name]
* **Results Analysis and Report Preparation:** [Name]

# **7. References**

1. Jiang, Z., Hu, X., & Gao, S. (2013). A parallel ford-fulkerson algorithm for maximum flow problem. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)* (p. 70). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).
2. Wahyuningsih, S., Octoviana, L. T., & Qohar, A. (2024, September). Performance of the maximum flow algorithm and its application to traffic density. In *AIP Conference Proceedings* (Vol. 3235, No. 1). AIP Publishing.
3. Ahuja, R. K., & Orlin, J. B. (1995). A capacity scaling algorithm for the constrained maximum flow problem. *Networks*, *25*(2), 89-98.
4. Çalışkan, C. (2012). A computational study of the capacity scaling algorithm for the maximum flow problem. *Computers & operations research*, *39*(11), 2742-2747.