Using the Edmonds-Karp Algorithm to Find the Maximum Flow Through a Network

Student Number: 710017661

November 2023

Abstract. The applications of flow networks are vast and varied, ranging from data mining to water plumbing and even to airline scheduling. Therefore, an efficient method which could optimise these networks, would be of appreciable interest to researchers and non-researchers alike. Apart from finding the maximum flow through a flow network, these methods enable better analysis of these networks, uncovering paths or objects which may hinder the network. In this paper an algorithm which can be used to solve the maximum flow problem is systematically assessed, with the motivations, principles, applications and complexity of the algorithm being explored. Modern algorithms which improve upon and surpass the limitations of this algorithm are investigated. Additionally, pseudocode is supplied for this algorithm and any supplemental algorithms used by it.

Word Count: 1487

I certify that all material in this report which is not my own work has been identified.

1 Introduction

Graph theory seeks to model graphs or networks, where these are mathematical structures representing a collection of nodes and edges [1]. Here nodes are any object we seek to model and edges represent the connection or path between two objects; these edges can have a specific direction or no direction at all. Graphs that contain directed edges are referred to as directed graphs, with un-directed graphs referring to the contrary.

Flow networks are a subsection of problems in graph theory that analyse directed graphs, where each edge has a capacity, with this being a non-negative number representing the maximum amount of *flow* that can be passed through the edge. Here flow is a value representing the volume of an arbitrary unit the graph tracks. The volume of flow entering a node must be equal to the volume of flow leaving it, unless it is the *source* or *sink* node. These refer to the nodes which only have an outgoing flow and incoming flow respectively, often seen as the start and end of the network.

Finding the maximum flow through a network, also known as the *maximum flow problem*, is an area of great interest with a breadth of applications, i.e. modelling the maximum amount of traffic that can be passed through a computer network, and is also the problem that *Edmonds-Karp* [2] seeks to solve.

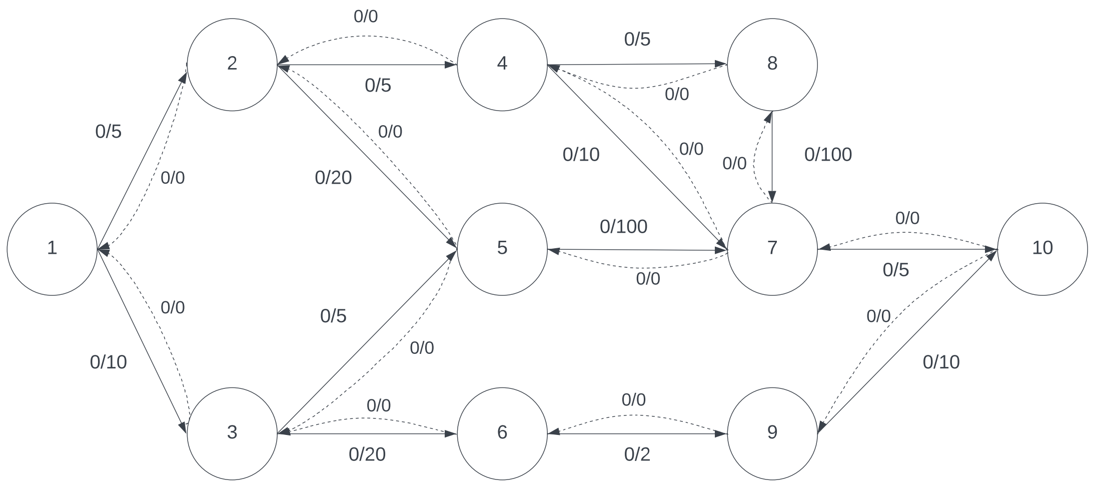

Figure 1: An Example of a Flow Network

2 The Edmonds-Karp Algorithm

Published in 1972 by Jack Edmonds and Richard Karp, this algorithm is used to find the maximum flow through a flow network, this algorithm is a more optimised implementation of the *Ford-Fulkerson* algorithm. Ford-Fulkerson [3] is a greedy maximum flow algorithm which leaves the method for finding augmenting paths unspecified. Augmenting paths are any paths through the residual network (from the source to the sink node) that more flow can be sent along. Therefore, the maximum flow through a flow network is found when no more augmenting paths can be found, at which point, the graph is said to be *saturated*.

Edmonds-Karp uses an unweighted breadth-first search to uncover the augmenting paths. Doing this guarantees that every augmenting path found is a shortest path with an available capacity. The logic behind this is that longer augmenting paths typically tend to have lower *bottleneck* values, where this is the maximum amount of flow that can be pushed through an augmenting path (the smallest edge in the path), and so, the overall runtime of the algorithm will be shorter as the graph can be saturated quicker. The steps for this algorithm are as follows:

1. While there exist augmenting paths:
    (a) Use an un-weighted breadth first search to find an augmenting path of shortest length.
    (b) Find the bottleneck value of the augmenting path.
    (c) Update the capacity values for every edge in the augmenting path
2. Calculate the maximum flow through the graph by summing the amount of flow entering the sink node.

2.1 Pseudo Code and Flowchart

Algorithm 1 Unweighted Breadth First Search

```
procedure BFS($G, S, T$)                ▷ G is the graph, S and T are the source and sink node
    queue = queue()
    visited = list()
    previousNodes = array(G.length())   ▷ Stores the node used to get to the current node

    while not empty(queue) and previousNode[T] = null do
        currentNode = queue.pop()

        for edge in G[currentNode] do
                    ▷ Visit a node if it has a remaining capacity and hasn't already been visited
            if edge.remainingCapacity > 0 and visited[edge.toNode] = false then
                visited[edge.toNode] = true
                previousNodes[edge.toNode] = edge
                queue.enqueue(edge.toNode)
            end if
        end for
    end while
    return previousNodes
end procedure
```

Algorithm 2 Edmonds-Karp Algorithm

```
procedure EdmondsKarp($G, S, T$)    ▷ G is the graph, S and T are the source and sink node
    Initialise $G_R$
    flow = 0
    previousNodes = array(G.length())    ▷ Stores the node used to get to a given node

    while not(previousNodes[T] = null) do
        previousNodes = BFS(G, S, T)

        if previousNodes[T] = null then
            return flow
        end if

        bottleneck = 0    ▷ Calculate the amount of flow in the augmenting path found
        edge = previousNode[T]
        while not(edge = previousNode[S]) do
            bottleneck = min(bottleneck, edge.capacity - edge.flow)
            edge = previousNode[edge]
        end while
                          ▷ Update the capacity values for each edge in the augmenting path
        edge = previousNode[T]
        while not(edge = previousNode[S]) do
            edge.flow = edge.flow + bottleneck
            edge.residual.flow = edge.residual.flow - bottleneck
            edge = previousNode[edge]
        end while
        flow = flow + bottleneck
    end while
    return flow
end procedure
```

3 Complexity Analysis

Below, V refers to the number of vertices (nodes) in the graph and E to the number of edges. This algorithm has the following characteristics:

- Runtime complexity of $O(VE^2)$.
- Space complexity of $O(E + V)$. The space complexity of the BFS used is $O(V)$ (in the worst case all nodes must be stored) and for the other part of the algorithm it is $O(E)$ (in the worst case all edges must be stored).
- Optimality: The maximum flow through the flow network is always returned.
- Completeness: Each augmenting path found is the shortest one, so the length of the augmenting paths remains non-decreasing and the algorithm is guaranteed to terminate.

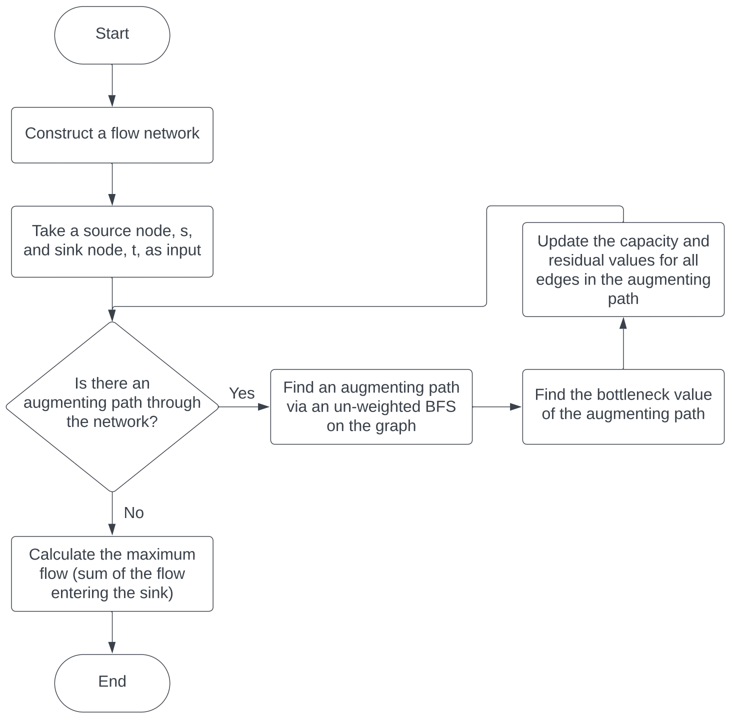
Start
Construct a flow network
Take a source node, s, and sink node, t, as input
Is there an augmenting path through the network?
Yes
Find an augmenting path via an un-weighted BFS on the graph
Find the bottleneck value of the augmenting path
Update the capacity and residual values for all edges in the augmenting path
No
Calculate the maximum flow (sum of the flow entering the sink)
End

Figure 2: A Flowchart of the Edmonds-Karp Algorithm

When compared to the standard Ford-Fulkerson method for finding the maximum flow, which has a time complexity of $O(Ef)$, where f is the maximum flow through the network, this algorithm achieves a much better time complexity. Even though it may not appear significantly more efficient, it is widely considered better as the time complexity is no longer dependent on the flow through the network. The slowest the flow value in a graph could be updated would be a value of one per iteration, for graphs with large flow values, having a time complexity dependent on the graph's flow can make the algorithm's execution time skyrocket. Additionally, as Edmonds-Karp is not dependent on the flow of the graph, its time complexity is said to be *strongly polynomial*, which is a tighter asymptotic constraint.

The proof for why this algorithm has a time complexity of $O(VE^2)$ relies on two statements. The first is that the length of the shortest path from the source to all other nodes in the residual graph increases *monotonically*, meaning that it always increases. This is easily proved, as breadth first search (BFS) runs in $O(E)$ time by always finding the shortest path. The second is that the total number of flow augmentations is $O(VE)$, which is proved by showing that *critical* edges (edges which have no remaining capacity after an augmentation) disappear from the residual graph after flow augmentation. Since a residual graph contains $O(E)$ edges and each edge can be critical $O(V)$ times, the total number of flow augmentations must be $O(VE)$. Using a BFS runs the total number of flow augmentations E times, resulting in the stated time complexity.[4]

4 Limitations and Improvements

Edmonds-Karp suffers from two key problems, the first is that while its time complexity is independent of the flow, f_{max}, this flow can occasionally be smaller than $V * E$. However, this is rarely the case, especially in practice with larger flow networks. The second, is that the algorithm is outdated, naturally, since its inception in 1972, new methods and techniques for further optimising the process of finding the maximum flow have been found.

An example of this is the *push-relabel* maximum flow algorithm, which utilises a maintained *preflow*, gradually converting this preflow into the maximum flow by transferring flow locally (amongst nodes) using *push* operations which are guided by *relabel* operations [5]. More modern algorithms, some from as recent as 2022 [6], have used more advanced techniques such as dynamic graph data structures to reduce the time complexity down to almost linear time.

Another improved algorithm of note is *Dinic's* algorithm, which is also an implementation of Ford-Fulkerson where the search order is defined via a breadth first search. This algorithm was published independently, two years prior to Edmonds-Karp, but uses the ideas of *blocking flow* and the *level graph* to achieve a better time complexity of $O(V^2E)$ (better as V is often $<$ E).

5 Applications

Much akin to other algorithms that calculate maximal flow in a flow network, this can be applied to any optimisation problem that uses or can be reduced to flow network.

A physical example, outside of the Internet of Things, is that of optimising utility networks, i.e. water plumbing or electrical grids This is done by determining the volume of the utility that can be passed through the network at a given time without overloading it. In the case of a water plumbing network, houses or outlets could be seen as the nodes and the pipes connecting them the edges. The flow through the network is then the volume of water than can be passed through the network from a starting water distribution centre to a given outlet, with augmenting paths representing different paths from the source to sink. This can prevent water buildup, prevent water from running out and be used to analyse any bottlenecks in the network.

Another example is that of airline scheduling [7] (or any scheduling for that matter, e.g that found in distributed computing). Airlines everyday have to produce optimal schedules for thousands of routes in order to maximise profits, equipment and employee allocation, minimise fuel costs, etc. Representing flight schedules as a flow network where every flight depends on the origin and destination airport, in addition to the time of departure and arrival of other flights, allows for algorithms such as Edmonds-Karp to be applied to the schedule. This finds the optimal schedule for the airline, identifying which sequence of flights leads to more flights being reachable (one plane used for multiple trips), enabling more flights to be taken and identifying which flights are likely to hinder, or cause problems with, the schedule.

6 Conclusion

Providing a mere glimpse into the wide world of flow networks, graph theory and maximum flow algorithms, this paper introduces a basic set of definitions and ideas that are central to understanding the Edmonds-Karp algorithm. The motivations and principles for the algorithm are then outlined, with a complexity analysis, explanation of the limitations, possible improvements and two applications of the algorithm being explored. Edmonds-Karp is an efficient maximum flow algorithm with an astounding range of applications that are ever more prevalent in the increasingly connected world we inhabit, despite it not being as efficient as contemporary max flow algorithms, it remains a historic piece of theory that propelled the boundaries of network-flow optimisation.

References

[1] R. J. Wilson, *Introduction to Graph Theory.* USA: John Wiley & Sons, Inc., 1986, ISBN: 0470206160. DOI: 10.5555/22577.

[2] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM (JACM)*, vol. 19, no. 2, pp. 248–264, 1972.

[3] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956. DOI: 10.4153/CJM-1956-045-5.

[4] S. D. Erik Demaine and N. Lynch, *Network Flow and Matching.* MIT, 2015. DOI: 6.046J/18.410J.

[5] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, no. 4, pp. 921–940, 1988, ISSN: 0004-5411. DOI: 10.1145/48014.61051.

[6] L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, and S. Sachdeva, "Maximum flow and minimum-cost flow in almost-linear time," in *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, 2022, pp. 612–623. DOI: 10.1109/FOCS54457.2022.00064.

[7] T. M. Murali, *Applications of Network Flow.* Virginia Tech, 2014.