Applications of Graph Theory in Networks

Alexei Pelyushenko

3/22/2023

Abstract

The purpose of this report is to talk about calculating the maximum flow rate of a network. Several principles and algorithms will be discussed. Ideally the reader will learn how to calculate flow rates of a network for themselves and understand several ways of thinking about networks.

1 Introduction

Graph theory is a relatively new concept in the field of mathematics. For reference, geometry was invented in 3000 BC, almost 5000 years before graph theory. The idea was conceived when famous mathematician Leonhard Euler solved the Konigsberg problem in 1736, also known as the 7 bridges of Konigsberg, shown below. The problem goes, create a path which crosses every bridge once and exactly once, or a trail that crosses every bridge. On the right is essentially the same graph as what is shown on the left. In other words, from every point on both graphs you can reach the same islands. It was through Euler's solution that the early concepts of graph theory, or earlier known as *geometria situs* (the geometry of position) began.

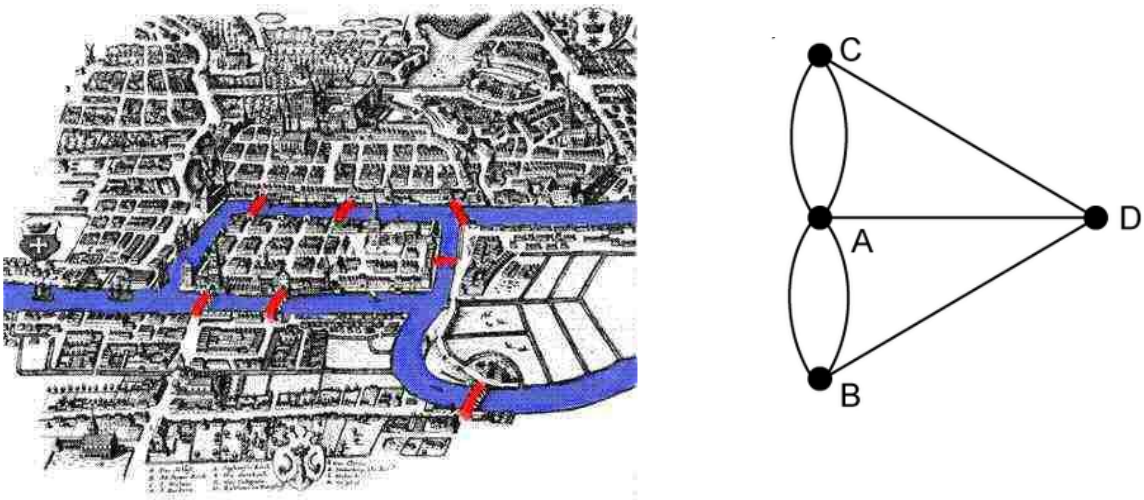

Figure 1.1 A picture depicting Konigsberg on the left and its topological equivalent in a modern graph representation on the right.

Shown above is Konigsberg and its 7 bridges: Blacksmith's bridge, Connecting bridge, Green bridge, Merchant's bridge, Wooden bridge, High bridge, and Honey bridge. Citizens of Konigsberg asked the question, is it possible to cross every bridge once and once only? No one in Konigsberg was able to solve it. It was Leonard Euler who gave a solution to the problem when prompted by Carl Leonhard Gottlieb Ehler in a letter. (Paoletti, Teo)

2 Graph Theory

2.1 Short Introduction into Graph Theory

Typically when we think about graphs we think about a Cartesian graph with (x,y) coordinates and functions. However, the definition of a graph is actually more general. Consider the example below.

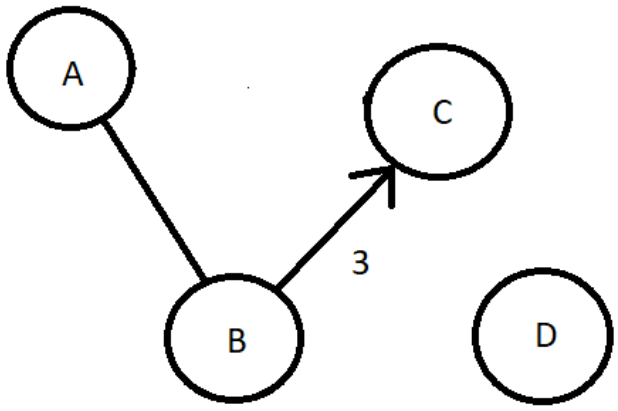

Figure 2.1.1: A simple graph

A graph is a collection of nodes--like the circles A, B, C, and D--and edges, lines connecting nodes. These edges represent connections between nodes. For example, a path exists between A and B since an edge between them exists. We call this edge AB. The notation is that the first letter is the starting point and the second letter is the ending point. We also have directed edges, the edges with arrows on them. These are one way connections. The edges AB and BA exist and are the same edge, but the edge AC exists and CA does not. These edges may also have what's called a weight, a number correlated with the edge. For example, BC has weight 3. If this were a map of 4 cities, we might say that the road between cities B and C is 3 miles long. Also, a node does not necessarily have to be connected to any other nodes, like node D.

We say that there exists a path between two nodes if you can follow some series of edges to get from one node to another. For example, a path exists from A to C since we can follow edges AB and then BC. However, no path exists from C to A because we can't follow the edge BC from C to B.

2.2 Networks

Suppose you're an internet service provider. When it comes to providing an internet service, there are 4 main characteristics you must bring to your service: fault tolerance, scalability, quality of service, and security. We will focus on quality of service, for example, from Internet service providers (ISPs). The amount of service for an ISP is how high the bit rates are, or in other words data per second. Figure 2.2.1 shows a simplified network. User personal computers (PCs) connect to switches, local access network devices that allow local computers to communicate with each other. PC3 and PC2 send their data to Switch1. PC0 and PC1 send their data to Switch0. Switch1 also sends its data to Switch0. Finally, Switch0 sends its information to Router0, which allows this local access network to reach the Internet. The diagram below is essentially what the ISP pipeline looks like at the end.

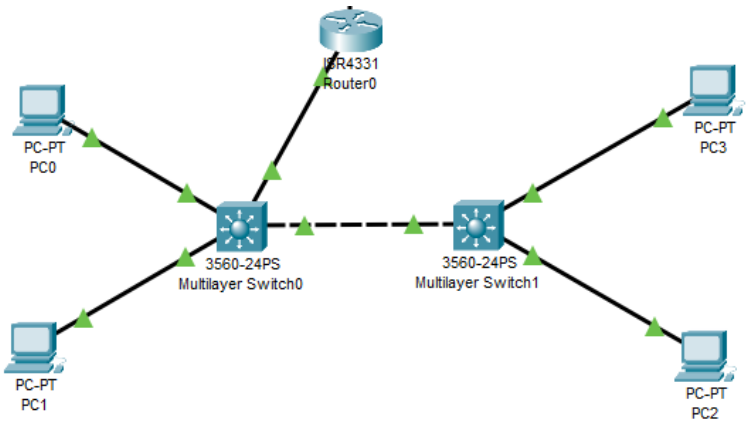

Figure 2.1.1: Sample network

2.2 Problem statement

Let us consider a scenario in a corporate network with a very sensitive production line. The production department requires constant updates from the testing department to ensure that products stay top quality. However, these constant updates also necessitate large amounts of data. The company has an internal network. Fortunately for the worker sending testing information, his boss says they can use any and all devices they want. Each individual device has its own specifications on how much data per second it can deliver. The worker needs to figure out how much data is possible to send from the testing department to the production department and the optimal routes through the network the data should go. Any one optimal route suffices as long as it provides the maximum network flow.

3 Max Flow Ford-Fulkerson Method (FF)

Calculating the maximum flow of a network refers to what's called the max-flow min-cut problem, and one way one can go about solving it is the Ford-Fulkerson method. The source node will be the source of all of our network data, it is able to output hypothetically infinite amounts of data. The sink node *t* can hypothetically receive infinite amounts of data. We need to determine how much data we can send per second from source node *s* to sink node *t*. The lettered nodes will represent different internet related devices, such as the switches and routers. Edges will represent connections between nodes--either sink nodes, source nodes, or devices like A, B, C, and D in the diagram-- and their respective transmission rates. For example, the connection AB has the transmission rate 4. Data can only travel from A to B since the edge indicates a direction using an arrow.

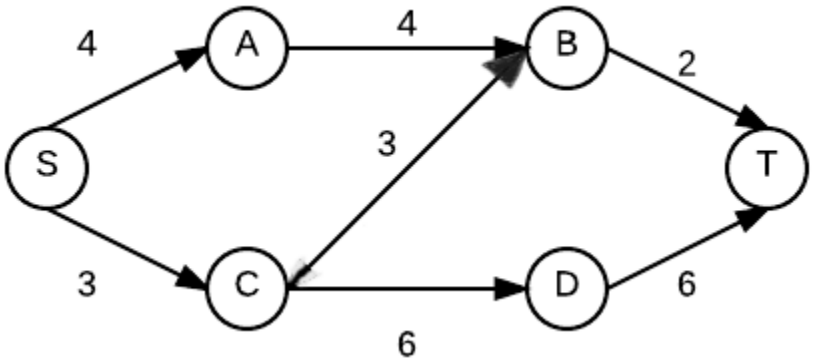

Figure 3.0.1 A directed graph with six nodes

Finally, we need to define the augmenting path. An augmenting path is a path we choose which can send a unit/s of flow equal to the smallest weight of the edges contained in the path, or transmission rate in this case.

3.1 The method

The method--not necessarily algorithm, since this is not specific enough to be an algorithm--is as follows: Select an augmenting path, so any path from *s* to *t*. Record the smallest transmission rate in the path as *b*, which we will refer to as the bottleneck value of the path, then decrease each edge along that path by that value. We will then add this bottleneck value to our eventual answer, *f*, which starts at 0. Let's choose the augmenting path (SC, CB, BT) first. In this notation, each value in the path represents a connection. For example, SC is the connection from node S to C, CB is the connection from C to B, and BT is the connection from B to T. The bottleneck value is 2, so change SC's transmission rate to 3-2=1, CB's to 3-2=1, and BT's to 2-2=0. We will then also draw new residual edges opposite to those we just altered. For the edges we just altered, we create new directed edges going in the opposite direction. So we create a new edge TB and set its transmission rate to 2. A new directed edge BC's rate becomes 2, and CS becomes 2. Then add 2 to *f*. *f* becomes 0+2=2. In the process, creating the

new residual graph shown below. This step of using an augmenting path to recalculate edges in the graph is called *augmenting*.

Rinse and repeat until it is impossible to pick any augmenting paths from *s* to *t* with minimum flow above 0. Selected augmenting paths must never contain edges with 0 or negative values for capacity. As we progress further and further along the method, *f* will eventually become the maximum flow of our graph. By applying these steps, we find that Figure 3.1.1 will result in a maximum flow of 5 (from 2+2+1).

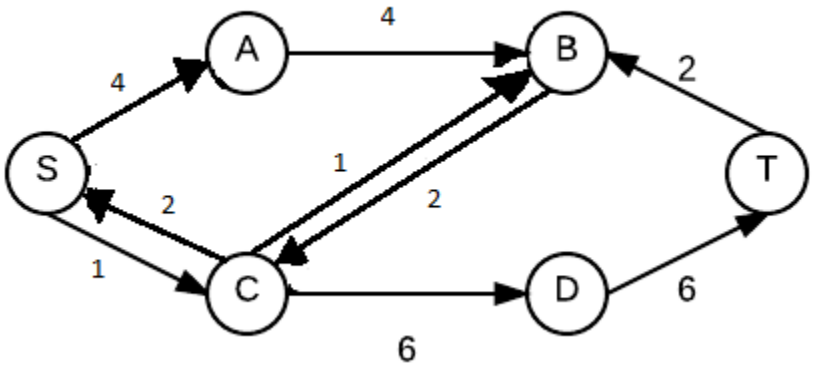

Figure 3.1.1: The residual graph after applying an augmenting path

The question now is, “What’s the point of the residual edges?” When choosing augmenting paths for more complicated networks, one may make a mistake and choose a path that wouldn’t result in the most efficient path. The purpose of these residual edges is to allow one to remake paths. For example, by examination, the path that was selected first was not actually an optimal path. What we should’ve chosen was (SC, CD, DT) since that uses a path only the edge SC can use. This is fine however. Using the residual edges, we will still be able to produce and apply more augmenting paths. All the method cares about is that, if there are still augmenting paths--paths that exist from *s* to *t*--then continue the method. We don’t change what we’re doing because we made a poor choice. We simply continue with the original method as though it wasn’t a poor choice.

From here, we repeat step 1, choose another augmenting path. We are allowed to use residual edges. Previously, BC never existed, but now because of the previous augmenting path it does now. Thus we are allowed to create a new augmenting path (SA, AB, BC, CD, DT) and what happens is this method will still work and allow us to get the maximum flow anyway, applying all of the previous procedures. For reference, if we had chosen the best paths to begin with, we would’ve received the same maximum flow with 2+3 instead of 2+2+1.

2+2+1: (SC, CB, BT), (SA, AB, BC, CD, DT), (SC, CD, DT)
2+3 : (SA, AB, BT), (SC, CD, DT)

The order in which these paths occur generally doesn’t matter. As long as valid augmenting paths exist and are applied, paths with bottleneck value that is strictly positive, the method will work. In other words, you don’t have to choose the augmenting path with the

maximum bottleneck value every time. You can also select augmenting paths with flow 1 each time, if one so desires.

3.2 Useful consequences of finding the solution

Now we have determined the maximum amount of data that can be sent at any given point in time from the source node to the sink. The worker knows exactly how much information they can send to the production line in peak conditions at all times. This problem can be generalized to many other problems as well. An ISP can determine how much data coverage they can provide to an area. A government can know how much traffic roads can handle. Airports can know how many planes they can and should operate at any given time. This maximum flow value proves to be incredibly valuable in many different facets of life. This includes the minimum cut values that one also receives as a byproduct of solving this problem. To clarify, the equivalent question when we are solving for the minimum cut problem would be, “what is the minimum amount of connections we can cut, or disconnect, from the network in order to stop all flow?” Now people can know exactly which and how many parts of a network are critical to a network.

Finally, the graph created as a result of solving the max-flow min-cut problem shows exactly how the resources are allocated in creating maximum flow for the network. Subsequently, the network can be optimized by rearranging the resources being used.

3.3 Complications

When implementing the Max Flow Ford Fulkerson method as a computer program, the details of how exactly one goes about coding this will matter. For example, when we are choosing the augmenting path in the first step we could just choose any augmenting path. Like how we chose the path (SC, CB, BT) instead of something like (SC, CD, DT). There was no definition for one other than “a path from source node *s* to sink node *t*”. The algorithm will work anyway assuming it is coded correctly. However, we find that with certain graphs, we can get extremely inefficient results if we implement the method haphazardly. Consider the following graphs below.

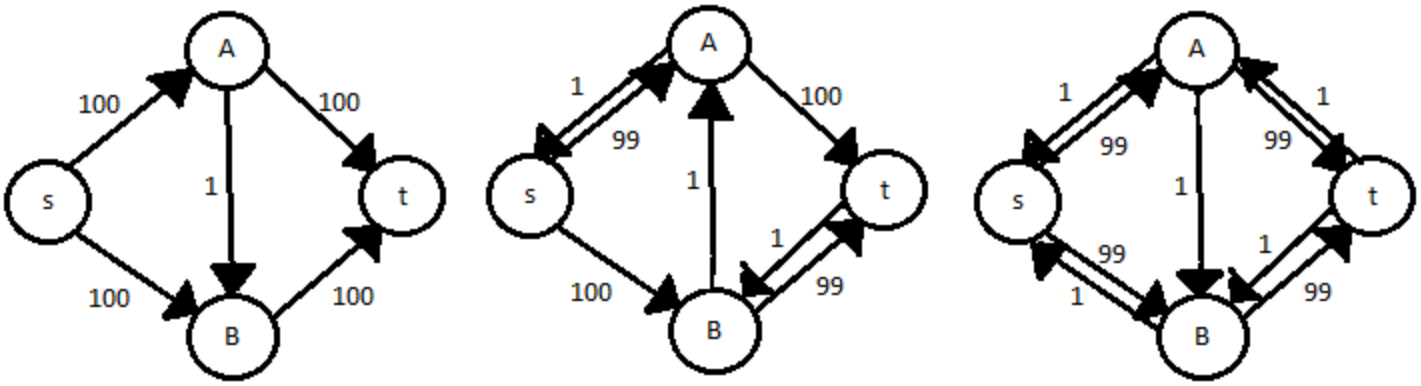

Figure 3.3.1: The consequences of poor augmenting path choices

In this example, we choose augmenting paths (SA, AB, BT) and then (SB, BA, AT). If we are careless with our algorithm we can end up making a 200 step process of creating sequences that add 1 flow to the result over and over again until the method is complete. To the human eye, this is a ludicrous solution. This is incredibly inefficient and adds tremendous time to the time complexity of the algorithm. In fact, this particular solution took 100 times as long to finish than the best solution. Unfortunately, computers do not see things the way we do, and will simply perform whatever it was asked, like a malevolent genie. This precise worst case scenario is the reason why the Ford Fulkerson has time complexity O(E * f)--time complexity refers to how long this algorithm will take in the worst case scenario--E times f, where f refers to the maximum flow in the graph and E the number of edges in the graph. We will discuss more optimal algorithms using Ford-Fulkerson (FF) in section 4 shortly.

3.4 Max-Flow Min-Cut Theorem

Now that we've discussed the Ford Fulkerson method, we are now able to use the algorithm and calculate the maximum flow of any network. In doing so, we also calculate the minimum cuts required to halt a network as well. If we revisit the solved graph for figure 3.0.1, we see that there is no longer a path leading to the sink. Through this algorithm, we "cut" all connections to the output. This is a consequence of the max-flow min-cut theorem on which this method is based. The theorem itself is related to the duality principle, that optimization problems can be viewed through either their primal problem or their dual problem.

> "The **max-flow min-cut theorem** states that the maximum flow through any network from a given source to a given sink is exactly equal to the minimum sum of a cut. This theorem can be verified using the Ford-Fulkerson algorithm. This algorithm finds the maximum flow of a network or graph." (Datta)

In other words, we can also think about solving the max-flow problem in terms of cutting the minimum amount of weight necessary from the graph to disconnect the sink from the source. Everytime an augmenting path is applied to the graph, we "cut" out a portion of available flow that could have been used in the network. We keep applying these augmenting path cuts

until no more cuts are possible, then we add the available flows to our running count to get the max-flow.

4 Improvements

Picking up where we left off in section 3.4, we can apply several improvements to the FF method. The method provides a baseline algorithm for calculating the max-flow of a network. However, its usage does not translate nicely into the computer world. In this section we will improve on the algorithm.

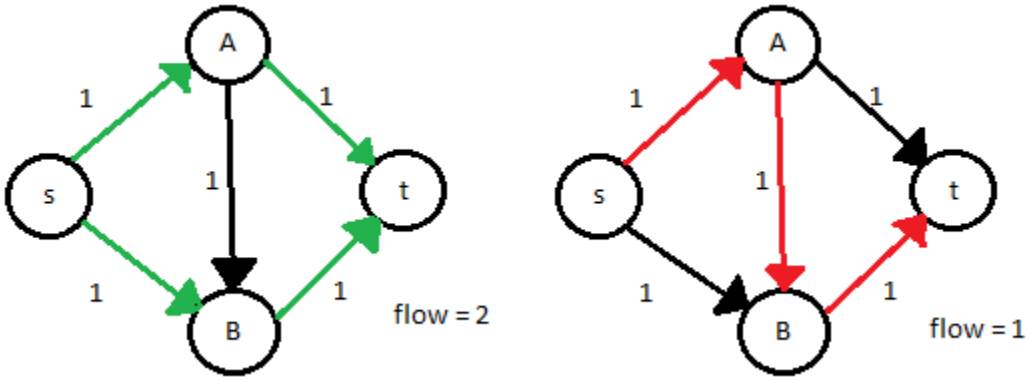

Figure 4.0.1: The consequence of not including the residual graph when performing the FF method. Without a residual graph, you cannot fix mistakes as easily, and a computer not at all without taking edge-cases into account.

First things first, starting with our Ford Fulkerson method, how should we choose the augmenting path? We shouldn't choose randomly based on the observation of the worst-case scenario. We should make the fewest, biggest cuts possible. The more we cut from the network, the faster we will get to the maximum flow number.

4.1 Depth first search and Breadth first search

From here on, we will be discussing algorithm optimizations. As such, we should know about what the algorithm uses first. By default, the implementation of the Ford-Fulkerson method as a computer program uses depth first search. The purpose here however is not to completely understand what depth first search or breadth first search (BFS) is. The point is to understand what they generally do.

Depth first search:

1. Start from a node in a graph G. From this node, select one edge to follow.
2. If from this node there are no edges to take other than an edge that has already been traversed, end this path.
3. Otherwise, select another random edge to follow that hasn't already been traversed.
4. Continue until a path is found to the sink

Breadth first search:

1. Start from a node in graph G. In a list, record the names of every node that is adjacent (there is a directed edge from the current node leading to the next node) to the starting node.
2. For each of these nodes, record in the list all nodes adjacent to the currently listed nodes. Do not include any nodes that have already been traversed.
3. Repeat step 2 until a path is found to the sink.

4.2 Edmonds-Karp Algorithm

As a baseline, the standard FF algorithm uses depth first search (DFS). A new augmenting path must be chosen every step, so a DFS is perfect for guaranteeing one. It won't necessarily provide the shortest or biggest one however. From here we will compare with another algorithm that provides advantages over the base FF algorithm.

One improvement that can be made to the FF method is the Edmonds-Karp algorithm. Where the typical FF algorithm uses DFS search to find random augmenting paths, Edmonds-Karp uses BFS, breadth first search, allowing us to find the shortest possible augmenting path from the source to the sink. The main difference between E-K and FF is that E-K switches the path finding method from DFS to BFS. As well as that, if two or more paths are found that have the same path length to the sink, E-K will choose the path with a larger bottleneck value.

E-K offers a few advantages over FF. Namely, E-K will usually be faster. E-K is guaranteed to pick the shortest path every iteration, whereas FF in a worst case scenario can pick a path as long as the number of vertices in the graph. This is a major improvement over FF. The longer a path is, the greater the chance that its bottleneck value will be lower since it will be the lowest of all the edges. Not to mention that it simply takes longer to calculate longer paths. Choosing the biggest path in tiebreakers will prevent situations like figure 3.3.1.

The time complexity of Edmonds-Karp is O(VE^2), the size of the vertex set V times the square of size of the edge set E. This looks like it's slightly worse than the time complexity of FF which is O($E * f$). However, this is not the case due to the reasons up above. Not being dependent on the maximum flow of the graph f allows one to use this algorithm for all max-flow min-cut problems regardless of how big the input values, the transmission rates in the edges, are. This was a major improvement in its time.

5 The Field of Networks

In conclusion, graph theory can be very valuable in applications. Be it internet flow, air traffic, package delivery, and more. Especially so when we apply it to networks which involve a starting point and end point. In the way we can calculate the maximum flow of a network, we can also think of it as cutting off the minimum amount of weight to completely block its flow. While Ford-Fulkerson is the foundation of this maximum flow problem, Edmonds-Karp improves on it by optimizing the way in which augmenting paths are chosen.

Despite the strengths of each of these algorithms, they have their drawbacks. For instance, Ford-Fulkerson is susceptible to inefficient choices and long convergence times, referring to completing the algorithm. Meanwhile Edmonds-Karp can have issues with memory concerns due to the nature of BFS. Nevertheless, the practical applications of these algorithms have demonstrated their usefulness in solving real-world problems, and will remain as significant contributions to the field of network flow. With more research, we can build on these algorithms and realize greater, more effective solutions.

Bibliography

Datta, Written by: Subham. “Minimum Cut on a Graph Using a Maximum Flow Algorithm.” *Baeldung on Computer Science*, 11 Nov. 2022, https://www.baeldung.com/cs/minimum-cut-graphs.

Teo Paoletti (The College of New Jersey), "Leonard Euler's Solution to the Konigsberg Bridge Problem," Convergence (May 2011)