Applications of a Directed Graph (Network-flow) to Find the Maximum Flow

Md Mahedi Hasan[1]
[1]WSU

December 16, 2021

Abstract

This paper demonstrated the application of a network flow algorithm to find the maximum flow of the directed graph. The conceptual details are provided with two applications. In the first application, the Ford-Fulkerson algorithm is used to estimate the maximum flow of a directed graph. And in the second application, the concept of linear programming is explained with mathematical details to find the maximum flow of another network problem.

Keywords: network-flow, max flow, Ford-Fulkerson, graph theory, linear optimization

1 Introduction

In graph theory, real-world problems are transferred (converted) into a graphical problem and solved in myriad instances. The Network flow is one of the vastly used approaches for solving problems using the graphical approach in a network. In this paper, we have discussed the concepts of flow-network and the different types of flow-networks and their approaches to solving real-world problems. After concept building, two applications are shown to get a practical understanding of the concept. This paper has not gone through the heavy details of mathematics; instead, it tries to explain the concept and show the result of the application.

2 Network Flow

In graph theory, network flow or flow-network is a directed graph with numerical capacities on its edges, and in this graph, the objective is to construct a flow [1]. Often in operations research, a directed graph is called a network, the vertices are called nodes, and the edges are called arcs. The numerical values on each edge represent the capacity constraints [2]. The amount of flow on edge cannot

exceed the edge's capacity. It follows the principle of "incoming flow" equals to the "outgoing flow" at all vertices, except for certain designated exits. A network can be used to model traffic in a computer network, circulation with demands, fluids in pipes, currents in an electrical circuit, or anything similar in which something travels through a network of nodes.

2.1 Type of network flows

There are different network flow problems, such as maximum flow problems, minimum-cost flow problems, nowhere-zero, etc. Different types of network flow serve different objectives. In a max-flow network, the goal is to maximize the total amount of flow from the source (s) to sink (t) [3]. Whereas the mincost-flow network, the edges have costs associated and capacities. The goal is to achieve a desired amount of flow (or a maximum flow) with a minimum possible cost. On the other hand, the nowhere-zero network is a type of flow where the flow amounts are restricted to a finite set of nonzero values.

2.2 Algorithms for constructing the flows

There are different algorithms available for constructing the flow-network. The common algorithms are "Dinic's algorithm", "Edmonds-Karp algorithm", "Ford-Fulkerson algoithm", "Network simplex algorithm" etc. For the purpose of this study, we have used the Ford-Fulkerson algorithm. So, details on this algorith is given here.

Ford-Fulkerson Algorithm

The Ford-Fulkerson method continues finding augmenting paths and augments the flow until no more augmenting paths from source (s) to sink (t) exist. We want to use the Ford-Fulkerson method to find the maximum flow for the directed graph [4]. By definition, the sum of the bottlenecks found in each augmenting path is equal to the max-flow. The following steps are used in a Ford-Fulkerson algorithm.

Step 1 We start with initial flow as 0.

Step 2 If there is a augmenting path from source to sink, we should add this path-flow to flow.

Step 3 Retund the flow

Time complexity of Ford-Fulkerson algorithm is O(maxFlow * E). We run a loop while there is an augmenting path. In worst case, we may add 1 unit flow in every iteration. Therefore the time complexity becomes O(maxFlow * E) [5].

Max-flow Min-cut Theorem

The max-flow min-cut theorem states that in a flow network, the amount of maximum flow is equal to the capacity of the minimum cut [6]. Using the Ford-Fulkerson algorithm, we get the capacity of the minimum cut. Here are the steps to follow:

Step 1 Run Ford-Fulkerson algorithm to print all edges of the minimum cut

Step 2 Find the set of vertices that are reachable from teh source in the residual graph.

Step 2 All edges which are from a reachable vertex to non-reachable vertex are minimum cut deges.

Finding minimum s-t cut in a flow netwrok

In a flow netwrok, an s-t cut is a cut that requires the source (s) and the sink (t) to be in different subsets. This consists of edges going from the source's side to the sink's side. The capacity of an s-t cut is defined by the sum of the capacity of each edge in the cut-set[7]. This concepts are inherently used in the Ford-Fulkerson algorithm to find the maximum flow.

3 Maximum Flow Algorithm

Here, we will discuss the details of a max flow algorithm with how it works. The max-flow algorithm starts with a flow graph. A flow graph (flow-network) is a directed graph where each edge has a certain amount of flow. The flow running through an edge must be less than or equal to the capacity. This intuitively makes sense because if we allow more flow than what the capacity permits, it means there is a chance of damage or something has to go wrong. Each edge has a certain flow and capacity specified by the fraction adjacent to each edge in the graph. Initially, the flow through each edge is 0, and the capacity is a nonnegative value. Now, to find the maximum flow (and min-cut as a by-product), the Ford-Fulkerson method can be used and this finds the augmenting paths through the residual graph and augments the flow until no more augmenting paths can be found. Here, an augmenting path is a path of edges in the residual graph with an unused capacity greater than zero from the source (s) to the sink(t).

Augmenting the Path

The key thing in augmenting paths is that it can augment only through the paths which are not yet saturated. And this is one hint to realize if we have achieved the maximum flow. When there are no more paths to augment, that implies we have reached the maximum flow. There is another important concept called "bottleneck" that we need to clarify. In the augmenting path, the bottleneck

is the smallest edge on the path. We can use the bottleneck value to augment the flow along the path. Augmenting the flow simply means updating the flow values of the edges along the augmenting paths. For the forward edges, this means increasing the flow by the bottleneck value. When augmenting the flow along the augmenting path, we also need to decrease the flow along each residual edge by the bottleneck value. Residual edges exist to "undo" bad augmenting paths which do not lead to a maximum flow.

Uses of Max-flow algorithm

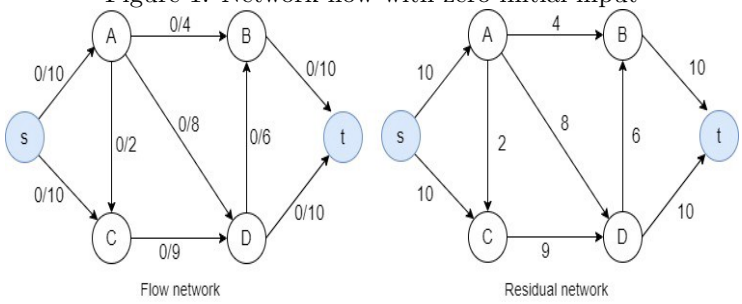
The max-flow algorithm can be used in numerous applications, where edges and nodes can represent any number of things. For example, the edges are roads with cars, pipes with water, or wires with electric current. Flow represents the volume of water allowed to flow through the pipes, the number of cars the roads can sustain in traffic, and net electric current. The maximum flow is the "bottleneck" value for the amount of flow that can pass through the network from the source to the sink under certain conditions.

4 Application1: Finding max-flow by Ford-Fulkerson method

Here in the first application, we have shown that how a max flow algorithm finds the maximum flow of a network. As we know, finding the max-flow begins with finding a flow graph where edges have a certain maximum capacity that cannot be exceeded. And the edges also have a flow value which is how many units of flow are passing through that edge. Initially, the flow is zero for all edges everywhere until we run a max-flow algorithm on it. And also, we have the source (s) and sink (t) nodes in the graph. Now we want to know with an infinite input source how much "flow" can we push through the network given that each edge has a certain capacity.

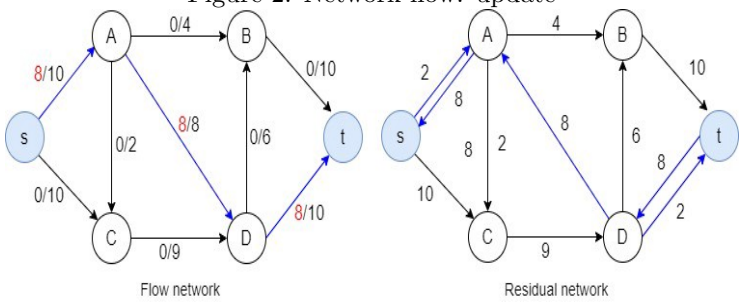
Let us take an arbitrary example and see how we can find the maximum flow using the Ford-Fulkerson method. First, we have a directed graph with nodes, A, B, C, and D, and of course, the source (s) and the sink (t) nodes. Each edge represents a certain capacity to flow and initially, we set the flow of every edge to zero.

Figure 1: Network flow with zero initial input



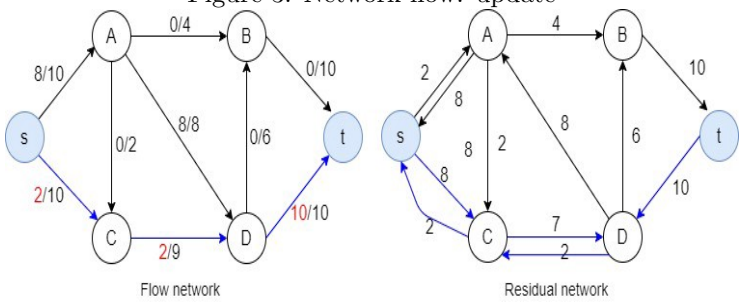
Now, we need to find an augmenting path in the residual network. We can start with any path. For instance, let us consider the path $s \to A \to D \to t$. Now, we want to determine the bottleneck capacity, i.e, the maximum capacity of flow for this path. The bottleneck capacity for this path is 8. In this step, we have the capacity constraint for the edges and without violating the capacity constraint, we can update the values of flow on the edges in the augmenting path. This will give us the following residual and flow network.

Figure 2: Network-flow: update



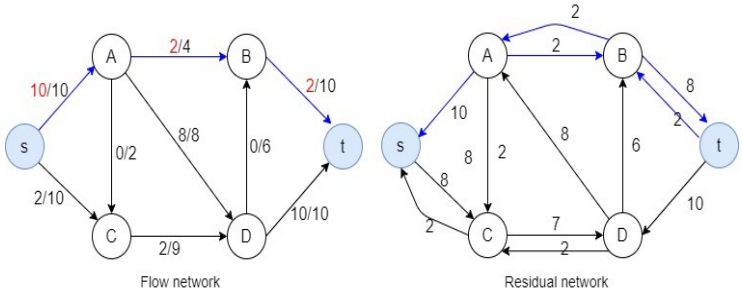
Now, we select another augmenting path, say, $s \to C \to D \to t$. And the bottleneck value is 2.

Figure 3: Network-flow: update



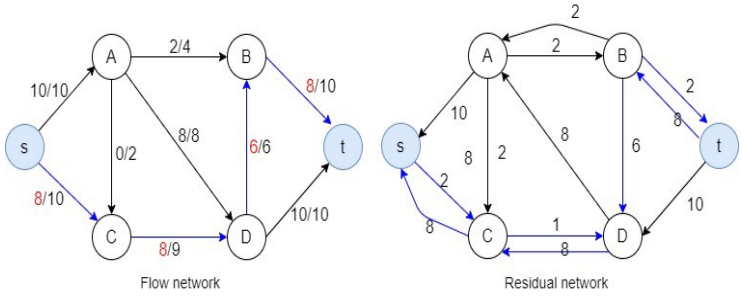
Still we have paths for augmentation in the flow-network. Let select the path $s \to A \to B \to t$ and the bottleneck value is 2.

Figure 4: Network-flow: update



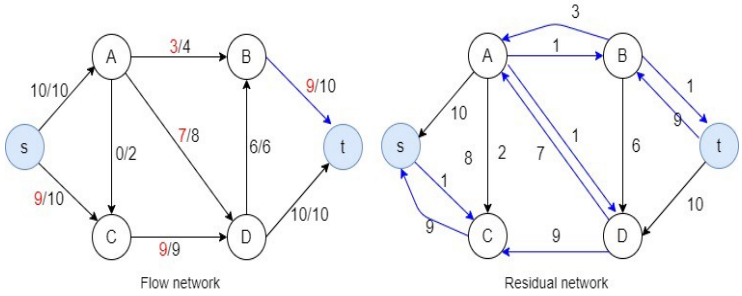
We have to augment further, and select the path $s \to C \to D \to B \to t$, and the bottleneck value is 6.

Figure 5: Network-flow: update



Now, we have more path to augment, $s \to C \to D \to A \to B \to t$, and the bottleneck value is 1.

Figure 6: Network-flow: update



Now, there are no paths left from the source (s) to the sink (t) in the residual graph to update. Hence, this attained the maximum flow through the edged

following the capacity constraints. Thus, the Ford-Fulkerson completed the update of flow and we can get the maximum flow from the graph now.

We know that the maximum flow is equal to the flow coming out of the source, hence, the maximum flow for our case is 10+9 = 19.

5 Application2: Finding max-flow through linear optimization

Lets us think about the water distribution network of a country (say in the USA). The water distribution network is a system that consists of water pipes of different capacities that are connected to the different nodes. Here, a node can be the source of water, a city that needs water, or could that also be just some transfer nodes. Our task here is to transfer the maximum amount of water to the cities using this network.

For simplicity, we assume that we have only one source of water in our network and one sink (which is a city). So, specifically, we want to find the maximum amount of water which can be sent through the pipes with their given capacities from the water source to the city. Here, the water sent from the source to the sink (city) can be represented by the amount of water send between every pair of nodes. By definition, as we also have stated previously, the flow of water should comply with the following properties:

- The amount of flow in every pipe is less than its capacity which is called the capacity constraints
- At each of the nodes, incoming flow is equal to the outgoing flow, however; this is different for source and the sink nodes
- The outgoing flow should be same for the sources(s) to the incoming flow at the sink(t)

In order to solve this problem, first of all, we can convert this problem into a linear programming (LP) problem. With the help of LP we need to find the maximum flow from the source (s) to the sink(t). In general, suppose, $f_{u,v}$ represents the flow from the vertex u to v. Considering all the conditions for the flow, the LP problem can be set as [3]:

$$max \sum_{s,u} f_{s,u}$$

$$s.t. \sum_{u:\{u,v\}\epsilon E} f_{u,v} = \sum_{u:\{u,v\}\epsilon E} f_{v,u}$$

$$f_{u,v} \leq C_{u,v} \quad \forall (u,v)\ \epsilon\ \mathrm{E}$$

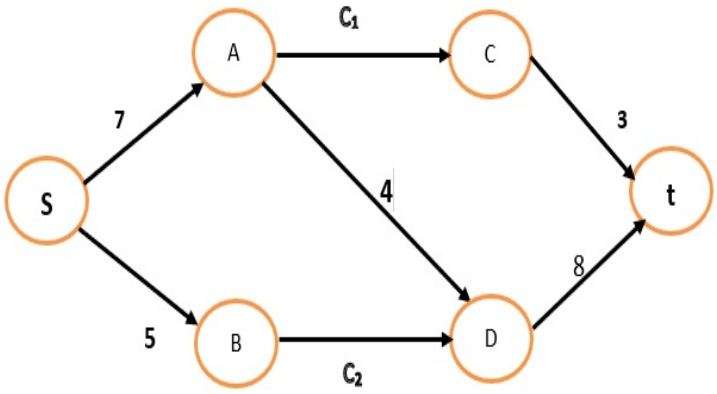
$$f_{u,v \geq 0}$$

Now we need to formulate an algorithm that can solve this LP problem. If we now provide an upper bound on the maximum flow using the capacities then it becomes.

$$max\ f_{s,u}$$
$$s.t. \sum_{u:\{u,v\}\epsilon E} f_{u,v} \leq \sum_{u:\{u,v\}\epsilon E} f_{v,u}$$
$$f_{u,v} \leq C_{u,v} \quad \forall (u,v)\ \epsilon\ \mathrm{E}$$
$$f_{u,v \geq 0}$$

There are different ways to solve the LP problem such as min-cut. We can adopt an approach to solve this LP problem to find the maximum water flow from the sourch to the sink. suppose a water treatment plant is designed to pump the maximum amount of water (millions gallosn per day) through the pipelines from the source node (s) to the sink node (t). The water supply network is showing in the follwing figure.

Figure 7: Max flow problem



Here from the node s to the node A, the capacity is 7 million gallons per day and the node s to the node B the capacity is 5 million gallons per day and so on.

Now we would like to formulate this maximum flow problem as a linear programming problem. First of all, we can define

$x_{i,j}$ = as the volume of water passess from i to j through the paipe per day

Here, we want to maximize the water flow throguh the arc $X_{t,s}$, so our objective function is

$$\text{max z} = X_{t,s} \quad or \quad (X_{s,A} + X_{s,B})$$

Now we also have the capacity constraint for the flows. The flow in each network should be greater than zero and less than or equal to the capacity. For an instance, for the edge s to A, the $X_{s,A}$ should be greater than or equal to zero but less than or equal to 7 and so on. Therefore the capacity constraints are:

$$\begin{aligned} s.t. : 0 \leq X_{s,A} \leq 7 \\ 0 \leq X_{s,B} \leq 5 \\ 0 \leq X_{A,C} \leq C_1 \\ 0 \leq X_{B,D} \leq C_2 \\ 0 \leq X_{A,D} \leq 4 \\ 0 \leq X_{C,t} \leq 3 \\ 0 \leq X_{D,t} \leq 8 \\ C_1 + C_2 < K \\ C_1, C_2 > 0 \\ 0 \leq X_{t,s} \end{aligned}$$

Now the node constraints are:

$$\begin{aligned} X_{t,s} &= X_{s,A} + X_{s,B} \\ X_{s,A} &= X_{A,C} + X_{A,D} \\ X_{s,B} &= X_{B,D} \\ X_{A,C} &= X_{C,t} \\ X_{A,D} + X_{B,D} &= X_{D,t} \\ X_{C,t} + X_{D,t} &= X_{t,s} \end{aligned}$$

This is a complete linear programming problem. Solving this linear programming problem in polynomial time, we would be able to solve the network flow problem and find the maximum flow from the source to the sink.

6 Implications and future direction

There is no doubt that the network-flow is a widely used approach with the directed graph across the disciplines. Specifically, we look at two of our applications; we would see that we can formulate different types of problems into a directed graph problem and solve them using different approaches. So, a proper formulation of a problem through the directed graph and solving them could lead us to the result we look for in our research.

7 Conclusion

This paper demonstrated the applications of a directed graph under a network flow algorithm. Conceptual detail on the network flow algorithm and the maximum flow network is explained. Then the Ford-Fulkerson method is used to

find the max flow of an arbitrary network. Besides, a concept of finding the maximum flow for a water distribution network based on a linear programming problem is also explained.

References

J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," in *Algorithmic foundations of robotics X*. Springer, 2013, pp. 157–173.

J. R. Evans and E. Minieka, *Optimization Algorithms for Networks and Graphs: Revised and Expanded*. CRC Press, 2017.

J. O. Royset and R. K. Wood, "Solving the bi-objective maximum-flow network-interdiction problem," *INFORMS Journal on Computing*, vol. 19, no. 2, pp. 175–184, 2007.

M. T. Kyi and L. L. Naing, "Application of ford-fulkerson algorithm to maximum flow in water distribution pipeline network," *International Journal of Scientific and Research Publications*, vol. 8, no. 12, pp. 306–310, 2018.

M. Mützell and M. Josefsson, "Max flow algorithms: Ford-fulkerson, edmond-karp, goldberg-tarjancomparison in regards to practical running time on different types of randomized flow networks." 2015.

G. Dantzig and D. R. Fulkerson, "On the max flow min cut theorem of networks," *Linear inequalities and related systems*, vol. 38, pp. 225–231, 2003.

S. Khot, G. Kindler, E. Mossel, and R. O'Donnell, "Optimal inapproximability results for max-cut and other 2-variable csps?" *SIAM Journal on Computing*, vol. 37, no. 1, pp. 319–357, 2007.