Finding Max Flow using both Ford-Fulkerson and Linear Programming

Tatum Rask

October 20, 2022

Contents

1 Introduction

These notes first provide a very short introduction to the language of network flows and finding maximum flow. Next, we briefly discuss the Ford-Fulkerson algorithm for finding maximum flow. We then apply the algorithm to an example: that is, we use it to maximize the wheat transported from St.Petersburg to Moscow. Next, we explain how to write a linear program to find maximum flow. Finally, we use Python to find the maximum flow of our previous St. Petersburg to Moscow example, verifying that the two methods used to solve the problem produce the same result.

2 Basics of Network Flows

In this section, we will briefly introduce network flows and the terminology we will use throughout this paper. This information is from class notes for Math 502: Combinatorics II at Colorado State University, taught by Alexander Hulpke [1].

A network is a directed graph (V, E) with weighted edges and two special vertices: the source vertex s and the target vertex t. For the purpose of this project, we will assume that the weights are non-negative and integer-valued.

Suppose we have labeled our vertices $V = \{i\}_{i \in I}$. Then, we may denote an edge by $e_{i,j}$, indicating that i is the the initial edge and j is the terminal edges.

It is helpful to think of the weights on edges as the maximum capacity of objects that can be transported across the edge. In this case, we sometimes call weights capacities.

Now, suppose we are interested in transporting objects from our source s to our target t: we formalize this by defining a flow. A flow is a function $f : E \to \mathbb{Z}$ such that:

1. The flow on a edge does nt exceed it's capacity: $0 \leq f(e) \leq c(e)$ for all $e \in E$.
2. The number of objects entering a vertex cannot exceed the number of objects leaving the vertex. That is, $\sum_{k:e_{k,i} \in V} f(e_{k,i}) = \sum_{j:e_{i,j} \in V} f(e_{i,j})$.

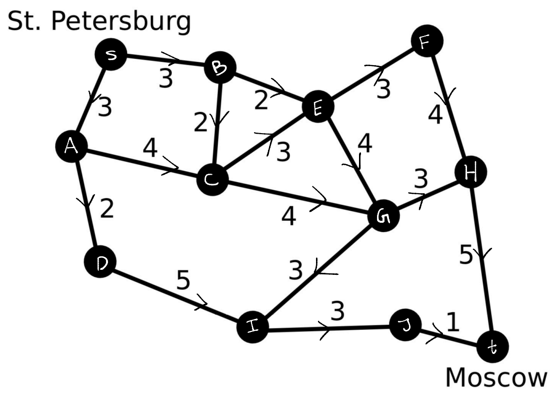
The value of a flow is defined as

$$\text{val}(f) = \sum_{j:e_{j,t} \in V} f(e_{j,t}) - \sum_{k:e_{t,k} \in V} f(e_{t,k}).$$

Indeed, we do need to subtract off any objects flowing out of the target, lest they be counted twice towards the value of the flow. In some examples (including the one we will consider), there will be no edges leading out of the target.

With the definition of a flow on a network, it is natural to ask whether we can find a *maximal* flow on a network. In the sections to come, we illustrate two methods for finding the max flow of a network. In both cases, we will illustrate max flow by focusing in on the following problem:

> Suppose we want to transport wheat from St. Petersburg to Moscow. There are various roads we can take, each allowing for us to transport a certain number of wagons filled with of wheat. We model this problem using the following network:



Noticed that I have (1) labeled the edges, and (2) simplified the problem by directing the edges according to the lexicographic ordering (with the exception of edges containing the source or the target).

3 Solving for Max Flow using Ford-Fulkerson Algorithm

In this section, we will briefly introduce the Ford-Fulkerson algorithm for finding the maximum flow of a network. I first learned this algorithm in Math 502: Combinatorics II, taught by Alexander Hulpke [1]. I augmented (pun intended) my understanding of the algorithm by watching the YouTube video "Ford-Fulkerson in 5 minutes" from Michael Sambol [2]).

The algorithm goes as follows:

1. Start with a flow. The flow with 0 on all edges is a realistic starting point.
2. Calculate the total flow.
3. Find an augmenting path. That is, find a path from the source s to target t with:
   - Non-full forward edges (so that we can increase flow).
   - Non-empty backward edges (so that we can redistribute flow).
4. If no such augmenting path exists, terminate. Maximum flow is as calculated in step 2.
5. If an augmenting path was found, find the *bottleneck capacity*. That is, find the maximum capacity we can add to every edge in the path.
   - If moving backwards along a path, we consider how much can be subtracted when finding bottleneck capacity.
6. Update the flow of each edge in the augmenting path accordingly.
7. Repeat steps 2-7.

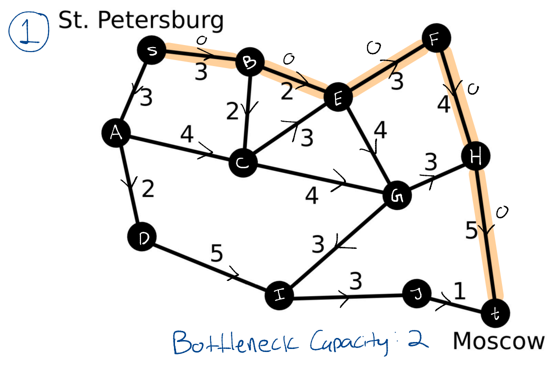
In the next section, we illustrate this algorithm with an example.

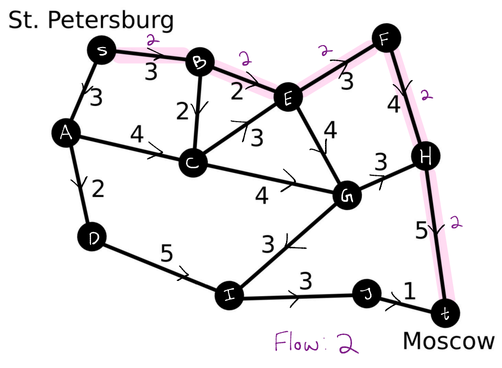
3.1 Example: Finding Max Flow from St. Petersburg to Moscow

Let's use the Ford-Fulkerson algorithm to find the maximal flow of wheat from St. Petersburg to Moscow.

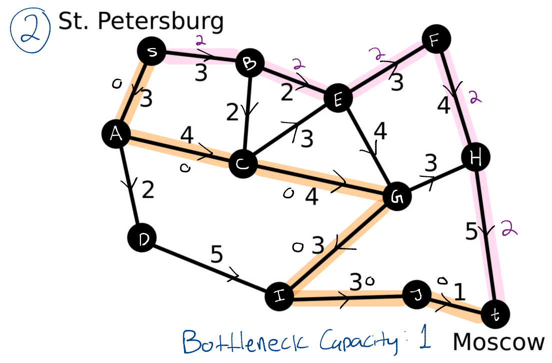
In each figure, the capacities of each edge are typed in black. I will hand-write in purple the flow along an edge, after we update. When 0, I will hand-write in black the current flow on an edge while we are searching for an augmenting path. I will highlight with pink our current path and highlight with orange an augmenting path.

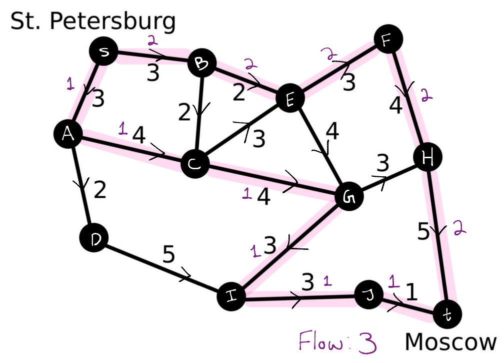
Step 1: start with the 0 flow and find an augmenting path. Compute the bottleneck capacity and update flow accordingly.



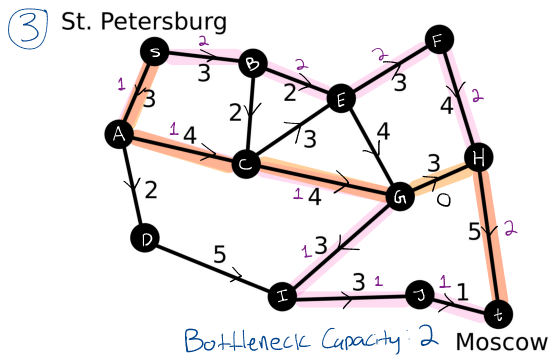


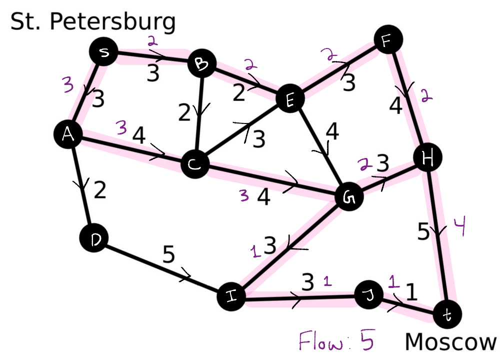
Step 2: Notice that we can still find an augmenting path. Compute the bottleneck capacity and update flow accordingly.



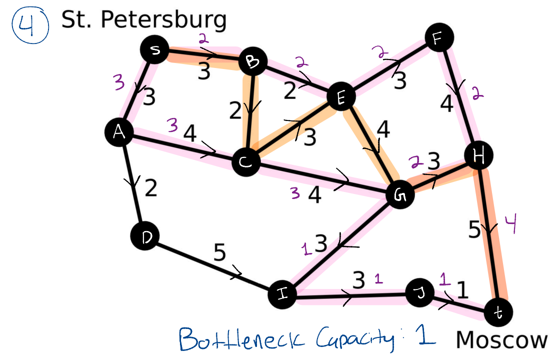


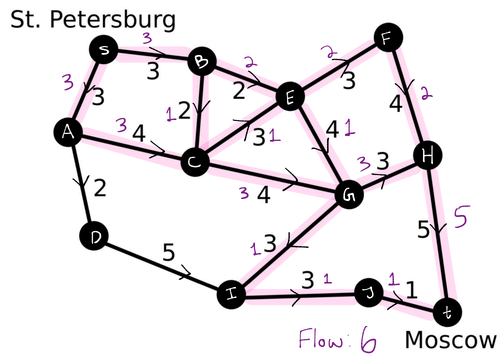
Step 3: Notice that we can still find an augmenting path. Compute the bottleneck capacity and update flow accordingly.





Step 4: Notice that we can still find an augmenting path. Compute the bottleneck capacity and update flow accordingly.





Step 5: There are no more augmenting paths. In this example, this is easy to see: the edges leaving our source are at capacity, as well as the edges entering our target. Algorithm is terminated.

In conclusion, we can transport 6 wagons full of wheat from St. Petersburg to Moscow.

Before moving on to describing this problem using linear programming, let's make a few remarks. First, notice that I never used an augmenting path that traveled backwards on a path (see step 3 of algorithm). If you are interested, the video [2] gives an example traveling backwards on a path. The author provides a great explanation of the process and reasoning. Second, notice that we often made arbitrary choices when selecting an augmenting path. Thus, it is possible that there exists two paths, both of which obtain the maximal flow. In fact, we will see in Section 4.2 that another optimal path does, indeed, exist.

4 Solving for Max Flow using Linear Programming

Finding maximum flow of a network can also be solved using linear programming. I will illustrate how to do this below.

Let (V, E) be a network, with vertices $V = \{i\}$ and edges $E = \{e_{i,j}\}$. Suppose the capacity of the edge $e_{i,j}$ is given by $\mu_{i,j}$.

Given a flow f on (V, E), we can set up a linear program to find max$f$ as follows:

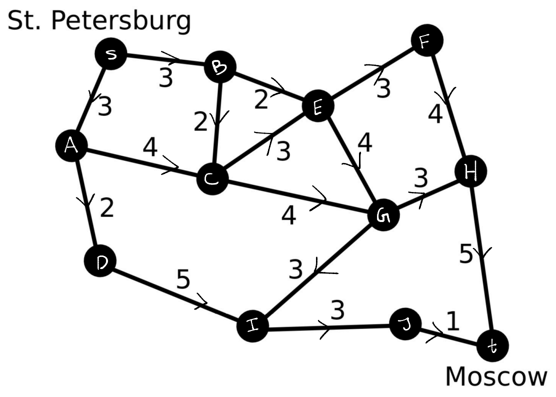
$$\begin{array}{rrll}
\text{Maximize} & f = \sum_{k:e_{k,t}\in V} f(e_{k,t}) - \sum_{j:e_{t,j}\in V} f(e_{t,j}) & & \\
\text{Subject to} & \sum_{j:e_{s,j}\in V} f(e_{s,j}) - \sum_{k:e_{k,s}\in V} f(e_{t,s}) & = f & (1)\\
 & \sum_{j:e_{t,j}\in V} f(e_{t,j}) - \sum_{k:e_{k,t}\in V} f(e_{k,t}) & = -f & (2)\\
 & \sum_{j:e_{i,j}\in V} f(e_{i,j}) - \sum_{k:e_{k,i}\in V} f(e_{k,i}) & = 0 \text{ for } i \neq s,t & (3)\\
 & e_{i,j} & \leq \mu_{i,j} \text{ for all } (i,j) & (4)\\
 & e_{i,j} & \geq 0 \text{ for all } (i,j) & (5)
\end{array}$$

After all, condition (1) requires that the entire flow *leaves* the source. Condition (2) requires that the entire flow *enters* the target. Condition (3) requires that the flow is balanced: that is, the flow entering a vertex must equal the flow leaving a vertex. Finally, condition (4) requires that we do not exceed the capacity of an edge while (5) establishes non-negativity.

This general set-up will allow us to find the maximum amount of wheat we can transfer from St. Petersburg to Moscow.

4.1 Example: Finding Max Flow from St. Petersburg to Moscow

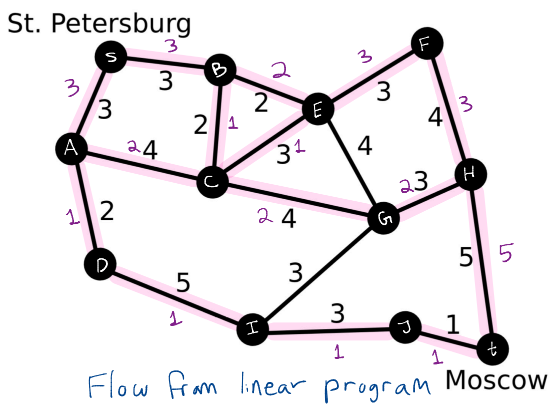
Recall the network we are working with:



I used a linear program package from SciPy, titled lpsolve, to run the linear program in Python. My code can be found in the file titled ‘StPetersburgToMoscow.ipynb’, but here is the output:

```
[['e_sA' '3']
['e_sB' '1']
['e_AC' '2']
['e_AD' '1']
['e_BC' '0']
['e_BE' '1']
['e_CE' '2']
['e_CH' '0']
['e_DI' '1']
['e_EF' '3']
['e_EH' '0']
['e_FG' '3']
['e_GH' '0']
['e_Gt' '3']
['e_HI' '0']
['e_IJ' '1']
['e_Jt' '1']]
```

This output gives us the flow illustrated below.



Here are some important takeaways:

- The linear program also found a max flow of 6. Phew!
- The path taken to obtain the maximum flow produced by the linear program solver in Python is *different* from the path we found suing Ford-Fulkerson. This was to be expected, but it is a nice illustration that there are may be multiple paths that produce the maximum flow.

5 Future Directions

In conclusion, we have demonstrated how to solve a max flow problem using both the Ford-Fulkerson algorithm as well as linear programming. We used both methods to find the maximum flow of wheat from St. Petersburg to Moscow.

Moving forward, I would be interested in answering the following questions:

- Finding the maximum flow of a network is the same as finding the minimum cut of a network [1]. Is linear program for finding the maximum flow dual to the linear program for finding the minimum cut? Brief google searches indicate that this is, indeed, true.
- I put a direction on the edges in order to make the Ford-Fulkerson algorithm simpler. Further, this meant the matrices used for the linear program had smaller dimension. Although it's impossible, in this problem, to get a maximum flow greater than 6, a different orientation on edges may produce a smaller maximum flow. If we allowed for movement along each edge in either direction, is there an easy way to implement this in Python (that is, without doubling the dimension of our matrix)?

References

[1] Hulpke, Alexander. *Combinatorics: Enumeration and Structure.* Lecture Notes, CSU, 2021.

[2] Michael Sambol. *Ford-Fulkerson in 5 minutes* [Video]. YouTube.
https://www.youtube.com/watch?v=Tl90tNtKvxs