Charles Lu Catchpoole

Leonidas Kontothanassis

DS 210

7 May 2023

Rail Explorers and The Maximum-Flow, Minimum Cut Theorem

In order to use this dataset in an effective way I ventured outside of the algorithms we learned in this class. After sitting down with Professor Kontothanassis for some time, he recommended something called the **Max-flow Min-cut** theorem, and a subset algorithm known as the **Ford-Fulkerson** algorithm.

A bulk of the work for this project was the research and note-taking on these two concepts, even though they are pretty straight forward. Imagine a graph network as a series of pipes, and you're trying to push an amount of liquid through the pipes. You want to choose a path that produces the maximum amount of flow in a minimum capacity 'cut', which is merely a partitioning of the vertices into subsets. So an 's-t' cut refers to the maximum amount of flow being pushed through a sink node s, through a minimum cut of a network, and then to the source node t. The Ford-Fulkerson algorithm tackles this problem by iteratively searching for the path that it can push the most flow through, given each edge has a 'capacity' of how much flow it can hold.

To apply this thinking for our dataset you need to become slightly familiar with the family business.

During each day beginning at 9am there is a tour every 1.5 hours until closing for a total of 6 tours per day. Each tour can seat a maximum of 56 people, and every row in the dataset represents a customer booking, for which there are roughly 10,000 of. From here we can think of the tours as nodes in our graph network and the amount of bookings as the edges

that connect them, an edge with a capacity of 56 and a weight corresponding to how many people have signed up.

In order for rust to read our dataset, I had to perform some data sorting. First, I grouped the tours in ascending order based on the date of the tour. Then, I generated a new column of tour ID's that concatenated the tour name, date, and time. Rust can then count the length of Tour_ID's with the same name and create a single node for every tour. Included in this is the rider count, which is the sum of party sizes for each unique Tour ID. This is given the name 'capacity' in the code, despite it not actually representing the capacity unless a tour is sold out.

From there the code imports the csv file and generates the graph and performs the Ford_Fulkerson algorithm. It prints the maximum flow, the minimum cut, and the "total possible flow" which would be the output if every tour was sold out on every day (if edge weight = edge capacity) as explained in the code, our source and sink nodes are generated separately and later connected to the rest of the graph.

We get a maximum flow of 35,134 for a possible maximum flow of 46,760. The minimum cut produces a large output of roughly 800 of the nodes. This is because minimum cut represents the nodes that, if removed, would minimize the maximum flow able to be created during the graph. Maximum flow is measured by moving through each to find how many people have been signed up for a tour.

This means that not every tour was sold out. Which is interesting because as someone who has worked as a tour guide at this company, the summer days are usually fully sold out. The dataset runs from May 1st to November 1st, so I am willing to bet the disparities in attendance lie in the tails between the start and end of a season, somewhat resembling a normal distribution. There are always exceptions though. Sometimes there are walk-ins, when there is space available. Or maybe there is bad weather so there are cancellations.

In conclusion, this optimization problem doesn't improve the actual business. Consumers here are uniform with uniform preferences. If we wanted to make it more exciting we could implement an economic cost function. What the program attempts to do is find the maximum number of riders that can ride on each tour given the party sizes in our dataset, which it succeeds at. While the minimum cut output is less clear, it still prints an output that makes sense, given that each edge has the same capacity, so their removal will affect the graph the same way.

Afterword

Conceptually this idea has been in my head since DS 110. As I developed as a programmer and data scientist, I wondered when I was going to be able to sink my teeth into a dataset that could actually make an improvement on someone's way of life.

As it turns out I'm still going to be waiting on that, as you'll see the true applications of this project are somewhat nonexistent and rather serves as a simulator of sorts, but I'm still enormously proud of this project. So enough rambling. To summarize: in 2015 my Mum and Dad founded *Rail Explorers,* an eco-tourism company based on pedal-powered railbikes. The company has represented an integral part of my upbringing and was my inspiration for this project. This dataset is customer booking information. I know, right? That was kind of anti-climatic.

I was extremely close to giving up and using six-degrees of separation on a generic dataset so many times during this project. I am so glad I stuck with it because seeing this project through has made me extraordinarily satisfied. Though there are no true practical applications of this code, I will show it to my parents with the pride of knowing their work is exhibited in my studies.