**\*Shortest path algorithm\***

**What to know abt Data set:**

**Create a new metro express line that capitalizes on the missed revenue of late night ridership**

**Ex. people take metro to dinner/bars and uber back because they stay out too late, sporting events, concerts, etc.**

**Plan of attack:**

* **Compile data that we can use to determine patterns of ridership during specific events/times**
  + [**Friday and Saturday nights across a two month time period (Sep-Oct24) to supplement provided data**](https://www.wmata.com/initiatives/ridership-portal/Metrorail-Ridership-Summary.cfm) **(go to last table)**

**Data we would need**

* **Daily/peak-hour ridership per station**
* **OD (origin-destination) matrices showing where passengers travel the most**
* **Transfer station loads**

**Metro Network Data**

* **Station locations (coordinates)**
* **Existing routes and track layout**
* **Travel times between stations -** [**https://developer.wmata.com/api-details#api=5476364f031f590f38092507&operation=5476364f031f5909e4fe330c**](https://developer.wmata.com/api-details#api=5476364f031f590f38092507&operation=5476364f031f5909e4fe330c) **(i’m so fucking lost)**
* **Transfer times at hubs**

**Operational Constraints**

* **Track-sharing rules (Can express trains skip stations?)**
* **Platform availability for express stops**
* **Infrastructure limits (number of express trains per hour)**
* **Use Shortest Path Algorithm (Dijkstra’s) to determine**
* **Optimize Lines by ridership - Shortest path algorithm**

**Roles - The Data ScieZYNtists**

**Formulate guesses based on trends we are finding**

**Cherry blossom bomb**

**Trends across seasons?**

**Trends based on sporting events? (nats, caps, wizards, commanders)**

**Find obvious and non obvious things**

**Bonus Data -**

**https://www.wmata.com/initiatives/ridership-portal/Metrorail-Ridership-Summary.cfm**

**DATES**

**Dec 4, 2024 (wednesday)-**

**Dec 21, 2024(saturday)**

**June 5, 2024(wednesday) — nationals home game vs NY Mets**

**June 15, 2024(saturday) — nationals home game vs Miami Marlins**

**March 6, 2024(wednesday)**

**March 23, 2024(saturday) — cherry blossom**

**September 4, 2024(wednesday) — Adams Morgan Day Festival: A family-friendly festival held in the Adams Morgan neighborhood (woodley park metro-red line)**

**September 21, 2024(saturday)**

**Stations to look at -**

**Origin**

**Dupont Circle →bars**

**Foggy Bottom → bars and GW**

**Gallery Place →Bars**

**Woodley Park → Adams Morgan**

**Destination**

**Tenleytown-AU → American U**

**Foggy Bottom → bars and GW**

**Navy Yard → Stadium/Homes**

**Navy → Gallery = 7 min 4 stops between**

***West Line***

**Gallery → Foggy = 1+5 = 6 (3 stops+1 transfer @ metro)**

**Foggy → Rosslyn = 3 min (no stops)**

***North Line***

**Gallery → Dupont 5min (3 stops)**

**Dupont → Woodley 2 min (no stops)**

**Woodley → AU 6 min (3 stops)**

**Yes! A shortest path algorithm can be used to design an express metro line based on ridership data. The goal would be to optimize travel time for the highest number of passengers while minimizing the number of stops.**

### **Data Needed**

**To make this work effectively, you’d need:**

1. **Ridership Data**
   * **Daily/peak-hour ridership per station**
   * **OD (origin-destination) matrices showing where passengers travel the most**
   * **Transfer station loads**
2. **Metro Network Data**
   * **Station locations (coordinates)**
   * **Existing routes and track layout**
   * **Travel times between stations**
   * **Transfer times at hubs**
3. **Operational Constraints**
   * **Track-sharing rules (Can express trains skip stations?)**
   * **Platform availability for express stops**
   * **Infrastructure limits (number of express trains per hour)**

### **Algorithm Choice**

* **Dijkstra’s Algorithm: Finds the shortest travel time between key stations.**
* **Floyd-Warshall: If you need all-pairs shortest paths (for checking multiple route options).**
* ***A Algorithm*\*: Can be used if you have heuristics (e.g., ridership-based weight).**
* **Graph Partitioning: To cluster high-ridership nodes and form express routes.**

### **How It Works**

1. **Build a weighted graph where stations are nodes, and edges are weighted by travel time + ridership penalties (high ridership stations are prioritized).**
2. **Run a shortest path algorithm that balances speed and ridership coverage.**
3. **Select key express stops where skipping intermediate stations minimizes travel time without losing too many riders.**
4. **Validate with simulation using ridership models.**

**Would you like help designing a model for this?**

**4o**

##### **You said:**

**can you walk through how to do that in python with csv files for data**

###### **ChatGPT said:**

**Sure! Let's walk through a Python implementation using Dijkstra’s algorithm with CSV files containing ridership and metro network data.**

### **Step 1: Prepare the Data**

**You'll need two CSV files:**

**metro\_network.csv (station connectivity and travel times)  
css  
CopyEdit  
from\_station,to\_station,travel\_time**

**A,B,5**

**B,C,7**

**A,C,10**

**C,D,3**

**ridership.csv (daily ridership per station)  
css  
CopyEdit  
station,ridership**

**A,5000**

**B,12000**

**C,8000**

**D,3000**

### **Step 2: Load Data into Python**

**We'll use pandas to read the data and networkx for graph operations.**

**python**

**CopyEdit**

**import pandas as pd**

**import networkx as nx**

**# Load the metro network**

**network\_df = pd.read\_csv("metro\_network.csv")**

**ridership\_df = pd.read\_csv("ridership.csv")**

**# Create a graph**

**G = nx.DiGraph()**

**# Add edges with travel time as weight**

**for \_, row in network\_df.iterrows():**

**G.add\_edge(row['from\_station'], row['to\_station'], weight=row['travel\_time'])**

**# Convert ridership data into a dictionary for quick lookup**

**ridership = dict(zip(ridership\_df['station'], ridership\_df['ridership']))**

### **Step 3: Modify the Graph for Express Routing**

**We’ll adjust weights based on ridership so that high-ridership stations are more likely to be express stops.  
For this, we penalize skipping high-ridership stations by increasing edge weights.**

**python**

**CopyEdit**

**# Normalize ridership (so values are manageable)**

**max\_ridership = max(ridership.values())**

**for u, v, d in G.edges(data=True):**

**mid\_station = v # The station being skipped if we take this edge**

**penalty = (ridership.get(mid\_station, 0) / max\_ridership) \* 5 # Scale factor**

**d["weight"] += penalty # Increase travel cost if skipping a busy station**

### **Step 4: Compute the Express Route**

**Now, we use Dijkstra’s algorithm to find the fastest express route between two key stations.**

**python**

**CopyEdit**

**# Define express route start and end**

**start\_station = "A"**

**end\_station = "D"**

**# Find shortest path considering both travel time and ridership-based penalties**

**express\_route = nx.dijkstra\_path(G, start\_station, end\_station, weight="weight")**

**print("Optimized Express Route:", express\_route)**

### **Step 5: Visualize the Network (Optional)**

**If you want to visualize the network and the selected express route:**

**python**

**CopyEdit**

**import matplotlib.pyplot as plt**

**plt.figure(figsize=(8,6))**

**pos = nx.spring\_layout(G) # Layout for visualization**

**nx.draw(G, pos, with\_labels=True, node\_color="lightblue", edge\_color="gray")**

**nx.draw\_networkx\_nodes(G, pos, nodelist=express\_route, node\_color="red") # Highlight express stops**

**plt.show()**

[**https://www.wmata.com/fares/basic.cfm?os=windhgbityl&ref=app**](https://www.wmata.com/fares/basic.cfm?os=windhgbityl&ref=app)

[**https://franchise.teriyakimadness.com/franchise-blog/subway-franchise-cost/#:~:text=Initial%20investment%20costs%20for%20a,to%20cover%20royalties%20of%208%25**](https://franchise.teriyakimadness.com/franchise-blog/subway-franchise-cost/#:~:text=Initial%20investment%20costs%20for%20a,to%20cover%20royalties%20of%208%25)**.**