Optimization Techniques for Train Route Selection

Utilizing Dijkstra's Algorithm for Improved Travel Planning

Your Name

A report presented for the completion of
AI-1 System Project

Department of Artificial intelligence
Friedrich-Alexander-Universität Erlangen-Nürnberg
Germany
Date: March 12, 2024

Abstract

In this project, we worked to find out the best route for Indian railways using two train schedule datasets. The goal of this project was to help people make their trips easier based on their preferences, whether they wanted the quickest, cheapest, or simplest route. To achieve our goal, we used Dijkstra's algorithm and turned the train schedules into a map-like format. The algorithm was changed according to different cost function requirements, like least number of stops, shortest distance, lowest cost, or earliest arrival time. Our efforts and the result showed that this approach can really help in suggesting the best travel options, making it clear that mathematical and computational algorithms can solve practical problems like planning a train journey. This work study shows that a smart use of algorithms can solve our everyday life problems in the easiest and most efficient ways.

1 Introduction

Navigating through India's vast railway network to find the optimal train connection poses a significant challenge due to the complexity of its schedules and the diverse needs of travelers. In this context, the efficiency of travel planning becomes a critical concern, reflecting a broader issue in transportation logistics where optimal solutions must be sought amidst varying constraints. This project investigates the application of computational algorithms to enhance train travel planning in India, focusing on the adaptability of such algorithms to meet different travel preferences, such as minimizing travel time, cost, stops, or prioritizing earliest arrival.

1.1 Research question

The primary question for this research is: How can computational algorithms like search algorithm be adapted to optimize train travel routes, considering various cost functions that reflect different traveler preferences? This inquiry stems from the observation that travelers often must compromise on one aspect of their journey to optimize another, due to the lack of a flexible, all-encompassing travel planning tool.

1.2 contribution

The contributions of this project are twofold:

1. The development of a graph-based representation of the Indian railway system that allows for the dynamic application of Dijkstra's algorithm to find the best train connections based on user-specified cost functions.
2. An analysis of the algorithm's performance with different cost functions, demonstrating its ability to provide customized travel solutions that cater to individual preferences.

1.3 Overview

The report unfolds as follows: Section 2 delves into the complexities of the Indian railway system and the necessity for an efficient travel planning tool. Section 3 introduces the problem briefly. The data processing and converting it to a graph are discussed in Section 4. The algorithmic approach to solving this problem for various cost functions is elaborately discussed in Section 5. Section 5 also includes a brief computational time comparison for each cost function. Finally, Section 6 concludes the report by reflecting on the findings, their implications for travelers and transport authorities, and potential future directions for research in this area.

2 Background

Understanding the optimization of train connections with Indian railway data requires familiarity with several foundational concepts and tools. This section outlines these essentials along with the mathematical notations pivotal for the discourse in this report.

Graph Theory Basics

Define the railway network as $G = (V, E)$, where:

- V: Set of vertices ($v \in V$), representing train stations.
- E: Set of edges ($e \in E$), representing train routes between stations.
- For each edge e, assign a weight $w(e)$, indicating the route's cost (could be distance, time, or price).

Dijkstra's Algorithm

For finding the shortest path within G:

1. **Initialize**: $d(v) = \infty, \forall v \in V \setminus \{s\}$ and $d(s) = 0$ for a source s.
2. **Use a priority queue** Q with vertices ordered by $d(v)$.
3. **Update distances**: For u selected from Q, $d(v) = \min(d(v), d(u) + w(u, v))$, $\forall v$ adjacent to u, until Q is empty.

Cost Functions

To guide optimization, modify $w(e)$ for $e \in E$, with:

- C_{stops}: Targets minimizing the number of stops.
- C_{distance}: Aims at minimizing the total travel distance.
- C_{price}: Seeks to minimize the overall cost of the journey.
- $C_{\text{arrivaltime}}$: Minimize the total travel time, including both on-train duration and station waiting times.

CSV File Format

The CSV (Comma-Separated Values) format is crucial for managing the train schedule data. Each file row corresponds to a record, such as a train's stop at a station, featuring columns for various attributes like Train No., station Code, Arrival time, Departure time, and Distance.

With this groundwork, the report advances to leverage these principles in tackling the challenge of identifying the best train connections. The graph representation enables a structured depiction of the railway network, Dijkstra's algorithm facilitates the route optimization process, and the cost functions ensure adaptability to diverse travel requirements. Employing CSV files streamlines the engagement with the railway schedule data, providing a solid foundation for our computational exploration.

3 Problem description

For optimizing train connections across the Indian railway system, our analysis begins with a comprehensive dataset delineated in a CSV format, capturing the intricate details of train schedules. This dataset, which meticulously records train numbers, station codes, along with arrival and departure times, and distances between stops, forms the cornerstone of our investigative framework. We model the railway network as a graph, denoted by G=(V,E), where V represents the stations, and E encompasses the direct connections between these stations. Each connection, or edge, within E is imbued with attributes relevant to travel time and distance, pivotal for route optimization.

The essence of our endeavor is to formulate an algorithm that unerringly identifies the most efficient pathways between any two stations within this network. This quest for efficiency is navigated through the prism of several cost functions, each aiming at a distinct optimization goal: reaching the earliest time by starting trip in Carrivaltime, curtailing the number of stoppage Cstops, seeking the shortest route in terms of distance Cdistance, and ensuring the journey is cost-effective Cprice. The methodology involves dynamically adjusting the weights of the edges in G to reflect the chosen cost function and deploying a customized variant of Dijkstra's algorithm to traverse the graph, thereby unveiling optimal routes that resonate with varied passenger preferences.

A crucial element of this task is to account for nuances such as the necessity for a minimum waiting period during train transfers and the algorithm's adaptability to itineraries that span across days, especially significant for overnight travels.

Through diligent processing of the dataset and strategic application of the algorithm, our goal is to curate a collection of optimized travel plans. These plans are not merely pragmatic and customized to a broad spectrum of requirements but also highlight the transformative potential of algorithmic methodologies in enhancing the operational efficiency of railway transportation. This project transcends the immediate objective of route optimization, offering insights into the broader applicability of data-driven strategies in surmounting real-world logistical challenges in transportation networks.

4 Data preprocessing

The original datasets were in CSV format; to make them usable for Dijkstra's algorithm, we converted CVS data into a graph $G = (V, E)$, where V represents all the stations and E represents all the connections between stations. To do so, we used the Python

dictionary data structure, where the key of the dictionary represents each station v ($\forall v \in V$), and we stored all the connections from the station as values in the dictionary. This is how we used a dictionary to represent CSV data in a graph.

5 Optimization Techniques for Train Route Selection

The Single-Source Shortest Path (SSSP) problem consists of finding the shortest paths between a given vertex v_i to every other vertex v_j in a graph, where $v_i, v_j \in V$ and $i \neq j$. Algorithms such as Breadth-First-Search (BFS) for unweighted graphs or Dijkstra solve this problem. As we have cost functions that are basically weight, we used Dijkstra to find out the best route from the source station s to the destination d. The priority queue, Q, was used to ensure vertex with the minimum distance min(d(V)) from the source vertex s is consistently selected. The following algorithm is our general approach:

Algorithm 1 Dijkstra's Algorithm for Optimized Train Route Selection

1: For each vertex $v \in V$, set the initial distance $d(v)$ from the source vertex v_s as:

$$d(v) = \begin{cases} 0 & \text{if } v = s, \\ \infty & \text{otherwise.} \end{cases}$$

2: Initialize Q, a priority queue with s_{obj}.
3: **while** Q is not empty and $u \neq d$ **do**
4: Remove a vertex u from Q with the minimum $d(u)$.
5: **for** each neighbor v of u where $v \in V$ and $(u, v) \in E$ **do**
6: **if** $d(v) > d(u) + w(u, v)$ **then**
7: $d(v) = d(u) + w(u, v)$
8: push (v_{obj}) into Q
9: **end if**
10: **end for**
11: **end while**
12: The algorithm terminates when Q is empty or $u = v_d$. At this point, $d(v)$ holds the shortest distance from the source vertex v_s to the destination v_d.

Given the distinct cost functions, the algorithm's implementation varied accordingly, especially in the calculation of weights $w(u, v)$ for the relaxation step. Here's a closer look at the cost functions:

5.1 Distance:

For the distance cost function, the edge weight $w(u, v)$ is directly the distance between u and v. The relaxation process follows the standard approach as demonstrated in the algorithm above.

5.2 Stops:

When minimizing stops, each edge $e \in E$ in the relaxation process is considered to have a weight $w(u, v) = 1$, indicating a uniform cost for each train change or traversal between stations.

5.3 Price:

To mathematically represent the conditions and adjustments described for handling train routes with specific ticket validity and cost considerations in the Dijkstra algorithm, we formalize the initialization and relaxation conditions as follows:

5.3.1 Initialization

Given that every ticket costs 1 and is valid for a single train journey until midnight, the initial cost $d(s)$ from the starting station s is set to 1:

$$d(s) = 1$$

For all other stations $v \in V$, where V is the set of all stations, and $v \neq s$, we set:

$$d(v) = \infty$$

5.3.2 Relaxation Condition

The relaxation condition for updating the distance to a neighboring station v from station u includes the cost of changing trains or when the next arrival time is past midnight. The weight $w(u, v)$ for the edge from u to v is set to 1 under these conditions:

$$w(u,v) = \begin{cases} 1, & \text{if train changes at u to reach v or arrival time at } v \text{ is past midnight} \\ \text{standard weight}(0), & \text{otherwise.} \end{cases}$$

The distance $d(v)$ to each neighbor v is then updated as:

$$d(v) = \min(d(v), d(u) + w(u,v))$$

5.3.3 Adaptation for Equal Weights

The algorithm must accommodate scenarios where different paths provide equal weights to reach the same vertex v. This condition is particularly relevant when considering the possibility of reaching v with minimal cost using the same train or a different train that goes nearest to v. Thus, the relaxation condition is adjusted to consider paths with equal weight, allowing for the update of $d(v)$ even when the weights are equal, to ensure all minimal paths are considered:

$$d(v) \leq d(u) + w(u,v)$$

This adjustment ensures the algorithm captures all optimal routes under the specified cost function, accommodating the unique ticketing constraints and travel preferences outlined.

5.4 arrivaltime

The trip's starting time from s is given, and the fastest route to reach the destination d is sought. To achieve the fastest route, the algorithm needs to choose connections e that arrive earliest. Two conditions were properly handled: a 10-minute time is required if $u_{\text{train\_no}} \neq v_{\text{train\_no}}$, and the arrival time at u will be considered the next day if it is past midnight. We initialized the initial weight as the arrival time for $v = s$ and a long time for $v \neq s$.

5.4.1 Initialization

$$d(v) = \begin{cases} t_{\text{start}}, & \text{if } v = s \\ \infty, & \text{otherwise} \end{cases}$$

where t_{start} is the starting time for the trip from station s.

5.4.2 Relaxation

for each edge $(u, v) \in E$ updates the arrival time at v considering:

1. **Train Change Requirement**: Adding a 10-minute transfer time if the train identifier for the edge (u, v), $id_{\text{train}}(u, v)$, differs from the previous edge.
2. **Midnight Crossing Adjustment**: Adjusting the day if the calculated arrival time at v crosses midnight relative to u.

Hence, the update for $d(v)$ during relaxation becomes:

$$d(v) = \min\left(d(v), d(u) + t_{\text{arrival}}(u, v) + \Delta t_{\text{transfer}} + \Delta t_{\text{midnight}}\right)$$

where:

- $\Delta t_{\text{transfer}} = 10$ minutes if $id_{\text{train}}(u) \neq id_{\text{train}}(v)$, and 0 otherwise.
- $\Delta t_{\text{midnight}}$ accounts for the day adjustment if $t_{\text{arrival}}(u, v)$ crosses midnight.

5.4.3 Adaptation for Equal Weights

Again, we accounted for situations where paths from u to v have equal weights when storing them in the priority queue. In such cases, the *train_id* or the parent of u is required to be unique from any existing connection. This strategy allowed us to address scenarios where paths of the same weight, yet via different connections, could yield the best solution. The rationale is that identical weights from disparate connections do not always necessitate a train change or may facilitate a timelier connection. Thus, the relaxation condition is adjusted to consider paths with equal weight, allowing for the update of $d(v)$ even when the weights are equal, to ensure all minimal paths are considered. Here, $w(u, v)$ is defined as $d(u) + t_{\text{arrival}}(u, v) + \Delta t_{\text{transfer}} + \Delta t_{\text{midnight}}$, where $\Delta t_{\text{transfer}}$ accounts for the required time if a train change occurs, and $\Delta t_{\text{midnight}}$ adjusts for the day if the arrival time crosses midnight:

$$d(v) \leq d(u) + w(u, v)$$

5.5 Determining the Optimal Route

Finally, we suggest the best route $x_i : y \to z; x_{i+1} : y \to z \ldots$ that includes $x =$ train IDs and y, $z =$ the starting and end stoppage numbers for each specific train. To do so, we used a list that stores each station's detailed connection information, such as the parent station code, train ID, etc. When we obtain the optimal weight for the target, we backtrack from each station to their parent station to generate the solution in required format.

5.6 Average Time Comparison

The table below presents the average computation times for each approach, comparing the performance on two datasets: "mini-schedule" and "schedule."

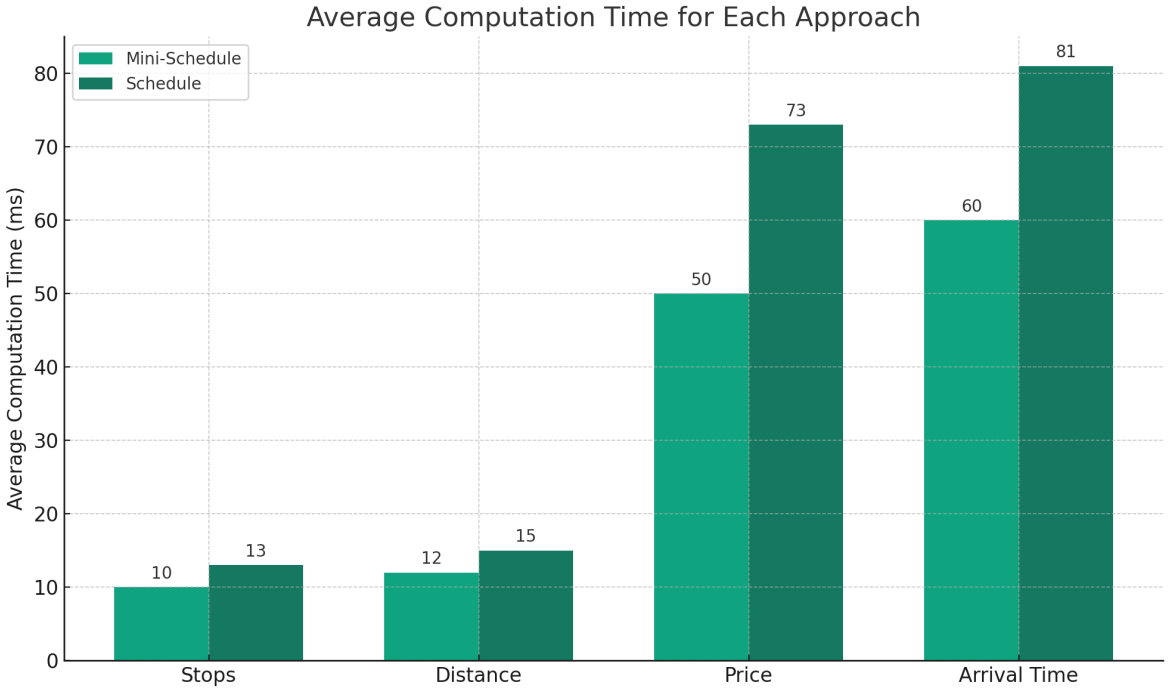

Figure 5.1: Average Computation Time for Each Approach

As shown in figure 5.1, the price and arrival time cost functions take a bit longer compared to other cost functions. The schedule dataset contains more connections compared to the mini-schedule, which results in additional computation time for the schedule dataset.

6 Conclusions

In this report, we have used advanced data and graph-based modeling to navigate the complexities of optimizing rail connectivity in the Indian railway network. By applying the Dijkstra algorithm developed for cost functions (C_{stops}, $C_{distance}$, C_{price} and,

$C_{arrivaltime}$), we developed an optimal solution that satisfies various passenger preferences. The main challenge we faced was to handling the arrival time and price cost functions (C_{price} and, $C_{arrivaltime}$), which required intricate adjustments to Dijkstra's algorithm to ensure accuracy and practicality in travel planning. To ensure that Dijkstra's algorithm required robust changes through iterative refinement and strategic optimization demonstrated the potential of algorithmic approaches to enhance travel planning and passenger experience within the rail system. In our approach, we found it takes a bit longer computational time for cost functions- price and arrivaltime- [5.1], therefore, optimizing the computation time for the price and arrival time cost functions is recommended as future research work. The insights and approaches developed here not only highlight the effectiveness of data-driven algorithmic solutions in transportation planning but also pave the way for future innovations in user-centered transportation networks which works well.