

Smarter Travel Google Map

Optimizing Best Visiting Route with Multiple Destinations for
Google Map

Introduction

The "Smarter Travel: Optimizing Best Visiting Route with Multiple Destinations for Google Maps" project seeks to address this pressing need by harnessing the power of advanced algorithms to compute the shortest path that allows an individual to visit a set of multiple destinations, starting from a designated origin point. By incorporating the Shortest Path and Held-Karp algorithms, this project aims to revolutionize the way individuals plan and execute multi-destination trips, saving time, reducing travel costs, and enhancing the overall travel experience.

Motivation

Our team was fueled by a shared sense of motivation and enthusiasm. The passion for travel, technology, and a desire to solve a real-world problem united us in this endeavor. Each of us brought our unique perspectives and expertise, and the prospect of enhancing route optimization in Google Maps excited us all. This initial motivation served as the driving force behind our collective efforts, propelling us forward to explore innovative solutions and make a positive impact on the travel experience for millions of users.

How We Approached This Problem

Collected data from Google & Google Map

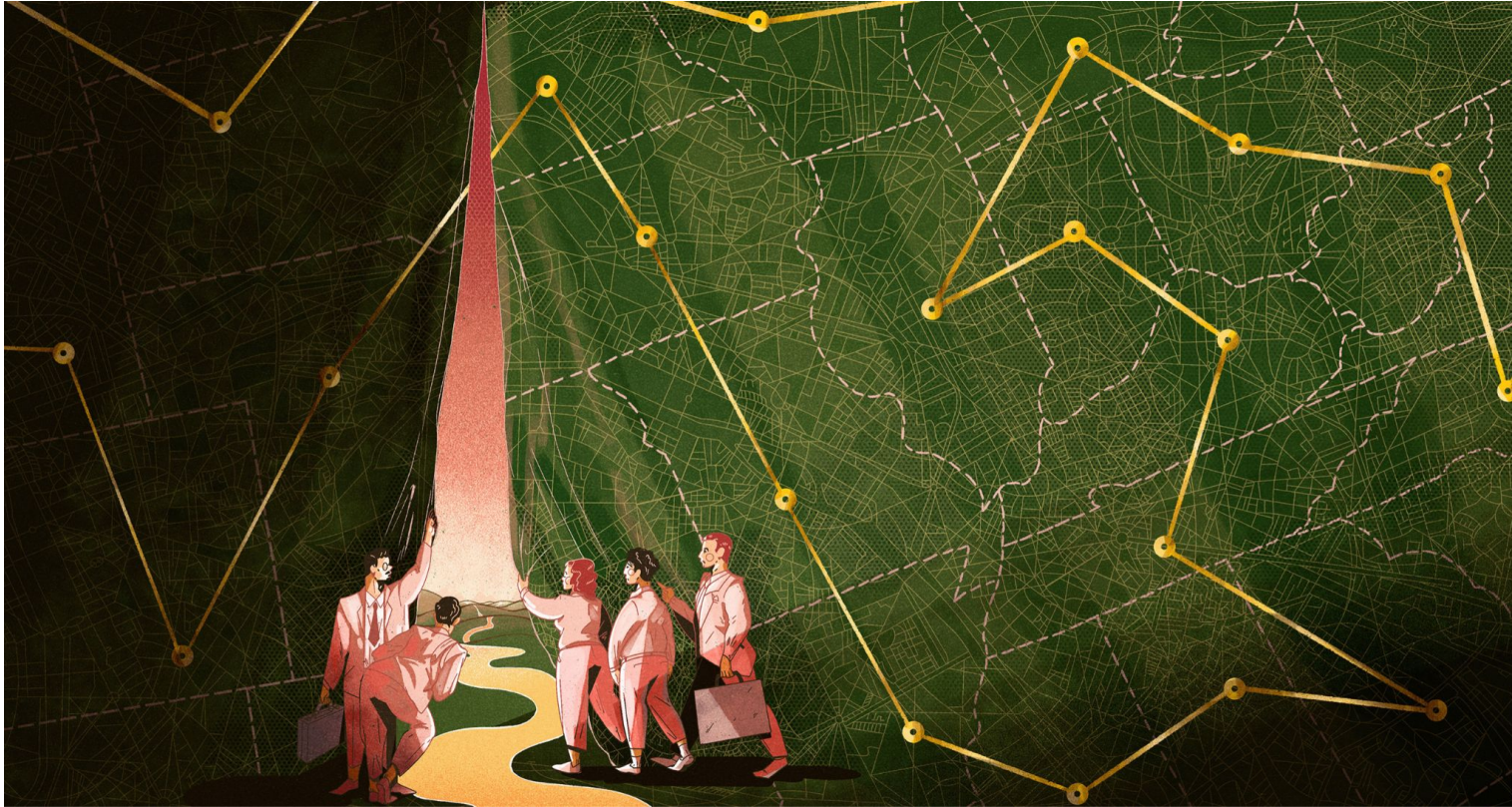
Determined it was a Travelling Salesman Problem (TSP), and we wanted to use Held-Karp algorithm

Defined project scope and assumption

Draw graphs on Boston maps

Wrote codes to implement the algo and retrieved final distance result

Traveling Salesman Problem



Traveling Salesman Problem

Given a list of cities (locations)

and the distances between each pair of cities (locations),


find the shortest possible route

that visits each city (location) exactly once

and returns to the starting city (location)


Held-Karp Algorithm - What and Why?

 Dynamic programming approach + Memoization

 An improvement over the brute-force approach, which would require checking all possible permutations of city orders ($n!$ permutations)

 Works well with relatively small instances of the TSP

 Time complexity of $O(n^2 * 2^n)$, n is the number of locations

 Impractical for very large instances, because of exponential time complexity and exponential memory usage

Problem Scope and Assumption

Single Means of Transportation (Car)

Single Visitor per Trip

Exploration Limited to Boston

Ignoring Visiting Time

Excluding Opening Hours

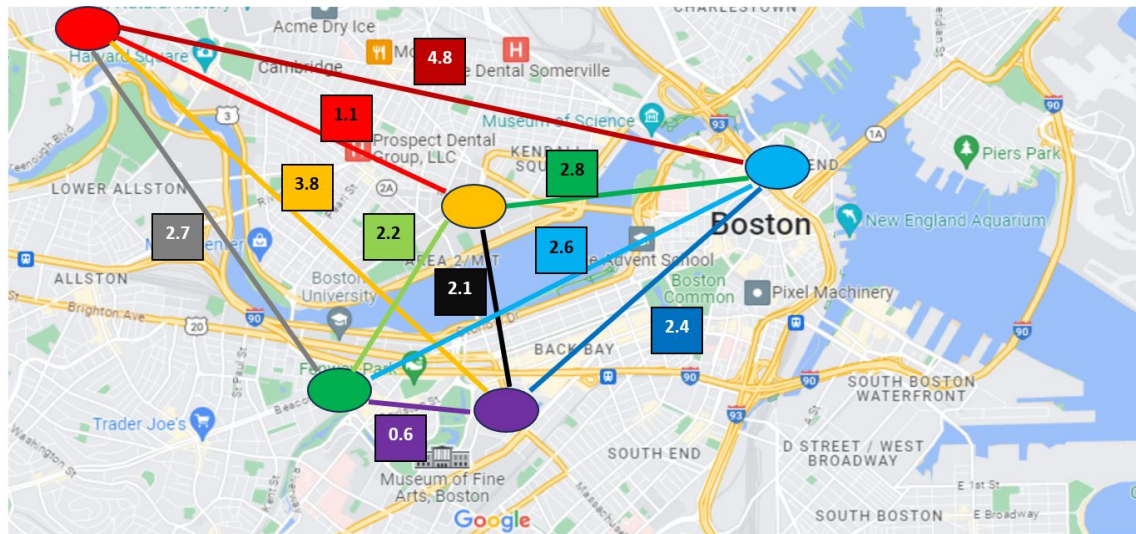
Shortest Distance Implies Shortest Time

Example Test Case

My friend, Sarah, is visiting from Washington DC and wants to see Harvard, MIT, MFA, and Chinatown. We're figuring out the best order for her trip, starting from my house near Brigham Circle.

Shortest distance:

Brigham -> MFA -> Chinatown -> MIT -> Harvard -> Brigham



Harvard



Brigham Circle



MIT



MFA



Chinatown

```

3 usages
def tsp_held_karp(graph, start):
    n = len(graph)
    dp = {}
    # Helper function to generate the key for the dp dictionary
    def get_key(visited, last):
        return tuple([last] + sorted(visited))

    # Base case for the recursion
    def held_karp_helper(visited, last):
        if len(visited) == n:
            return graph[last][start], [start]
        key = get_key(visited, last)

        if key in dp:
            return dp[key]
        min_distance = float('inf')
        min_path = []

        for location in graph[last]:
            if location not in visited and location != last:
                visited.add(location)
                distance, path = held_karp_helper(visited, location)
                distance += graph[last][location]
                if distance < min_distance:
                    min_distance = distance
                    min_path = [last] + path
                visited.remove(location)

        dp[key] = min_distance, min_path
        return min_distance, min_path

    # Starting the recursion from the given start location
    min_distance, min_path = held_karp_helper(set(), start)

    return min_distance, min_path

```

Code

tsp_held_karp:

- Take graph and start location name to then present the minimal distance along with minimum path.
- Uses dynamic programming with memoization

Held_karp_helper:

- Recursive function to explore all possible paths from the starting city to visit all other cities and return back to the starting city
- uses memoization to avoid recomputing results for the same set of visited cities and last city

Does the code give the correct output?

YES!!

smarttravel2 x

```
C:\Users\ahmad\PycharmProjects\cs5800\venv\Scripts\python.exe C:/Users/ahmad/PycharmProjects/cs5800/smarttravel2.py
```

```
Visitor 1: Minimum distance = 9.60 miles
```

```
Visitor 1: Visited places in order = ['Brigham', 'MFA', 'Chinatown', 'MIT', 'Harvard']
```

What's Next?

- Enhance the tool by incorporating walking, public transportation, and biking options.
- Customize transportation options based on user preferences.
- Integrate real-time traffic information for efficient route planning.
- Explore the possibility of including operating hours for practical itineraries.

Conclusion

- Resounding success in developing an innovative and user-friendly solution for finding the shortest path between different places.
- Leveraged the Held-Karp algorithm for accurate and efficient route planning.
- Ensured travelers can optimize itineraries and save time and gas.
- Simplicity and accessibility of the model for effortless route generation.
- Satisfies travel preferences and ensures a seamless journey experience.
- Eco-friendly options to minimize gas consumption and promote sustainability.
- Benefits individual tourists and contributes to reducing carbon emissions.
- Provides travelers with a reliable companion for enjoyable and environmentally conscious travel.