

Trip Planner Web Application based on OpenMP

Professor John Gash

CMPE275 Project 2 Report

Semester: Fall 2016

Xing Yang 010696101

Jiongfeng Chen 010830105

Dan Su 009266920

Table of Content

**1. Introduction**

**2. User Requirements**

**3. System Architecture**

**4. System Components**

4.1 User Request Module

4.2 Route Decision Module

4.3 Database Module

**5. Test Case**

5.1 Frontend

5.1 Small input case (3 cities)

5.2 Large input case (13 cities)

5.4 Performance measurement

**6. Conclusion**

# 1. Introduction

In this project, a web application based on OpenMP is built for the purpose of maximizing the utilization of computing resources of the server. It is highly useful and will largely reduce the response time in the scenarios when the web application requires to apply complex algorithms or require computational load heavy tasks.

The problem to solve for this web application is trip planner. Based on the destinations (we use cities in this application) the users input, the web application which not only tell the user the route which cover all the destinations he want to travel exactly once but also give the best routes such as lowest cost route, shortest distance route, and shortest time route etc as long as the relevant information is available. The web application is highly extensible and be much more powerful by showing exact real time shortest route which are based on the available traffic status data and road construction data.

The trip planner application is essentially to solve the problem using a complex and classic algorithm named Travelling Salesman Problem (TSP), which is located at the heart of the trip planner question. The TSP could be briefly described as followings, if we have a list of cities and the distance information between any of them, what is the shortest possible route to visit each city exactly once and then returns to the original city. It is an NP-hard problem and very important in optimization operations and theoretical computer science.

The TSP solving procedure is highly time consuming because the algorithm running time for solving TSP increases almost exponentially with the number of cities. The algorithm is highly suitable for the parallelization. We implement a parallelized dynamic programming algorithm module for TSP which is C++ code version. The module is developed using the OpenMP and is connected and invoked by the Python Flask based web application. The users input is passed by python, and the TSP module is invoked and returns the desired route results to the web application for demonstration.

The technologies applied include the OpenMP, Python Flask, Google Map API and MySQL database. The user data and request history, along with the relevant city data is saved to the MySQL database.

In order to compare the performance before and after parallelization using OpenMP, we also implement the single thread TSP python code for the web application which don’t have any parallelization. The shorter response time and performance improvement has clearly showed what is the role OpenMP played and how the OpenMP technology is vital for the our trip planner web application.

# 2. User Requirements

The web application should be able to respond to the user request with the corresponding route with lowest cost, such as shortest distance route, lowest cost route or the based on the a series of destinations submitted by the users.

In this project, the input data users submitted will be city names, and the web application should validate whether the user has a legal input or no to ensure the security and avoid crash or error of the web application. The cities are the places where user want to travel exactly one which means the response route should cover all these cities.

Based on the input cities, the web application should return the best route found based on what options the users have selected such as shortest, lowest cost route, and shortest time route etc as long as the relevant geo-information is adequate for the calculation. The web application is easy to extend and to add more functionalities when more information available. For instance, the web application can be extensible to show more route options powered by real time traffic status data and road construction data from online web APIs.

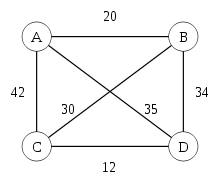
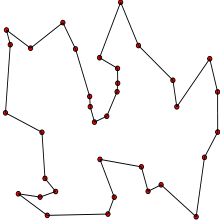


Fig 1 Travelling Salesman Problem (TSP, left) and four cities case(right)

# 3. Architecture Design

The system has two major modules, user request module and route decision module. The user request module is responsible for receiving and handling user request containing the city names. And this module will also finally return the final “best” routes from different aspects. RESTful API is implemented by the user request module. After receiving the data, the user request module will send requests to Google Map API to collect the distance and cost information between the cities from user request. The user request module is implemented based on the Python Flask and served as the backend of the trip planner web application.

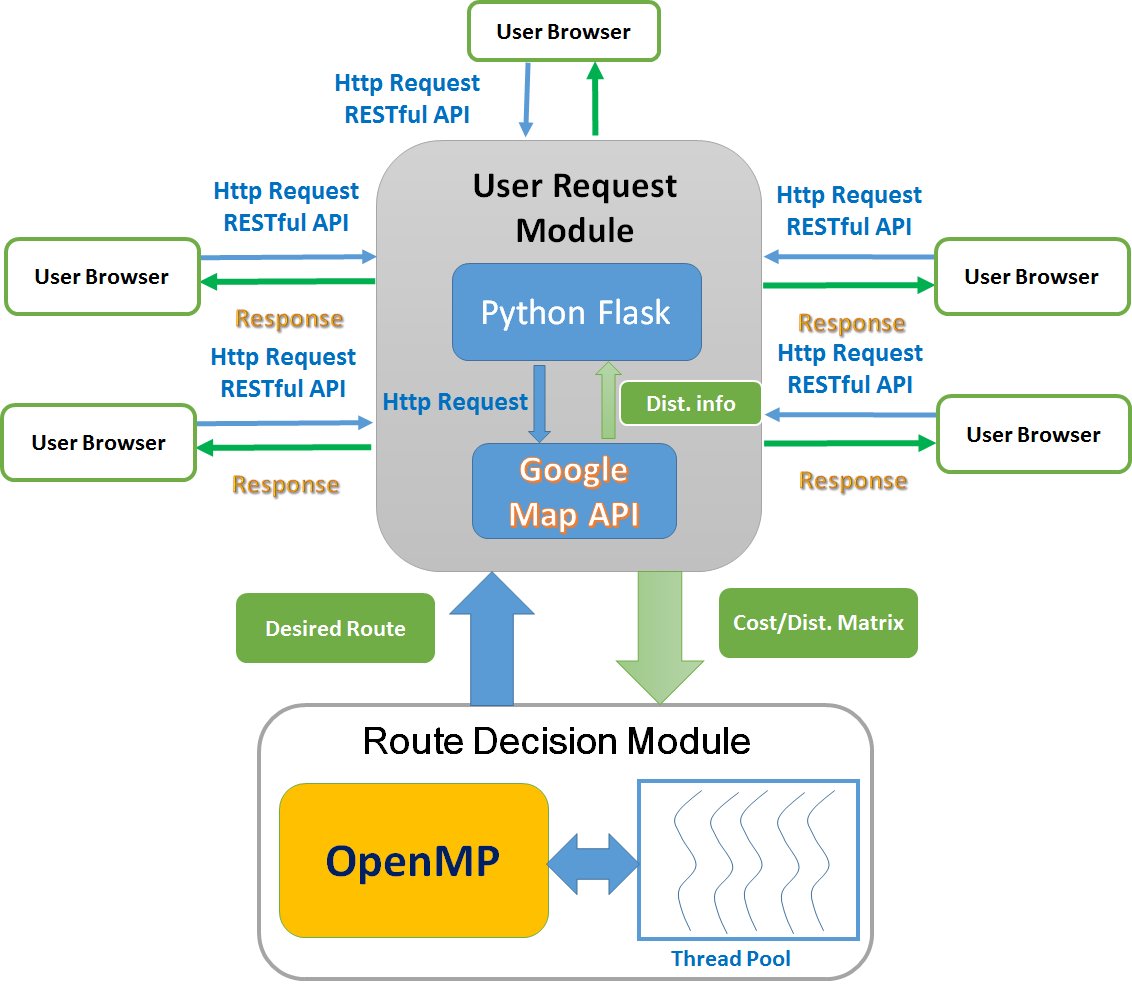


Fig 2 System Architecture Overview

These two modules communicate via the cost/distance information collected by the user request module. The route decision module will read cost/distance information in form of the cost matrix file which contains the matrix size and the distance, time or road construction information required for the TSP route calculation. The route module is implemented with a fully parallelized dynamic programming algorithm for TSP which is C++ code version based on OpenMP. The server side route decision module will send the determined route results back to the user request module.

The TSP solving procedure is highly time consuming because the algorithm running time for solving TSP increases almost exponentially with the number of cities. The algorithm is highly suitable for the parallelization. The module is developed using the OpenMP and is connected and invoked by the web application backend based on Python Flask. The users input is passed by python, and the TSP module is invoked and returns the desired route results to the web application and then the user browser for demonstration.

**Architecture analysis: pros and cons**

The system design could provide high stability, availability and future scalability.

* The loosely coupled constructure make sure only the maintenance of the python request processing module is needed while the compiled TSP server code is not required to be compiled again, and it is easy to deploy to more servers when the upscale is required for processing large amount of requests.
* The loosely coupled structure is easy to implement and suitable for prototype system development.
* The system also dispatch the task by calling the Google Map API which is highly reliable. The application design is scalable by dynamically adding servers when the massive requests comes in simultaneous.

However, the module communication using a matrix file may introduce the disk I/O overhead and cause delay when there are large amount of concurrent incoming user requests. The method to improve is directly retrieving the matrix information from the Python Flask’s backend using the TSP OpenMP program.

# 4. Components Implementation

The system has two major modules, user request module and route decision module. These two modules communicate via the cost/distance information collected by the user request module.

4.1 User Request Module

The user request module is responsible for receiving and handling user requests which contain the city names, and it will also finally return the final “best” routes from different aspects. The user request module will send requests to Google Map API to collect the distance and cost information (which will be in form of matrix for the route decision module) between the cities from user request. The user request module is implemented based on the Python Flask and served as the backend of the trip planner web application.

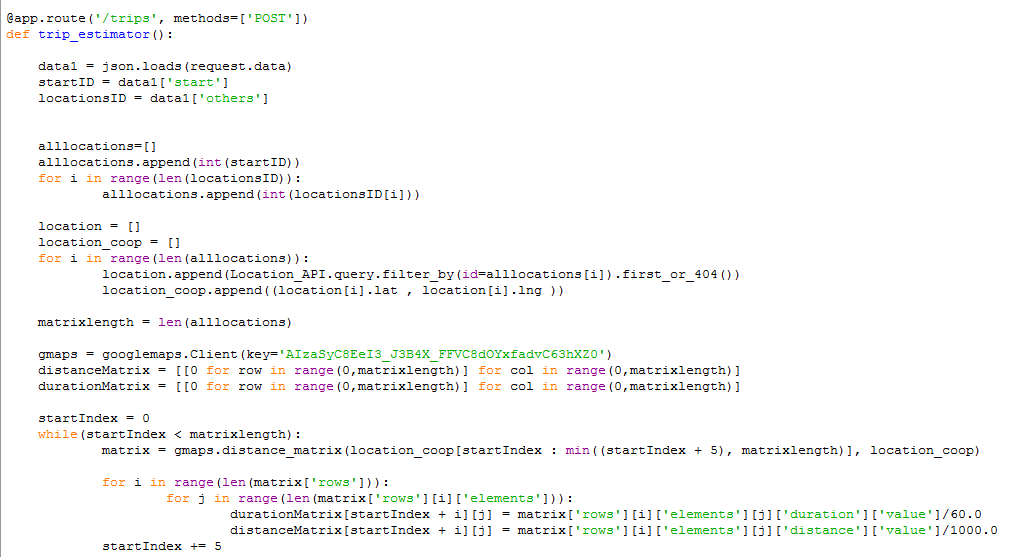


Fig 3 code for handling user request

4.2 Route Decision Module

The route module is implemented with a fully parallelized dynamic programming algorithm for TSP which is C++ code version based on OpenMP. The route decision module will read cost/distance information in form of the cost matrix file which contains the matrix size and the distance, time or road construction information required for the TSP route calculation. This module will return the desired route results to the web application’s user request module which will be responsible to respond to user.



Fig 4 code for route decision

4.3 Database Module

The user data and request history, along with the relevant city address data is saved to the MySQL database. The data attributes include name, address, city, state, zip, latitude and longitude. The web application support data retrieve, update, and replica.

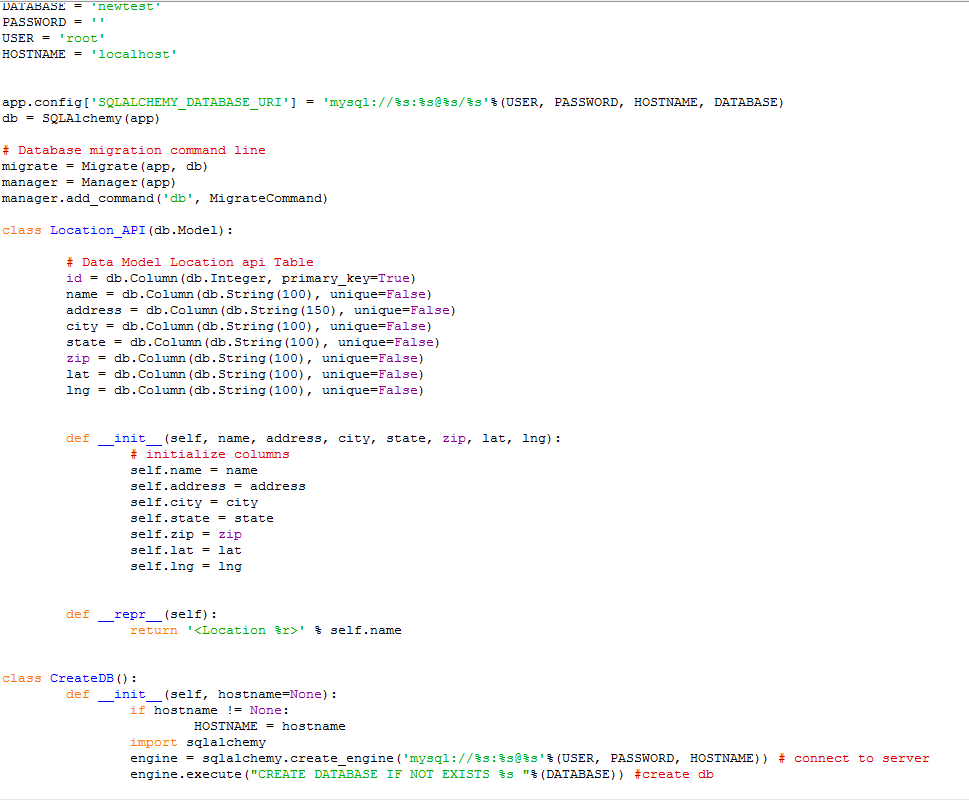


Fig 5 location information store procedure

# 5. Test Case

5.1 Frontend

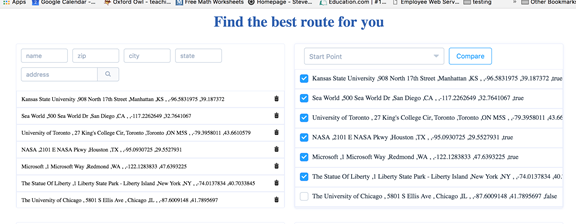




Fig 6 Trip planner web application frontend

5.2 Small input case (3 cities)

The following figure shows the best route found when the user input three city locations.

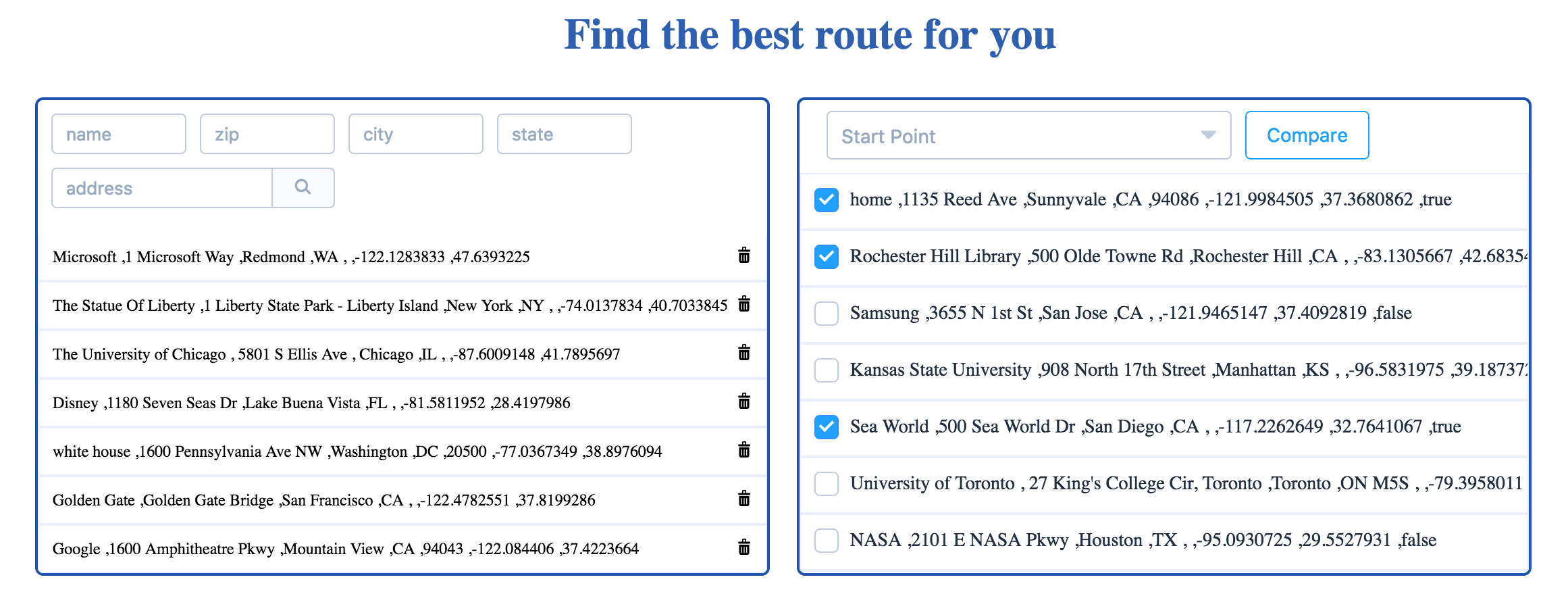




Fig 6 shortest route found in case user input 3 locations

5.3 Large input case (13 cities)

The following figure shows the best route found when the user input thirteen city locations.

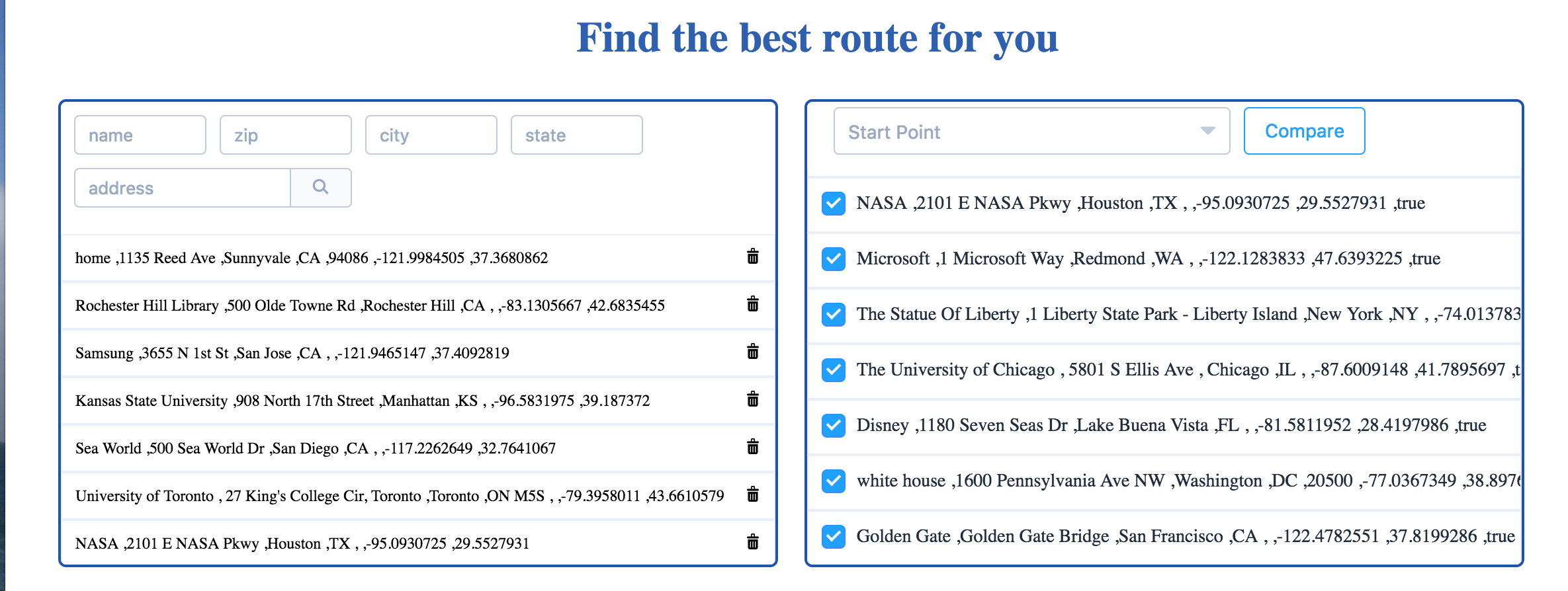




Fig 7 shortest route found in case user input 13 locations

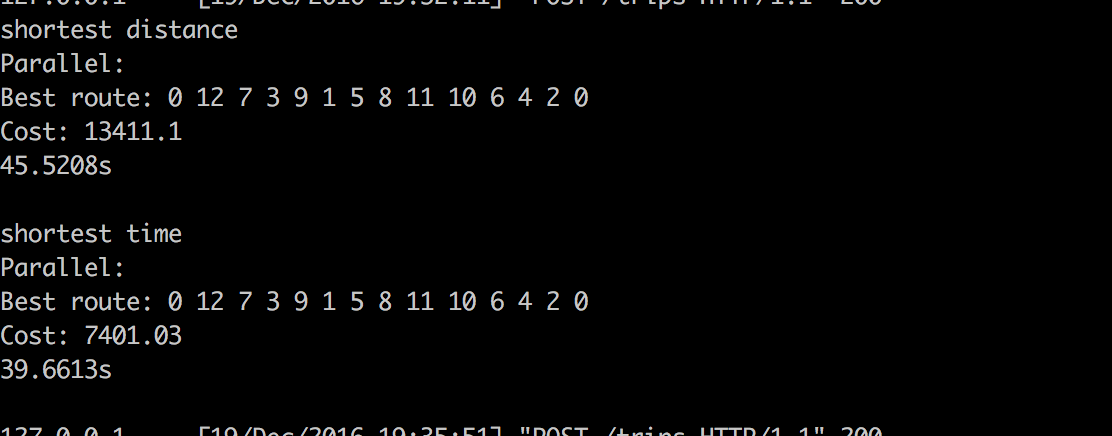


Fig 8 backend output screenshot for shortest route in case user input 13 locations

5.4 Performance measurement

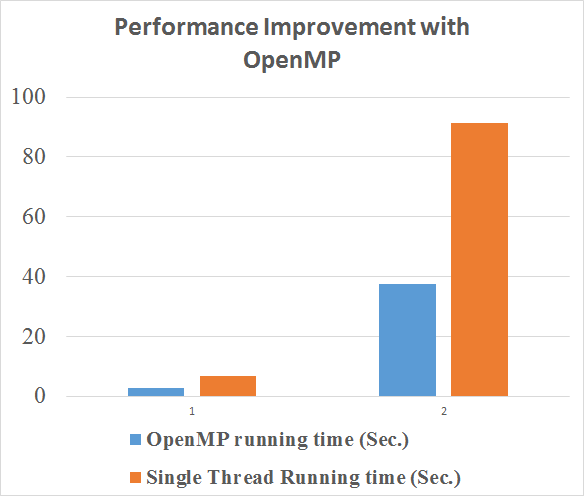


Fig 9 Performance improvements observed

The above figure shows the running time for 2 cases. The case 1 shows the performance comparison between TSP with OpenMP and without OpenMP, when the user request with 12 city locations. The case 2 shows the comparison with 13 city locations input. The OpenMP code has largely reduced running time needed. For case 1, the running time with OpenMP is only 2.75 seconds compared to without OpenMP’s 6.78 seconds, and the performance improved by 60.1%. The running time decreased from 91.41 seconds to 37.62 seconds for case 2.

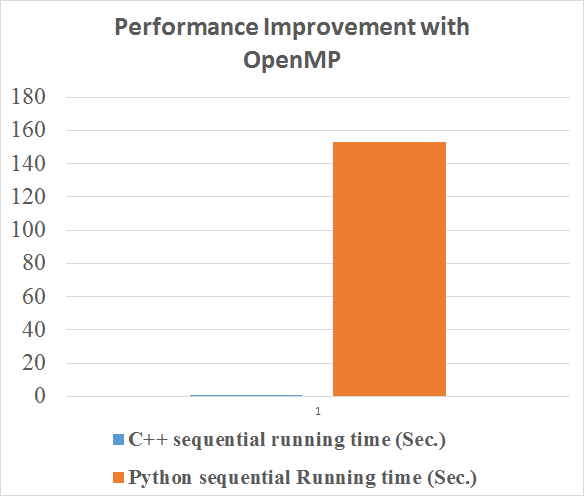


Fig 10 Performance comparison between C++ and Python TSP

The reason we choose C++ instead of Python for solving TSP is that we have measured and compared the performance from both C++ and Python. The above figure shows the Python sequential running time is more than 100 times than C++ sequential running time, in the case when the user input 8 city locations.

# 6. Conclusion

This project implemented a trip planner web application which not only tell the user the route which cover all the destinations he want to travel exactly once but also give the best routes as long as the relevant information is available. We implement a parallelized dynamic programming algorithm module for TSP which is C++ code version. The module is developed using the OpenMP and is connected and invoked by the Python Flask based web application. The shorter response time and performance improvement has clearly been observed which show the OpenMP played a key role and help improve the trip planner web application.