CSE 6010 Project Final Report: Traveler Planning Problem

Linyu Yao, Taoyouwei Gao, Hanyuan Zhang

1. Project Introduction

In this project, we try to solve a traveler planning problem through a model of undirected weighted graph. It is assumed that a man plans to travel around several cities, return the original city, and only visit every city for once. There are three transportation methods that he can choose including: flight, railway, and highway. Our goal is to choose a most economical transportation method between every two cities according to different money and time cost weight. And we will calculate the shortest path for the whole trip. In other words, we can get a travelling route, which is the most efficient or economical consistent with users' requirement.

The several travel cities are set as Atlanta, Los Angeles, Seattle, New York, Washington, Boston, Chicago, Hawaii, San Francisco, Las Vegas, Houston, Philadelphia, San Diego, and Miami. Firstly, we read three files including: 1). latitude and longitude of every city, 2). railway and 3). highway distance between every two cities. The main function of this program is to calculate the flight distance between every two cities according to the known latitude and longitude documents. Output of this code will be one file that contains the flight, railway, and highway distance between every two cities.

Then we calculate the time and money cost between each two cities through the output file of part two. On the basis of an external input parameter weight, we acquire a most suitable transportation method between each two cities. The output of this part is a file contains the distances among all cities, which are consistent with users' requirement. Last, we create an undirected weighted graph and calculate the shortest path to go through all vertexes. Cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. The output is a shortest path for this travel.

Our works includes:

- Literature search to learn Traveling Salesman Problem and NP problem
- Realize algorithm in C and debug
- Test the Part Three Code
- Analyze the results and draw some conclusions

2. Literature Research

TSP (Traveling Salesman Problem) can be modelled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. It is a minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once. Often, the model is a complete graph (i.e. each pair of vertices is connected by an edge). If no path exists between two cities, adding an arbitrarily long edge will complete the graph without affecting the optimal tour. It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

NP-hardness (non-deterministic polynomial-time hardness), in computational complexity theory, is the defining property of a class of problems that are, informally, "at least as hard as the hardest problems in NP". More precisely, a problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H; that is, assuming a solution for H takes 1 unit time, we can use H's solution to solve L in polynomial time. As a consequence, finding a polynomial algorithm to solve any NP-hard problem would give polynomial algorithms for all the problems in NP, which is unlikely as many of them are considered hard.

Combinatorial optimization is a subset of mathematical optimization that is related to operations research, algorithm theory, and computational complexity theory. It has important applications in several fields, including artificial intelligence, machine learning, auction theory, and software engineering.

3. C program function list

3.1 Part One

Function Prototype	Description
typedef struct {} city	Define a structure city which contains name, latitude. longitude and distance
city *malloc_matrix(city *C,int N)	Allocate memory for the matrix city
void free_matrix(city *C,int N)	Free memory of the matrix city
void storeFile(city *C,int N)	Write the results of distances between each two cities to a file named Distance_air
int main()	Read the file contains latitude and longitude information of all cities; calculate distances between each two cities according this; utilize the malloc & free function
int out(city *C,int N)	Print results of distance

3.2 Part Two

Function Prototype	Description
typedef struct {} city	Define a structure city which contains name, time, money and distance

city *malloc_matrix(city *C,int N)	Allocate memory for the matrix city
void free_matrix(city *C,int N)	Free memory of the matrix city
city *read_distance(city *C,char *way)	Read the distance file of different transportation methods
city *read_money(city *C,char *way)	Read the money file of different transportation method
void storeFile(city *C,int N)	Write the results of distances between each two cities after acquiring the suitable transportation method between them to a file named bestDistance
city *calculate_time	Calculate the time needed to travel around these cities according to the distance and speed of different methods
int main(int argc, const char * argv[])	According to the money and time cost weight to choose the most suitable transportation method between each two cities and put the distance results into a file

3.3 Part Three

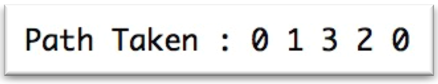
Function Prototype	Description
void copyToFinal(int *curr_path)	Copy the current solutions to final path
int firstMin(double adj[N][N], int i)	Find the minimum of the one city's neighbor
int secondMin(double adj[N][N], int i)	Find the second minimum of the one city's neighbor

void TSPRec(double adj[N][N], double curr_bound, double curr_weight, int level, int curr_path[])	Realize the algorithm to find the shortest path
void TSP(double adj[N][N])	Calculate the start bound; update shortest path using a recursion version of branch and bound
int main(int argc, const char *args[])	Allocate memory for final path; allocate memory for visited array; create a 2D array to store the input value; utilize the previous funciton

4. Test of Part Three Code

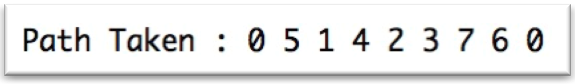
We use a small number of cities to generate several results of shortest path and verify that our program is correct.

1. N = 4; weight = 0, 0.2, 0.5 & 1.0



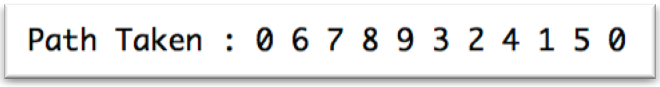
```
Path Taken : 0 1 3 2 0
```

2. N = 8; weight = 0, 0.2, 0.5 & 1.0



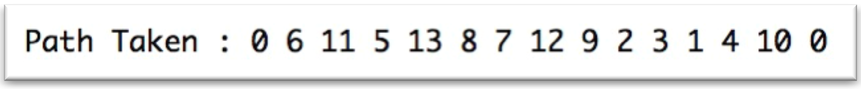
```
Path Taken : 0 5 1 4 2 3 7 6 0
```

3. N = 10; weight = 0, 0.2, 0.5 & 1.0



```
Path Taken : 0 6 7 8 9 3 2 4 1 5 0
```

4. N = 14; weight = 0, 0.2, 0.5 & 1.0



```
Path Taken : 0 6 11 5 13 8 7 12 9 2 3 1 4 10 0
```

5. Analysis

5.1 Different weight of money and time cost

1. weight = 0 (which means time weighs 0 and money weighs 1)

Transportation Method	Number
Airplane	59
Car	32
Train	0

2. weight = 0.1(which means time weighs 0.1 and money weighs 0.9)

Transportation Method	Number
Airplane	73
Car	18
Train	0

3. weight = 0.2(which means time weighs 0.2 and money weighs 0.8)

Transportation Method	Number
Airplane	77
Car	14
Train	0

4. weight = 0.3(which means time weighs 0.3 and money weighs 0.7)

Transportation Method	Number
Airplane	81

Car	10
Train	0

5. weight = 0.5(which means time weighs 0.5 and money weighs 0.5)

Transportation Method	Number
Airplane	86
Car	5
Train	0

6. weight = 0.6, 0.7, 0.8, 1.0

Transportation Method	Number
Airplane	91
Car	0
Train	0

5.2 Time and Money Cost for Traveling 14 cities

1. weight = 0 (which means time weighs 0 and money weighs 1)

The sum of time cost for the shortest path is 68.7h, and the sum of money cost is $1543.78.

2. weight = 0.1 (which means time weighs 0.1 and money weighs 0.9)

The sum of time cost for the shortest path is 57.56h, and the sum of money cost is $1597.82.

3. weight = 0.2 (which means time weighs 0.2 and money weighs 0.8)

The sum of time cost for the shortest path is 42.3h, and the sum of money cost is $1634.39.

4. weight = 0.5 (which means time weighs 0.5 and money weighs 0.5)

The sum of time cost for the shortest path is 37.85h, and the sum of money cost is $1729.59.

5. weight = 1 (which means time weighs 1 and money weighs 0)

The sum of time cost for the shortest path is 37.85h, and the sum of money cost is $1729.59.

The following is two diagrams which represent the relationship between weight, time and money cost.

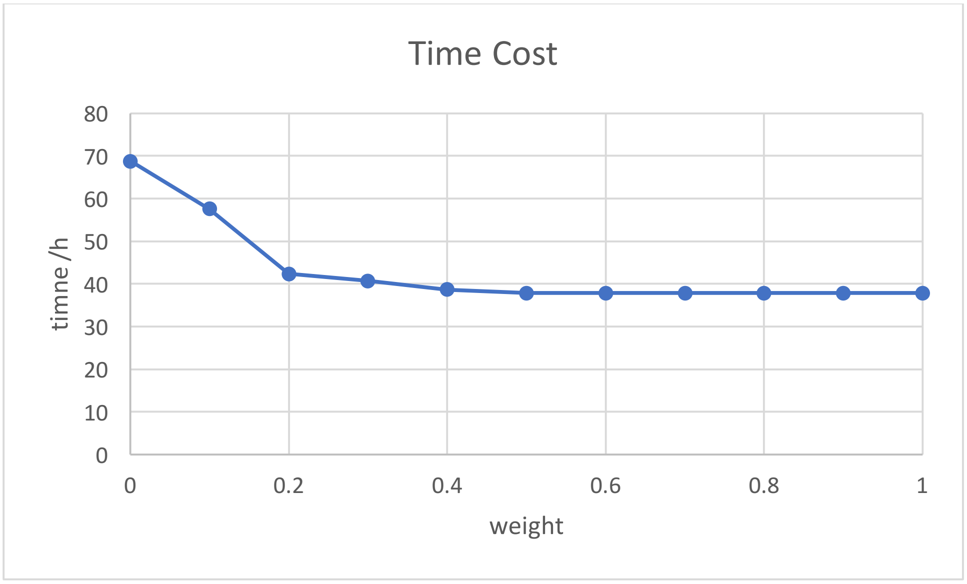


diagram 1: relationship between weight and time cost

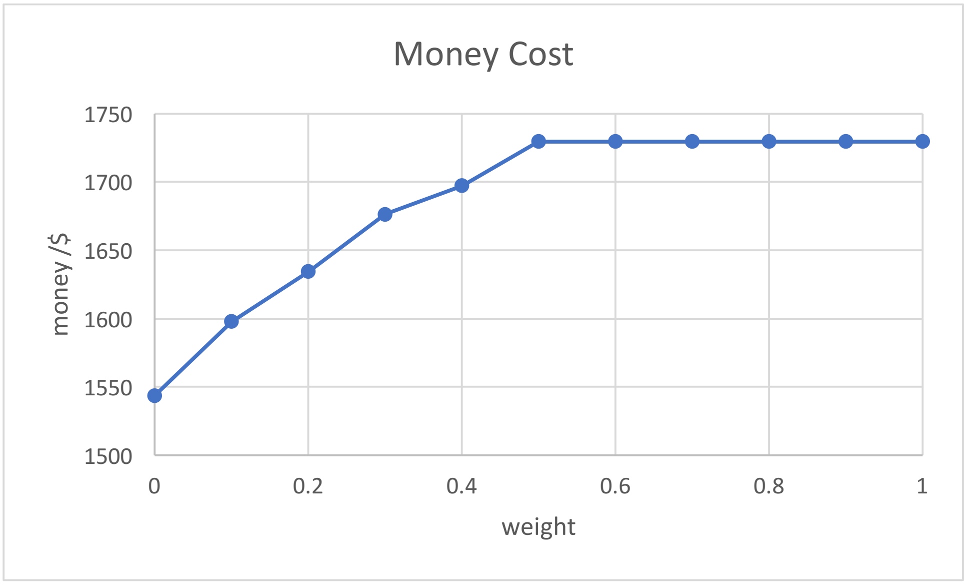
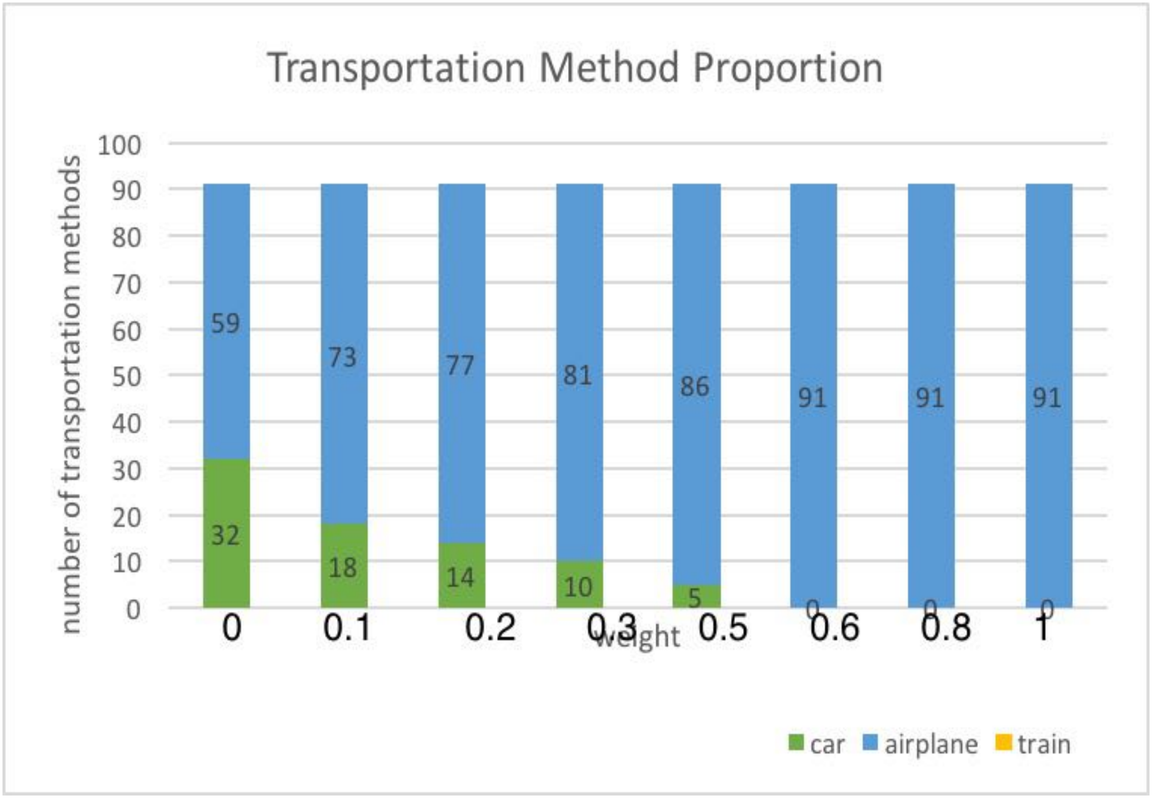


diagram 2: relationship between weight and time cost

Thus, it is easy to conclusion that when the weight is a small value, which means the users care more about money, the time cost is more and money cost is smaller. When weight increases, which means people care more time, the time cost decreases and money cost increases. And when weight is over 0.6, the time and money cost stay unchanged because all routes choose the airplane as transportation method.

5.3 Transportation Method Proportion



We can draw a conclusion that when users are concerned more about money, it is recommended to utilize cars as transportation method. If users care more about time, we suggest them to use airplane. As for train, it is both expensive and time costive so we do not recommend this method.