CS512 Project: Traveling Tourist Problem

Serge Kas Hanna, Juntao Tan, Vatsal Parikh
Rutgers University, New Brunswick, NJ, USA

***Abstract*— Tourism is one of the biggest industries in the world, and tour planning and tour arrangement is an essential aspect to consider for tourists and tourist agencies alike. In this project, we design an algorithm that helps a tourist find the most optimum path to visit tourist attractions in a particular city or among many cities based on the preferences (like budget) of the tourist.**

I. Project Description

In this project a modified version of the traveling salesman problem (TSP) is proposed. The TSP asks the following question "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?". The goal in our project is to answer the following question "Given a list of cities and (i) distances between each pair of cities; and (ii) a touristic characteristic of each city, what is the best possible route that a tourist should take subject to time and money constraints?". We call this problem the traveling tourist problem (TTP). The route has to end at the location from which the tour had started (e.g., hotel, train station). The touristic characteristic of a city is a score that captures the value of multiple features of the city, such as: popularity, cost of visiting (time and money), etc.

To the best of our knowledge the modification that we make to the TSP problem is novel. The TTP may be interesting to many tourists or touristic agencies who want to plan a tour based on (i) some constraints that the tourist sets, and on (ii) the characteristics of the visited cities. For instance, not all cities have the same importance or popularity in terms of touristic activities. The goal is to design an algorithm that finds the best possible route based on the tourist's constraints and geographical characteristics.

Several stumbling blocks could arise in this project. First, modeling the problem using a graph is by itself a challenging task, given that several factors (i.e., constraints) are simultaneously in effect. Second, formulating an evaluation or utility function that adequately represents the needs of the user. Third, designing a simple yet powerful user interface.

The project has four stages: Gathering, Design, Infrastructure Implementation, and User Interface.

A. Stage1 - The Requirement Gathering Stage.

- The general system description: This project intends to plan a tourist route for users based on constraints like time and money. At the beginning of the planning system, the user needs to input a specific location from which the tour starts (e.g., hotel, train station). Then, the system will run an algorithm and output an optimal touring path based on the specified constraints. To this end, a cost value is assigned to each attraction. There are multiple factors which contribute toward determining this cost value. Next, we show a simplified example covering four tourist attractions within the city of New York.

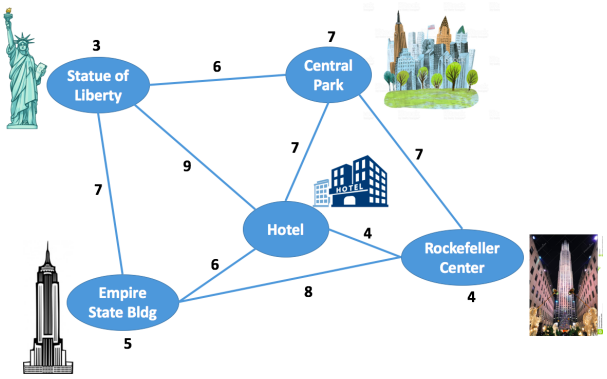

Fig. 1. Simplified modeling of the TTP in NYC using a graph.

In the example above the weights of the edges are based on the distances between the touristic attractions. If the tour starts at the hotel, then the closest venue that the tourist can first visit is the Rockefeller center. The major difference in the TTP as compared to the TSP is the fact that the vertices of the graph also have weights. Here, Rockefeller center has weight of 4. The weights of the vertices model the cost of visiting a certain tourist attraction. Several factors contribute in determining the weights of the vertices. For example, the time needed to explore the attraction and the amount of money needed to access the attraction are two factors that increase the weight of a certain vertex (i.e., increase the cost). Whereas, the popularity of a certain venue decreases the cost of visiting the attraction (i.e., weight of the vertex), since the more popular the attraction is, the more satisfied the tourist is. Modeling the TTP in an adequate manner is part of our project and beyond the scope of this submission (Stage 1). Nevertheless, we provided the example above in an attempt to convey the high-level idea of the problem.

- Real world scenarios:
  - Scenario1 description: A tourist wants to visit New York City in one day with no money constraints.
  - System Data Input for Scenario1: City(New York), money constraint(infinity), time constraint(1 day).
  - Input Data Types for Scenario1: String and integer
  - System Data Output for Scenario1: A sequences of

vertices by formulating cost as minimum value for money, default value for popularity, and high value for time.
  - Output Data Types for Scenario1: String and integer
  - Scenario2 description: A tourist wants to visit popular and major tourist attractions of New York City but spending power is limited.
  - System Data Input for Scenario2: City(New York), money constraint(limited), time constraint(default).
  - Input Data Types for Scenario2: String and integer
  - System Data Output for Scenario2: A sequence of vertices by formulating cost as maximum value for money, low value for popularity, and default value for time.
  - Output Data Types for Scenario2: String and integer
- Tentative Project Time line. The time line that we expect to follow for project is as follows-
  1) Stage1 (Project Description) - 17 October.
  2) Stage2 (Project Design) - 31 October.
  3) Stage3 (Implementation and Testing) - 14 November.
  4) Stage4 (User Interface) - 28 November.
  5) Evaluation, project report and presentation - First week of December.
- Division of Labor.
  1) Stage 1
     - Vatsal Parikh - Writing project description, project time line and division of labor.
     - Serge Kas Hanna - Writing project description and providing an example to explain the project description.
     - Juntao Tan - Writing general system descriptions, Specifying constraints, detailing user types and real world scenarios.
  2) Stage 2
     - Vatsal Parikh - Short textual project description and flow diagram.
     - Serge Kas Hanna - High level pseudo code system description
     - Juntao Tan - Specifying algorithms and data structures used, flow diagram, major constraints.
  3) Stage 3
     - Vatsal Parikh - Testing and documentation
     - Serge Kas Hanna - Implementing algorithm and displaying results
     - Juntao Tan - Implementing algorithm and displaying results.
  4) Stage 4
     - Vatsal Parikh - Designing navigation and interaction screen.
     - Serge Kas Hanna - Designing welcome screen.
     - Juntao Tan - Testing interface and documentation

B. Stage2 - The Design Stage.

- Short Textual Project Description: In what follows we provide a textual description of the algorithm.

1) Input: The weights of the edges follow from geographical characteristics and are part of the input. The user's input is divided into two parts. Firstly, the user inputs percentage values to express preferences for the tour. For example, a user that is mainly interested in visiting the most popular attractions and does not care much about the time and the money spent at an attraction, may input 90% popularity, 5% time, 5% money. We use these values to determine the weights of the vertices. So in this case we favor a popular attraction by giving it a low weight (cost). Secondly, the user inputs overall time and money constraints. For instance, here a user might input 10 hours and 900 USD. We use these values to check throughout the algorithm (on the fly) whether a certain tour is affordable or not.

2) *Algorithm and Output:* To the best of our knowledge, the algorithm that provides an exact solution to the TSP is a dynamic programming algorithm, called Held-Karp algorithm. To solve the TTP, we make some modifications to the Held-Karp algorithm. These modifications are based on two main differences between the TSP and the TTP:

(a) The vertices of the TTP graph have weights. We consider the asymmetric version of the Held-Karp algorithm. Let $\ell(u, v)$ be the weight of edge $(u, v) \in E$. In the TTP, we quantify the cost of going from u to v, through edge (u, v), by the value $\ell(u, v) + w(v)$, where $w(v)$ is the weight of vertex v. This definition takes into account only the weight of vertex v (excludes $w(u)$) because we consider that the tourist is already at u. So the cost of going to v is measured by the traveling distance (i.e., $\ell(u, v)$), in addition to the cost of visiting attraction v (i.e., $w(v)$).

(b) The optimal "affordable" tour may not pass through all the vertices. In the TSP we look for the optimal path that passes through all the vertices, whereas in the TTP such a path might not be affordable. By the notion of affordability we refer to the scenario where the constraints set by the user (described previously) make it impossible for the tour to cover all the attractions. The Held-Karp algorithm solves the TSP by solving subproblems corresponding to the optimal path that passes through a subset $S \subseteq \{1, \ldots, n\}$ of vertices, $|S| = 2, \ldots, n$. We leverage this idea to find a solution for the TTP. For instance, assume that the optimal solution for the TTP contains m vertices, where $1 \leq m \leq n$. Therefore, in this case it is easy to see that the solution of the TTP is equivalent to the solution of the TSP subproblem that corresponds to $|S| = m$.

That said, the problem boils down to determining the value of m. To this end, we run the dynamic programming algorithm in the linearized order of the sizes of the subsets $|S| = s$, $s = 2, \ldots, n$. If for some value s_0 of $|S|$, we find that the *minimum* cost over all subsets of size s_0 is not affordable, (i.e., every subset of size s_0 is not affordable), we terminate the algorithm and ouput the optimal affordable path among the subsets of size $m = s_0 - 1$.

- High Level Pseudo Code System Description. Next, we

provide a pseudo code of the algorithm. We divide the pseudo code into three parts. In the first part we describe how we solve the TTP given the graph and the user's money and time constraints. We follow this part by a discussion on the time and space complexity of the algorithm. In the second and third parts, we describe how to construct the graph (i.e., assign the weights of vertices and edges).

Definitions: Let $G(V, E)$ be the graph which models the TTP. The size of the vertex set is $|V| = n$. G is a complete graph, hence $|E| = \binom{n}{2}$. $\ell(u, v)$ is the weight of $(u, v) \in E$; and $w(v)$ is the weight of vertex v. Let $C(S, i)$ be the cost of the shortest path that visits each vertex in set $S \subseteq \{1, \ldots, n\}$ exactly once, starting at vertex 1 and ending at vertex i. The variable $m, 1 \leq m \leq n$ represents the number of vertices in the optimal path of the TTP. The value of m is initialized to n, and as previously explained the optimal value of m is determined by the algorithm.

Algorithm 1: TTP algorithm pseudo code

input : Graph $G(V, E)$, $|V| = n$,
User constraints: money, time
output : Optimal affordable path

$C(\{1\}, 1) = 0$;
$m = n$;
for $k = 2$ **to** n **do**
  $C(\{k\}, k) = \ell(1, k) + w(k)$;
end
for $s = 2$ **to** n **do**
  for *all subsets* $S \subseteq \{2, \ldots, n\}$ *with* $|S| = s$ **do**
    for *all* $i \in S$ **do**
      $C(S, i) = \min_{j \in S, j \neq i} \{C(S \backslash i, j) + \ell(i, j) + w(i)\}$
    end
  end
  if $\min_{i \in S} C(S, i) >$ `budget`(money,time) **then**
    $m = s - 1$;
    break;
  end
end
return $\max_{i \in S, |S| = m} \{C(S, i) + \ell(i, j) + w(i)$ within `budget`$\}$

Time and space complexity: The dominant factor in the complexity of the TTP (or TSP) is the number of subproblems that the dynamic programming algorithm considers. For instance, the total number of subsets of $\{1, \ldots, n\}$, of sizes $s = 2, \ldots, n$, is

$$\sum_{s=2}^{n} \binom{n}{s} = \mathcal{O}(2^n).$$

So assuming that $m = n$ for a certain instance of the TTP, it is easy to see that the number of subproblems is $\mathcal{O}(n2^n)$. Hence, the space complexity is $\mathcal{O}(n2^n)$, and the time complexity is $\mathcal{O}(n^2 2^n)$ because each subproblem can be solved in linear time. However, unlike the TSP, the optimal path of the TTP may not pass through all the vertices, i.e., $1 \leq m \leq n$. Therefore, the complexity of the TTP varies according to m. For instance, if $m = \mathcal{O}(1)$ (i.e., constant), then

$$\sum_{s=2}^{m} \binom{n}{s} = \mathcal{O}(n^m),$$

and as a result we obtain a polynomial complexity. The regime where $m = \mathcal{O}(1)$ corresponds to the scenario where the tourist sets heavy constraints, and thereby the number of attractions that the optimal tour can cover is very small. In conclusion, the worst-case complexity is exponential, but a polynomial complexity is possible in some scenarios. The table below summarizes the space and time complexity of the TTP algorithm for some different regimes of m.

	Time Complexity	Space Complexity	Order
$m = \mathcal{O}(1)$	$\mathcal{O}(n^{m+2})$	$\mathcal{O}(n^{m+1})$	Polynomial
$m = \mathcal{O}(\log n)$	$\mathcal{O}(n^{\log n+2})$	$\mathcal{O}(n^{\log n+1})$	Quasi-polynomial
$m = \mathcal{O}(n)$	$\mathcal{O}(n^2 2^n)$	$\mathcal{O}(n2^n)$	Exponential

TABLE I
COMPARING THE TIME AND SPACE COMPLEXITY OF THE PROPOSED TTP ALGORITHM FOR DIFFERENT REGIMES OF m.

Next, we provide the pseudo codes for calculating the weights of the vertices and the edges of the graph. This part of the algorithm is relatively simple, and the main idea we use is to normalize the raw values collected from the database in order to remove the bias. For instance, the money cost of an attraction could be represented in the database by the entrance fee in USD, while the popularity may be measured by the number of people who visited that attraction in a particular year, so using raw values would create a bias. Therefore, if the entrance fee for a certain attraction is 20 USD, we divide this value by the *maximum* entrance fee among all the attractions. Thereby, we obtain a value between 0 and 1 (the attraction that has the maximum entrance fee will be assigned a money cost of value 1). We do the same for all the characteristics (money, popularity, time, distance), i.e., divide the raw value by the corresponding maximum.

Definitions: $X[n]$ represents the actual raw data for popularity which is quantified by the number of tourists that visited the attraction in a certain year. $B[n]$ represents the corresponding normalized values, and $P[n] = 1 - B[n]$. $P[n]$ reflects the actual normalized scores for popularity, since low cost refers to high popularity. Similarly we have $Y[n]$ (raw data) and $M[n]$ (normalized) for the money needed to visit an attraction (i.e., entrance fee). Also, $Z[n]$ (raw data) and $T[n]$ (normalized) for the time needed to visit an attraction. The final resulting vertex weights $w(v), v = 1, \ldots, n$ is a weighted linear combination of $P[n], M[n]$ and $T[n]$. As mentioned previously, the weighting coefficients are based on the user's preferences (e.g., 90% popularity, 5% time, 5%

money). We also apply a similar approach for the weights of the edges.

Algorithm 2: Pseudo code for calculating the normalized weights of the vertices $w(v)$

input : $X[n], Y[n]$, and $Z[n]$,
Weighting coefficients: a, b, and c
output : Vertex weights $w[n]$

for $i = 1$ **to** n **do**
  $B(i) = X(i)/\max_i\{X(i)\}$;
  $P(i) = 1 - B(i)$;
  $M(i) = Y(i)/\max_i\{Y(i)\}$;
  $T(i) = Z(i)/\max_i\{Z(i)\}$;
  if *default-case* **then**
    $w(i) = (P(i) + M(i) + T(i))/3$;
  else
    $w(i) = aP(i) + bM(i) + cT(i)$;
  end
end

Algorithm 3: Pseudo code for calculating the normalized weights of the edges $\ell(u, v)$

input : Adjacency matrix $A[n, n]$ having raw data
output : Normalized edge weights $\ell(u, v), (u, v) \in E$

for $i = 1$ **to** n **do**
  for $j = 1$ **to** n **do**
    $\ell(i, j) = A(i, j)/\max_{i,j} A(i, j)$;
  end
end

- Flow Diagram.
  Flow diagram textual description is as follows:
  u is the popularity measure in percent input by user.
  v is the affordability measure in percent input by user.
  w is the time spent at attraction in percent input by user.
  n is the total number of attractions.
  x, y, z values are actual values of popularity, affordability and time to be spent of an attraction respectively.
  e is the normalized edge weight value between 0 and 1 from i th vertex to every other vertex.
  p, a, t are the normalized values for popularity, affordability and time spent at the attraction (between 0 and 1) for every i node in n.
  C(s,i) - cost of most optimum tour for s subset of nodes.
  S is number of possible paths available for a subset of s nodes.

- Algorithms and Data Structures.
  As mentioned above, the main algorithm used in the program is similar to Held-Karp but with some adjustments to satisfy the constraints input by the user. At the same time, it not only counts the weight of edges but also counts the weight of each vertex.
  The data structures used in the algorithm are as follows:
  – Adjacency matrix (2-Dimension array): This data structure is used to represent the map of attractions in the city. Since each two attractions in the map can reach each other directly, this graph is a complete graph (i.e., with high density). That's why we choose adjacent matrix to represent this graph. Each value [A][B] in the matrix indicates the cost from attraction A to B (weight of the edge from A to B).
  – Hash map: For each step, like C(SET, A) = min(SET - i + W (i, A)) we need a hash map in which using a vertex and its subsets as indicator can return the value of optimal path.
  – First 1-Dimension array: Use a 1-D array to store the path (tour) in our program. For an attraction B in this array, previous element of B is the last attraction before visiting B, and next element of B is the attraction visited after B

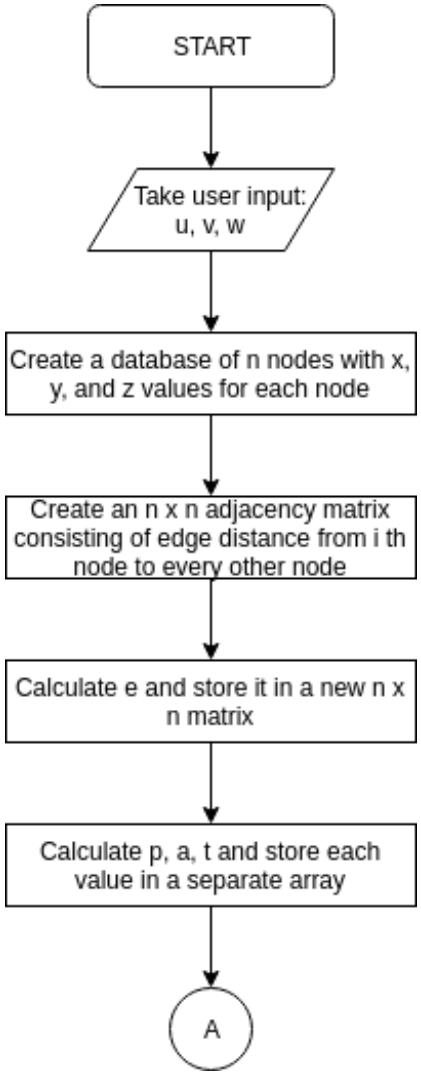

Fig. 2. TTP Flow Diagram 1.

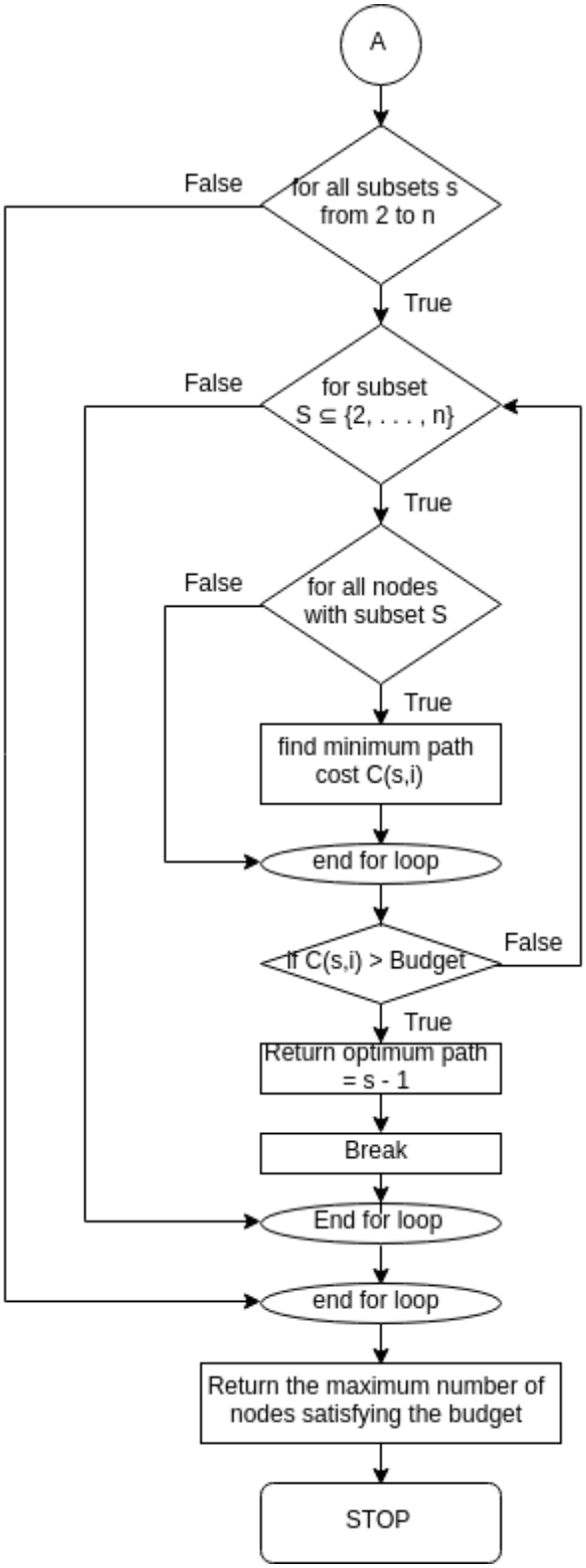

Fig. 3. TTP Flow Diagram 2.

in this path.

– Second 1-Dimension array: This array is used to store the cost values of all attractions in this city ($e.g., a[4] = 0.8$ indicates the cost of the 4th attraction as 0.8).

- Flow Diagram Major Integrity Constraints.
  – Integrity constraint of data input-
  At the beginning of the program, the user should input three things:
  City: Which city he/she wants to visit.
  Money constraint: How much money he/she can afford at most when traveling in this city?
  Time constraint: What is the time limitation when traveling in this city?
  The city cannot be null, because only after knowing which city the user wants to visit can the program run the algorithm in this city and find out a route.
  Money and time can be null. If no money/time is input by the user, it indicates that there's no money/time constraint (i.e., the limitation of money/time is infinite).
  – Integrity constraint of database-
  We have three main tables in our database. For each table, the integrity constraints and descriptions are as follows:
  1) TABLE CITY
  CITY_ID CHAR(9) NOT NULL PRIMARY KEY
  CITY_NAME VARCHAR(20) NOT NULL
  PRIMARY KEY (CITY_ID)
  );
     * Not null constraints: Obviously the name of cities in the database cannot be null.
     * Primary Key constraints: The primary key in this table is CITY_ID. Since different cities may have different names, so we need to use CITY_ID to identify different cities.
  2) TABLR ATTRACTION
  (
  ATTRACTION_ID CHAR(9) NOT NULL PRIMARY KEY,
  ATTRACTION_NAME VARCHAR(20) NOT NULL
  CITY_ID CHAR(9) NOT NULL
  COST_TIME INT(5) NOT NULL
  COST_MONEY INT(10) NULL
  POPULARITY INT(10) NOT NULL
  PRIMARY KEY (ATTRACTION_ID)
  CONSTRAINT CITY_ID FOREIGN KEY (CITY_ID) REFERENCES CITY (CITY_ID)
  );
     * Not null constraints: Popularity cannot be null, it's because if a attraction has no popularity, it won't be visited any time. Cost_time cannot be null, it's because each attraction at least need to cost some time to visit. Cost_money can be null, it's because many attractions do not ask for a ticket or something that cost money.
     * Primary Key Constraints: Different attractions may have same name, so we need a city_id to identify different attractions. And for each city_id, all other informations in this table can be decided.
     * Foreign Key Constraints: City_id is the foreign

key in this table. It means each attraction must be in a city which is exist in the city table.

3) TABLE DISTANCE
(
ATTRACTION_ID1 CHAR(9) NOT NULL
ATTRACTION_ID2 CHAR(9) NOT NULL
DISTANCE CHAR(4) NOT NULL
PRIMARY KEY (ATTRACTION_ID1, ATTRACTION_ID2)
);
   * Not null constraints: The distance between two attractions can't be null.
   * Primary Key Constraints: We combine Attraction_ID1 and Attraction_ID2 as one primary key to indicate the distance between this two attractions.

C. Stage3 - The Implementation Stage.

We used Python to implement the TTP algorithm that we previously explained. The IDE software we used is PyCharm [1]. Next we show a data sample that we constructed based on the characteristics of six touristic attractions in NYC. The first part of the data shown in Table II contains: (i) Number of tourists that visit the attraction per year (in millions); (ii) Time needed to visit the attraction (in hours); (iii) The required entry fee (in US dollars). The second part of the data shown in Table III contains the distances (in miles) between the different attractions, including the hotel that the tourist starts from, which we assume to be the "Pennsylvania Hotel" located in NYC. The information shown in Table II and Table III was collected through various online sources, the references [2]–[17] are provided in the end of this document.

Attraction	Number of visitors per year (millions)	Time spent (hours)	Entrance fee (USD)
Statue of Liberty	4	2	20
Times Square	50	2	0
Central Park	38	3	15
Metropolitan Museum of Art	7	3	0
Empire State Building	4	2	30
Brooklyn Bridge	0.33	1	0

TABLE II
TOURISTIC CHARACTERISTICS OF SIX ATTRACTIONS IN NYC.

Next we show three different output samples that were recorded based on the data shown in Table II and Table III. Each output corresponds to a different input, i.e., time and money constraints. The first output shown in Fig. 4 is for the default case (no constraints). The second output (Fig. 5) corresponds to the constraints: 8 hours, 40 USD. The third output (Fig. 6) corresponds to the time constraint of 4 hours without any money constraint.

Discussion. In the default case (Fig. 4), where there are no constraints on time and money, the optimal TTP path covers all the six attractions and the incurred time and money costs are 13 hours and 66 USD, respectively. In the second case (Fig. 5), the input constraints are 8 hours and 40 USD. Here, it is immediately clear that the TTP path for the default case is not "affordable". In fact, the optimal affordable path that the

Distances (miles)	Hotel	SL	TS	CP	MMA	ESB	BB
Hotel	0	5.5	1	2.5	3.5	1	3
SL	5.5	0	5	6	10	5.5	1.5
TS	1	5	0	1	2.5	1	4
CP	2.5	6	1	0	0.5	1.5	7.5
MMA	3.5	10	2.5	0.5	0	3	7.5
ESB	1	5.5	1	1.5	3	0	3
BB	3	1.5	4	7.5	7.5	3	0

TABLE III
DISTANCES (IN MILES) BETWEEN THE HOTEL AND THE ATTRACTIONS.

```
/Library/Frameworks/Python.framework/Versions/2.7/bin/
python2.7 /Users/serge/Downloads/TTP/TTP.py
Please input the time constraint in hours. If no time
constraint, input none
none
Please input the money constraint in USD, If no money
constraint, input none
none
TTP path in terms of the vertices of the graph :
[0, 6, 1, 5, 4, 3, 2, 0]
total cost [time (hrs), money (USD)] is [13, 66]
TTP path shown in terms of the names of the attractions:
Hotel Pennsylvania  -> Brooklyn Bridge -> Statue of Liberty
 -> Empire State Building -> Metropolitan Museum of Art ->
Central Park -> Times Square -> Hotel Pennsylvania

Process finished with exit code 0
```

Fig. 4. Output for the default case.

```
/Library/Frameworks/Python.framework/Versions/2.7/bin/
python2.7 /Users/serge/Downloads/TTP/TTP.py
Please input the time constraint in hours. If no time
constraint, input none
8
Please input the money constraint in USD, If no money
constraint, input none
40
TTP path in terms of the vertices of the graph :
[0, 6, 5, 4, 2, 0]
total cost [time (hrs), money (USD)] is [8, 31]
TTP path shown in terms of the names of the attractions:
Hotel Pennsylvania  -> Brooklyn Bridge -> Empire State
Building -> Metropolitan Museum of Art -> Times Square ->
Hotel Pennsylvania

Process finished with exit code 0
```

Fig. 5. Output for the input constraints: 8 hours, 40 USD.

```
/Library/Frameworks/Python.framework/Versions/2.7/bin/
python2.7 /Users/serge/Downloads/TTP/TTP.py
Please input the time constraint in hours. If no time
constraint, input none
4
Please input the money constraint in USD, If no money
constraint, input none
none
TTP path in terms of the vertices of the graph :
[0, 6, 5, 0]
total cost [time (hrs), money (USD)] is [3, 30]
TTP path shown in terms of the names of the attractions:
Hotel Pennsylvania  -> Brooklyn Bridge -> Empire State
Building -> Hotel Pennsylvania

Process finished with exit code 0
```

Fig. 6. Output for time constraint of 4 hours, with no money constraint.

algorithm finds covers four out of the six attractions and the corresponding time and money costs are 8 hours and 31 USD, respectively. In the third case (Fig. 6), the input consists of only a time constraint of 4 hours. In this case, the optimal TTP path outputted by the algorithm covers only three attractions and the time and money costs are 3 hours and 30 USD. The observed results show that, as expected, the heavier the constraints are, the fewer the number of visited attractions will be.

Running time: The running time for the default case is roughly 2.66 milliseconds. Whereas the running time for the second and third cases are 2.1 and 0.45 milliseconds, respectively. The numbers show that the running time decreases when the constraints are heavier. This is actually expected and validates the theoretical time complexity analysis discussed in the previous section (Table I).

Memory usage: After monitoring the RAM usage in the terminal, when running this project in the default case, which includes 6 attractions in the final route, it uses roughly 11mb of RAM. But when running the algorithm with constraints, the memory used doesn't reduce too much. It's because when stopping in the middle of the process, in order to decide which is the final route, extra memory is required to reserve all the combinations of a certain size of attractions' set as well as the minimum cost for each combination.

Novelty: The novelty of the outcomes obtained in this project can be explained as follows:

This application takes into consideration the preferences of user. It takes a granular view of the major factors a tourist looks into while touring a particular city, the popularity of attraction, the money needed and the time required. This application combines all these factors effectively and concisely and normalizes them to feed the data as input to the code. A regular TSP problem gives a path that visits all vertices once and returns back to the starting point. Whereas the novelty with TTP lies in the fact that we provide the tourist with the best possible attractions that can be visited in a certain amount of time, given constraints like popularity and money. We not only give tourist the number of attractions they should visit, but we also provide the sequence in which these attractions can be visited by optimizing for time, money, and distance.

Usefulness: The usefulness of the outcomes obtained in this project can be explained as follows:

This application is useful for individual tourists who wish to get a high level overview of which attractions are worth visiting first and which attractions would provide the most utility in terms of satisfiability. Also, this application can be easily scaled up to include neighboring cities and attractions in those neighboring cities. Given enough and accurate input data for distances between attractions, the popularity of attraction (number of tourist visits each year), the money needed to visit the attraction(entry fee), and time required to tour that particular attraction (in hours), we can easily find the best path for various cities and attractions in those cities. One other usefulness of this application is that tourist agencies can use this application to create an optimal path tour for tourists. Tourist agencies can plan tours and make tour arrangements based on user preferences. If a tourist doesn't give any preference, a default tour can be chosen for the tourist.

D. Stage4 - User Interface.

- The very first interaction of user with this application is by using a website link. We provide the user with a link to access an HTML page in a browser. Here's how the page looks when a user clicks on the link:

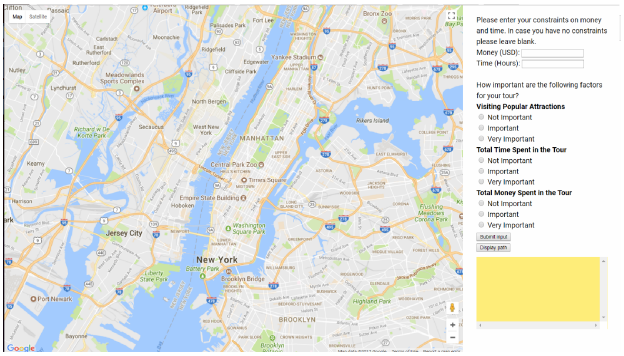

Fig. 7. Initial screen for user interaction with application

- The screenshots below show two different sample navigation user paths.
  First path is generated with no constraints for time and money and all preferences marked important.

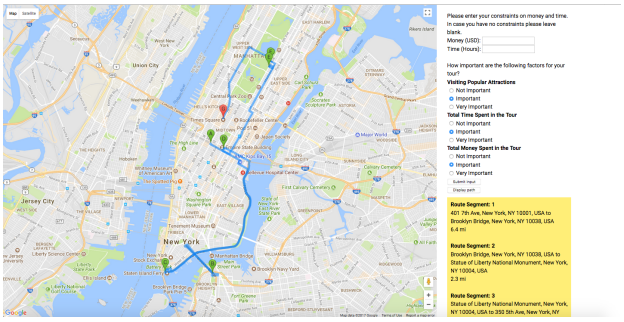

Fig. 8. First sample navigation user path

  Second path is generated with $30 as money constraint and 4 hours as time constrain and all preferences marked important.

- Two error messages may pop up when a user interacts with the user interface. They are as follows:
  - The first error message: Please enter positive value for money
  - The first error message explanation: This error message pops up when a user tries to input a negative value for time or money. Since time or money value should be non-negative, this application is robust to detect negative values and prevents the user from proceeding

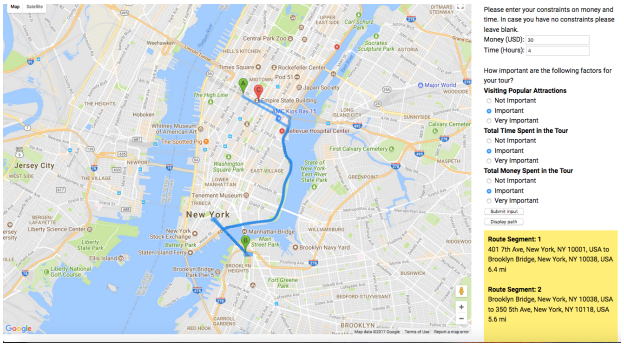

Fig. 9. Second sample navigation user path

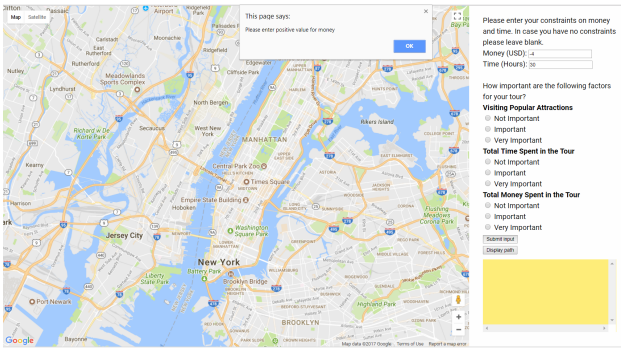

Fig. 10. First error

further if a negative value is entered. Figure 10 shows a screenshot of first error.

- The second error message: Please specify the importance of all the factors
- The second error message explanation: This error message pops up when a user does not give any preferences for popularity, time, and money. As we cannot proceed without knowing user's preferences, this prevents accidental results being displayed and reminds the user to input their preferences. Thus, this application is robust to user input preferences. Figure 11 shows a screenshot of second error

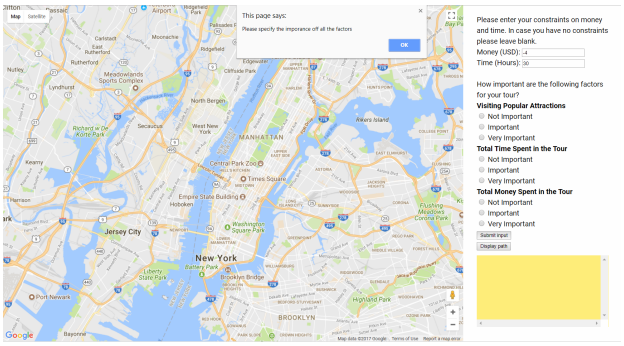

Fig. 11. Second error

- In this project we have created one view that is self-sufficient to take input and display the results on the same page.
  - The interface mechanism: The interface mechanism that generates a different subview, (i.e. the output) is the Display path button. When a user clicks the display path button, the Google Maps API instantly shows the optimum path that the user should take from starting point to end point. The output also shows the order in which the path should be traversed in the form of letters. So if the path shows A, B, C, D, E, then the order is the ascending order of the path shown with E as the second last vertex to be visited before coming back to A.

II. Project Highlights.

- In this project, either decimal numbers or integers are accepted as input.
- Updates to the database are off-line, which means that if a certain tourist attraction needs to be included in the tour, we need to add the distance of that attraction from every other attraction and also calculate all the parameters to find vertex weight for that attraction.
- In the worst case, size of TTP is similar to TSP. In cases where there are more constraints the TTP does not go through all possible paths, thus performing faster than TSP.

References

[1] "PyCharm," https://www.jetbrains.com/pycharm.
[2] "Tourist visits per year to the statue of liberty," https://www.statista.com/statistics/254218/number-of-visitors-to-the-statue-of-liberty-in-the-us/.
[3] "Time required in hours to vist the statue of liberty," https://www.statueoflibertytickets.com/faq/.
[4] "Entry fee in dollars for statue of liberty," https://www.nps.gov/stli/planyourvisit/fees.htm.
[5] "Tourist visits per year to times square," http://www.travelandleisure.com/slideshows/worlds-most-visited-tourist-attractions.
[6] "Time required in hours to vist times square," https://www.tripadvisor.com/Attraction_Review-g60763-d110145-Reviews-Times_Square-New_York_City_New_York.html.
[7] "Entry fee in dollars for times square," .
[8] "Tourist visits per year to central park," https://en.wikipedia.org/wiki/Central_Park.
[9] "Time required in hours to vist central park," https://www.tripadvisor.com/Attraction_Review-g60763-d105127-Reviews-Central_Park-New_York_City_New_York.html.
[10] "Entry fee in dollars for central park," http://www.centralpark.com/things-to-do/central-park-zoo.
[11] "Tourist visits per year to metropolitan museum of art," https://www.metmuseum.org/press/news/2016/annual-attendance.
[12] "Time required in hours to vist metropolitan museum of art," https://www.tripadvisor.com/Attraction_Review-g60763-d105125-Reviews-The_Metropolitan_Museum_of_Art-New_York_City_New_York.html.
[13] "Entry fee in dollars for metropolitan museum of art," https://rsecure.metmuseum.org/admissions.
[14] "Tourist visits per year to empire state building," http://www.nyctrip.com/pages/Index.aspx?PageID=1176.
[15] "Time required in hours to vist empire state building," https://www.tripadvisor.com/Attraction_Review-g60763-d104365-Reviews-Empire_State_Building-New_York_City_New_York.html.
[16] "Entry fee in dollars for empire state building," https://www.citypass.com/new-york/empire-state-building.
[17] "Time required in hours to vist brooklyn bridge," https://www.tripsavvy.com/time-to-walk-the-brooklyn-bridge-442751.