A Real-Time Dynamic Route Control Approach on Google Maps using Integer Programming Methods

Faruk BULUT[1], Mehmet Hamza EROL[2]
[1](corresponding author) Computer Engineering Department, Halic University, Istanbul, Turkey
faruk.bulut@halic.edu.tr, Phone: +905337440512
[2]Izmir Private Ugur College, Karsyaka, Izmir, Turkey
mehmet.hamza.erol3@stu.ugurokullari.k12.tr

The Travelling Salesman Problem (TSP), defined as returning to the starting point after visiting all the points with the least cost, is the modeling framework for many engineering problems. In this study, a real-world application that draws the real time route of the TSP using the current traffic intensity information taken from Google Maps is proposed. In developing the GUI application, different integer programming methods such as Exhaustive Search, Heuristic A-Star Search, BitMask Dynamic Programming, Branch-and-Bound Algorithm, and Greedy Search have been implemented with the help of Google APIs. All these methods, sometimes even Greedy Search, have given the same TSP route for any of test cases. Additionally, a dynamic route update mechanism with Hamiltonian Circuit function is adopted to enhance the conventional TSP system. Sometimes the TSP route list changes according to some sudden reasons or when the traffic intensity changes while travelling the nodes. In this case, the proposed system updates its current route for the rest of the nodes by using the enhanced system to keep the total travel-cost minimum. A user-friendly and dynamic interface, displaying visually the shortest route in distance or duration on Google Maps, has been developed by adding different features such as travelling mode options, remaining route distance and time. This proposed study, which is powered by different algorithms with visual artifacts, might be accepted as a unique blueprint in its field.

Keywords: Travelling Salesman Problem, Hamiltonian Circuit, linear search methods, API, GUI

1. INTRODUCTION

How should be the optimal route if there is an obligatory to return to the home point after visiting N different locations by car, by bicycle, by public transport vehicles or by walking? This question, known as Travelling Salesman Problem (TSP), has an overwhelming fame both in real life and in engineering world for a long time. Optimal route in the explanation of TSP is an important term. The concept of the optimal route, of course, changes according to the needs or the structure of the problem. The optimal route might be firstly the shortest distance, then the least cost, or finally the shortest time. As an instance from real life, a cargo company, which distributes packages, requires an optimal route to deliver them in the shortest time or distance. There are also lots of cases in which TSP solutions should be used in daily life activities such as distributing newspapers, courier visits, and travelling touristic places. Sometimes they need to reach to the destination in a short time, sometimes they try to cut back the travelling expenditures, sometimes they try visit many more nodes as possible as they can in a limited period of time (Voudouris, Christos, and Tsang, 1999)[1]. Integer programming approaches can be applied to TSP problems when small number of visiting nodes exist. In the integer programming filed, the most common solution methods for this problem are Exhaustive Search, Greedy Algorithm, Dynamic Programming and enhanced with heuristic search in order to calculate the least cost of a route (Lancia, Giuseppe, and Serafini, 2018)[2]. TSP problem normally is accepted in the non-deterministic-polynomial (NP-hard) problem category since there is no a suited linear solution. Namely, if a problem cannot be solved in a polynomial-time, it should be considered in NP-hard category. When the number of visiting nodes increase in the route, the complexity of the computational solution increases as well. Similarly, TSP problem can be solved in totally O(N!) time complexity by using recursion programming methods. As it is known, the big-O notation indicates the time complexity of the related algorithm. Recursion methods always give the best result of all possibilities when compared with optimization methods. When the number visiting points increase, calculation time increases exponentially as well. That's why optimization methods are preferred then linear approaches. These linear approaches require bigger computational time and system resources like memory and more CPU speed (Goerigk, Marc, and Schbel, 2016)[3]. On the other hand, there are some other solution styles that can be implemented by optimization algorithms for this problem. Genetic Algorithms (GA), Artificial Bee Colony (ABC), Ant Colony Optimization (ACO), Multi-Layer Perceptrons (MLP), Particle Swarm Optimization (PSO), and Fruit Fly optimizers are common methods which have already been applied to this field. But it is a compulsory to emphasize a reality that those optimization methods in the literature do not always guarantee to find the best TSP solution, they just find out one of the possible solutions. If any possible solution is acceptable for the system, then there is no need for extra calculations. On the other hand, if the best solution is needed, then one of the integer programming methods should be preferred (OLAK, 2010; Rao and Hedge, 2015)[4,5]. Up to now, in this filed, lots of theoretical studies and practical application have been performed. The first optimal solution has been advised in 1930s by using scientific methods. Lately, Rao and Hegde has made a literature survey on TSP using genetic algorithms. Also they have presented an approach to solve TSP by using Sequential Constructive Crossover operator in order to improve the quality of solution space (Rao and Hegde, 2015)[6]. Hoffman et al. has presented a theoretical and practical study called as The traveling salesman problem (Hoffman, L., Padberg, and Rinaldi, 2013)[7]. Choong et al. has introduced an approach for the TSP using ABC optimization method with a modified choice function (MCF) using the Hyper-heuristic Flexible Framework (HyFlex). MCF, as a heuristic search method, is implemented to regulate the neighborhood search heuristics which are adopted by the onlooker and employed bees naturally (Choong, Wong, and Lim, 2017)[8]. Before this study, in 2008, Wong et al. has presented a blueprint approach to the TSP problem using ABC optimization method. The ABC optimizer model is algorithmically implemented using the collective intelligence presented in bee foraging behaviors. This optimizer employs a natural metaphor to create an optimization model. Bees in the selected colony are able to build continuously feasible tours as described in waggle dances model. This proposed model gives successful experimental results when compared with the other existing approaches on benchmark problem sets (Wong, Li-Pei, Low, and Chong, 2008)[9]. A new study recently has presented in the same field using another optimization approach, a Genetic Ant Colony Optimization by Changdar et al. A hybrid method, combining GA and ACO together, is developed to solve a solid multiple Travelling Salesmen Problem (mTSP) in fuzzy rough environment. In this mTSP model, there are more than one traveler using conveyance facilities to visit one city after one. Each of the visitors selects their route using ACO and the constructed route is controlled by GA simultaneously (Changdar, Chiranjit, Pal, and Mahapatra, 2017)[10]. Onder et al. has also presented a traveling repairman problem (kTRP) solution approach using integer programming formula for the purpose of solving TSP problems. kTRP is known as the minimum latency deliveryman problem (Onder, Gozde, Kara, and Derya, 2017)[11]. This proposed paper mainly focuses on dynamic route control. In other words, the aim is to find out the optimal TSP route with different kinds of algorithmic methods and to adopt it to the real life by a desktop application. This study is a progressive form of the previously proposed proceedings by Erol and Bulut (Erol and Bulut, 2017)[12]. New features, dynamic route control units, and optimized heuristic models are added to improve the current study. The Google Maps API (Application Programming Interface) application is implemented in Java platform in order to enable to trace the TSP route. The GUI (Graphical User Interface) application calculates the least distance or least timed route for N points by using instant traffic information taken from Google APIs.

Instant traffic information in JSON (JavaScript Object Notation) formats is given freely by Google in a limited quota. This paper totally has five parts. The next second part describes the selected linear TSP methods and algorithmic comparisons of the described techniques. The third part explains how to update the route with Hamiltonian Circle. The fourth part is about experimental outputs. The last section finally concerns about conclusion.

2. LINEAR TSP METHODS

TSP is about finding least cost cycle for visiting N points in two-dimensional plane. Increase in the number of visiting points causes exponentially growth in the computational complexity. Thats why TSP in much more visiting points is considered in NP hard difficulty level. With a reasonable number of points for TSP, the optimized recursive methods can give successful results. In this study, a heuristic approach is proposed to the recursive method in order to decrease both time and space complexities. Whereas lower time complexity decreases computational time, lower space complexity decreases memory requirements. In searching the possible solutions, recursion function is optimized by using some methods such as Branch & Bound algorithm, A-star heuristic methods, and Heap Data structures in order to find the optimal solution. The theoretical explanations of the selected conversional methods are briefly as follows.

2.1 Exhaustive Search

In computer science, Exhaustive search is also known as brute-force search and direct search method. This algorithm sequentially evaluates every possible cases in order to find the best one. Exhaustive search basically does not contain any theoretical or algorithmic approach to reach the goal state in a less computational complexity. Therefore, it is accepted as the slowest, the most natural, and the most canonical method since there is no logical elimination on the cases. In computational complexities, Brute-force method is generally categorized in the NP-Hard type problems. However, it should be emphasized that most of the NP-problems can only be solved by Brute-force search methods (Kullmann, 2017)[13]. In this project, Exhaustive Search for TSP is considered as a permutation problem. All the N points in the list are put onto a line to arrange. The starting and final point is placed both to the head and tail of the line. All the possible alignments of N-1 points are considered in the interval of head and tail points. The question finally transforms into simple permutation problem for N-1 points. Then, all the possible arrangements of N-1 points are calculated one by one to find out the shortest distance. As an example, suppose there are 6 cities named as A, B, C, D, E, and F. Let A is the starting and ending point for TSP. Then, the Exhaustive approach can be illustrated in the Table 1.

Table I: Permutation List of TSP for 6 nodes

Case#	**Arrangement style**	**Cost**
1	A - B - C - D - E - F - A	$Cost_1$
2	A - C - B - D - E - F - A	$Cost_2$
...	...	...
120	A - F - E - D - C - B - A	$Cost_{120}$

In the interval, the points of B, C, D, E, F will be placed onto the line in 5! possible arrangements. The cost of shortest distance of possible 120 (5! = 4.3.2.1 = 120) cases will be evaluated in order to find out the least cost. The time complexity of this search method in big-O notations is obviously O(N!). When the magnitude of N increases, the complexity asymptotically increases. Thus, this permutational approach is regarded as in NP-Hard complexity.

2.2 Greedy Search

Greedy search, as its name can give a brief idea for the concept, always chooses the best option at each state without turning back to previous states. The fundamental mechanism is based on to choose the local optimal solution in order to reach to the global optimal solution. The best option can be either minimum value or maximum value according to the problem type. To find out the best global optimum by using this greedy search style is not always certain. Even though it works fast with a O(N) complexity, it cannot always guarantee to give the ultimate solution. On the other hand, in some engineering problems, Greedy method might quickly give the required optimal solution (Chen, Yu, and Chang, 2014)[14]. There are three main variations of Greedy search styles. These are Pure, Orthogonal, and Relaxed. Pure one in this study is implemented in order to prioritize options within a search.

2.3 A-Star (A* Heuristic) Search

A-Star (A*) is one of the heuristic search algorithms. Its basic mechanism has two main functions as seen in the Formula 1 to ensure the overall optimality.

$$f(x) = h(x) + g(x)$$

The total $f(x)$ evaluation in A* method is obtained with the summation of both $g(x)$ and $h(x)$ functions. $g(x)$ function gives the exact total cost until reaching to the current state. Heuristic $h(x)$ function gives estimated total cost from the current state to reach to the last state. $h(x)$ function has a customizable assumption in evaluation the possible cost. The total estimated cost, calculated by the total $f(x)$ function, has to be smaller or equal to the real cost. If the estimated cost is bigger, this solution will be dropped. This type of an approach fastens the overall performance (Aygn and Akay, 2015)[15]. In the implementation of the proposed study, alternative solutions, which are obtained after each states, are stored in a vector such as X = $\{x_1, x_2, x_3, .., x_N\}$. The array X are put into a Heap data structure (Priority Queue) (Kaplan, Haim, and Tarjan, 1999)[16] in order to both minimize the execution time and decreases the requirements of system resourses. The optimal $f(x_i)$ solution is determined after all possible cases examined recursively. Heap data structures is used to store the alternative solutions throughout the executions. Although there are some other sorts of heaps, min heap provides a swift operation on swapping the outputs. Heap basically uses a linear vector container for the solutions. Pushing (inserting) a solution to heap is performed in big-O complexity of $O(logN)$. Similarly deleting or popping a solution from heap results in $O(logN)$ (Bulut and Amasyali,)[17]. Normally a basic operation can induce a $O(N)$ complexity when a simple container such as a vector is selected. Since this kind of a data structure fastens the overall computational time, it becomes preferable in the A-Star method implementation.

2.4 Branch & Bound (B&B) Method

Branch and Bound is a kind of optimized version of recursive methods. It is widely used in non-linear problems. B&B has a similar structure like A-Star algorithm. In both algorithms there are $h(x)$ and $g(x)$ functions for the same purposes. Briefly "Branch and Bound" explains the basic philosophy, branch to all the possible solutions and control the limits (bound) if they fit to the limitations. In B&B, there is a recursive function which checks all the possibilities. In each iteration of this function, the h and g functions are summed. Branching from the current state to deeper states recursively is performed if a satisfied summation is taken when compared with the lower bound or upper bound value. This lower and upper bound values in the execution dynamically varies in each iterations of the recursive function. As the TSP is a kind of minimization problem, the lower bound limitation system is run in implementations (MUTLU, 2010)[18].

2.5 BitMask Dynamic Programming

Each data is coded with binary digits, 0 and 1, in computer systems. The permutations of 1 and 0 generates different unique data. These permutations are called BitMask. As an instance, Letter A in ASCII code is 01000001. This binary sequence represents a number and its decimal equivalence is 65. This basic alignment philosophy of BitMask can be similarly adopted to the

TSP problem. Let N points are to be travelled. 1 is given to a point which has already visited and 0 is given to a point which is not visited yet. If the visited points and unvisited points are grouped in a certain order, a binary number in N digits is generated. As it is known, recursive functions make a similar calculation at inner steps again and again. In dynamic programming, there is no need to re-recalculate the previous cases. BitMask is used with dynamic programming in order to reduce the computational cost by omitting some calculated cases. The time complexity of TSP application using traditional integer programming methods can be decreased to $O(N^2 * 2^N)$ using BitMask Dynamic Programming (Urrutia, Milans, and Lkketangen, 2015)[19].

2.6 Algorithmic comparisons of the methods

According to the explanations of the methods, they have some different time complexities. In Table 2, there are time complexities in big-O notations and whether optimal solution is guaranteed by the method or not. Obviously BitMask Dynamic Programming style outperforms than the others in time complexities.

Table II: Comparison of TSP Algorithms

TSP Algorithms	**Time Complexity**	**Data structure**	**Optimal Solution**
Exhaustive Search	Exactly $O(N!)$	Not needed	Guaranteed
Greedy Search	$O(N)$	One dimensional Array	Not Guaranteed
A-Star (A* Heuristic) Search	Strongly better than $O(N!)$	Min Heap	Guaranteed
Branch & Bound Method	Better than $O(N!)$	Not needed	Guaranteed
BitMask Dynamic Programming	$O(N^2 * 2^N)$	Two dimensional Array	Guaranteed

3. UPDATING ROUTE WITH HAMILTONIAN CIRCUIT

While visiting the nodes in a given list, updating the current route according to some reasons is the most important part of the introduced study. During the travel of nodes according to a certain TSP route, there might be some unexpected changes according to some reasons listed below:

- a way (road) might be closed due to a traffic accident
- a heavy traffic jam might occur in the current way
- a way might be closed and Google Maps are not aware of the current status
- one of the visiting points should be dropped from the list
- a new visiting point might be instantly added to the route.

The TSP route should be updated due to these sudden cases. Hence, the starting and target points will not be the same so that a new problem type will appear. The starting point will be the current point where we stay and the target point will be the starting point of the TSP route at the initial state. Therefore, this scenario is not a shortest part, nor a minimum spanning tree. It is a specific problem that an appropriate route to reach the target point after visiting all the remaining points with the least cost. Even though there are some proposed methods in the literature to solve this kind of a problem, the Hamiltonian Tour (Circuit) method has been preferred since it satisfies the corresponding requirements. The illustration of a scenario for the presented case is given in the Figure 1. Assuming the A node is the current state and the E node is the destination node, (a) represents the initial undirected and weighted graph of the given scenario and (b) represents the calculated Hamiltonian cycle starting from A and ending in E with the least coast.

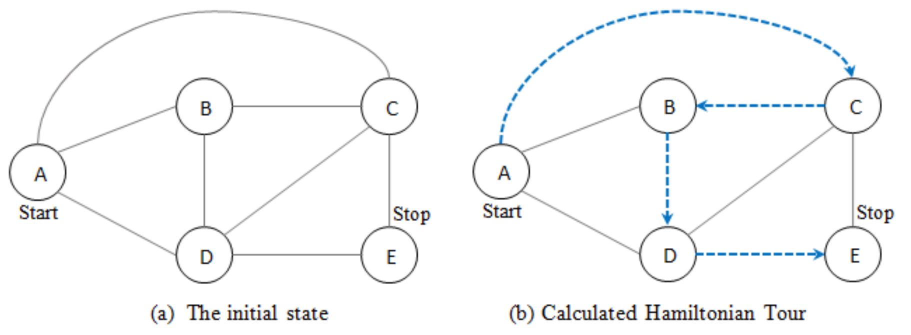


Figure 1.An illustration of Hamiltonian Circuit

In an undirected or directed graph G, Hamiltonian Circuit is a directed path that visits each node in the list exactly once to reach the goal state from the start state. A and E are the starting and stopping vertexes respectively. If the starting point and the target point is the same it is clearly the Hamiltonian Cycle. In contrast, Hamilton path has only one direction. Each edge in the graph has an individual weight different from others. Thus, the dotted blue path, which connects adjacent nodes, probably has the minimum total distance cost of all other alternate routes. Maybe the other Hamiltonian routes is longer (Dorigo, Marco, and Gambardella, 2016)[20]. Actually, Hamilton Tour is a NP-Complete problem since it has a nonlinear time complexity. This means that when the number of vertex increases, the computational complexity increases asymptotically as well. Most of the linear approaches to solve this problem type in the literature has big-O complexities. BitMask Dynamic approach is chosen in order to complete the Hamiltonian tour since it has the least time complexity of all the other linear methods. Adding an extra Hamiltonian Circuit function to the TSP route transforms this study into a dynamic model that can be easily used in real life applications

4. EXPERIMENTAL STUDIES

Java programming language has been preferred due to its compatibility to all of the APIs services provided by Google. Since all the search methods mentioned in this paper are implemented by the authors, the codes have a flexible structure and suite the needs. Additionally, the source codes have been compiled in both Linux and Windows operating systems. The implementation codes written in Java and the essential documents related with this study can be found the web site for examinations further studies: `https://sites.google.com/site/bulutfaruk/study-of-route-control-on-google-maps`. The implementation process, the testing stages, and evaluation steps are as follows.

4.1 Project Deployment Procedures

These essential steps as listed below are the implementation process for the setup.

Step 1. JDK (Java Development Kit) v8.12x 64Bit software development kit and Eclipse Neon v3 developing editor (IDE, Integrated Development Environment) should have been installed first

Step 2. Swing can be preferred due to its common and rich features for GUI. The Window-Builder tool has been installed to Eclipse for visual GUI applications (`https://eclipse.org/windowbuilder`)

Step 3. The Gradle v4.x tool is added to Eclipse in order to be able to run new projects in the same project (`https://gradle.org/gradle-download`).

Step 4. The google-maps-services-0.1.17.jar file has been downloaded from the web site, https://github.com/googlemaps/google-maps-services-java to add to the project via Gradle.

Step 5. A free Google API Key has been taken from the web site, `https://developers.google.com/console` with a Google account in order to access to map services.API is described as a set of functions and procedures which allows to create new applications and services. API actually is an online communication tool between applications, operating systems, and services.

Step 6. Definitions of the activated API services for the project are as follows:

- Directions API: It enables the users to find their ways, return multi-part directions for a series of waypoints, and directions for several transportation modes.
- Distance Matrix API: Both distance and duration information for multiple destinations and transport modes can be taken with this API. The Google Maps Distance Matrix API gives information based on the suggested route between start and stop nodes, as computed by the Google Maps API, and consists of rows containing distance and duration values for each pair.
- Elevation API: The Google Maps Elevation API supplies elevation data for all locations in the world, including even depth locations on the ocean from the ground (which return negative values).
- Geocoding API: The Google Maps Geocoding API is another service providing geocoding and reverse geocoding for addresses. Geocoding is simply the process of transforming addresses (e.g., a street address name) into geographic coordinates (e.g., latitude and longitude numbers), which are used to place markers on the map.
- Places API: Local data all around the world taken from the huge Google database where there are millions of location information. Places features more than 100 million points of interest that are powered by owner-verified listings and user-moderated contributions.
- Roads API: The Google Maps Roads API labels the roads a vehicle is visiting along and provides extra metadata about those roads, such as speed limits and traffic signs.
- Time Zone API: The Google Maps Roads API allows to map GPS (Global Positioning System) coordinates to the geometry of the road, and to figure out the speed limits through the road segments. The API can be available via a simple HTTPS (Secure Hypertext Transfer Protocol) interface, and exposes some following services such as snap to roads, nearest roads, and speed limits. The Google Maps Time Zone API provides also time offset data for locations on the earth from the UTC (Universal Time Coordinated).

Step 7. Instant messages are received from Google as the JSON packages rather than XML. JSON is a JavaScript based data transfer model in text format between a browser and a server. Data in JSON objects can be easily deserialized with no complicated parsing and translations.

Step 8. The proposed Java GUI application has calculated the least time or the shortest distance of the route by using the JSON messages. In other words, the most appropriate route for both distance and time can be received from Google Maps in JSON packages. These calculations are performed using the integer programming methods such as B&B, A-Star Search, and so on.

Step 9. JxMaps APIs JAR (Java ARchive) files have been added to the project in order to activate the Google maps in the GUI workplace and to make interactive operations on the map (`https://www.teamdev.com/jxmaps`). JxMaps, as a registered software product written in Java, is a complete tool for Google Maps API. There is no need to learn or use JavaScript to work on Google maps. Marking and adding locations on the map in the application is done by the JxMaps

API library. Furthermore, TSP route with markers of the visiting points can be visualized in different colors on the map is performed by the JxMap service.

4.2 Experiments

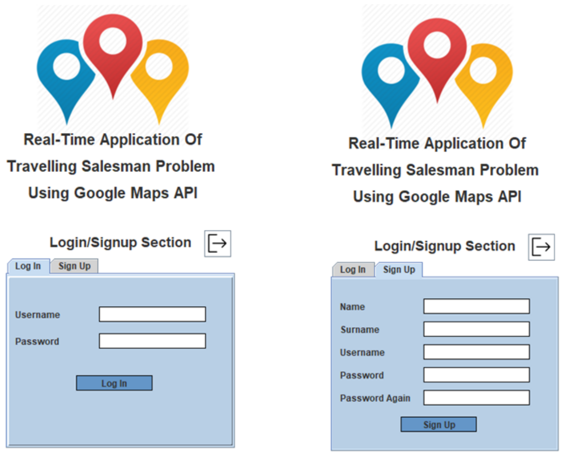


Figure 2. Start window of the GUI application (Logging in and registration)

The startup window of the GUI has an optional registration section. The registered users can store the visiting points and recall them in next runs.

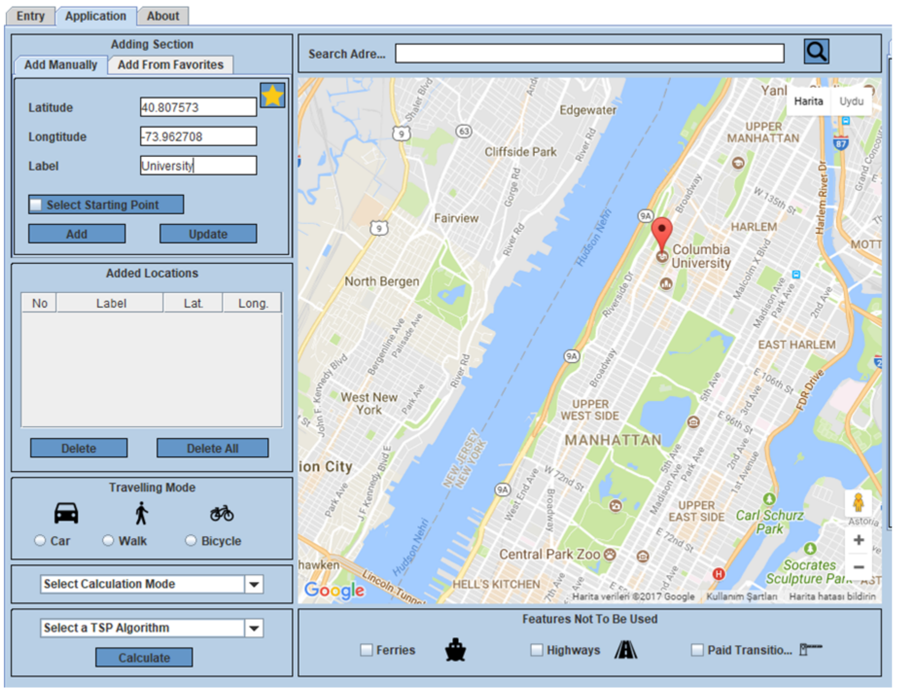
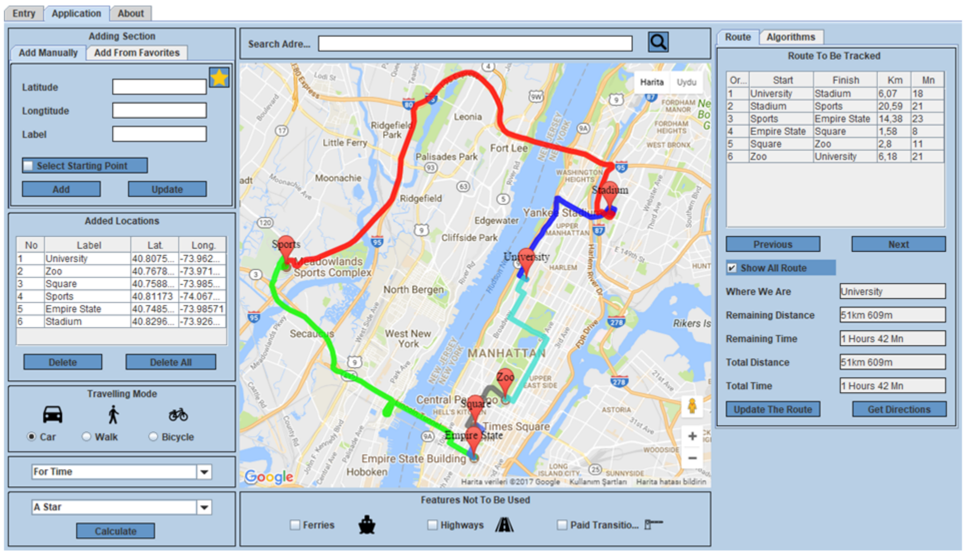


Figure 3. Cropped GUI screen shot

In Figure 3, there is a cropped screen shot of the GUI application. Manual configurations for a user starts from the upper left side to downstairs step by step. At the upper left side, there is an Add from Favorites section, where there are the places which have been visited before by the authenticated user. Favorite places can be added easily to the Added Locations window at the middle. At the Add Manually tab, any location can be easily added to the list by either clicking the destination point on the live Google map at the right side or manually entering the latitude and longitude coordinates into the related boxes with a label name. When clicking a location on the map, its original name, which is given by Google automatically, is pasted in to the Label text box. Latitude and longitude numbers for a specific location can be entered manually. TSP starting point can be assigned by clicking the checkbox of the Select Starting Point. At the Added Locations section, any point can be deleted or updated. Furthermore, there is valuable contribution in the GUI: a text box at the top, called as Search Address. Any address can be written there and the address can be automatically found by the program. Moreover, the orange star sign adds the current location into the favorites section for the user. After creating the Added Locations list, any travelling mode of car, walk, bicycle can be selected by clicking the radio button. Select Calculation Mode offers two options: the least TSP route distance or the shortest TSP route time. TSP route can be obtained by using different algorithms Select a TSP Algorithm serves 5 different algorithmic methods such as Brute-force, Greedy, A-Star (A* Heuristic), Branch & Bound, and BitMask Dynamic Programming. Lastly, in the Feature Not To Be Used section, any transportation style such as ferries, highways, and paid transitions can be removed Calculate button outputs both the TSP route on the Google map and travelling statistics at the right side as shown in the Figure 4.



Added Locations

No	Label	Lat.	Long.
1	University	40.8075...	-73.962...
2	Zoo	40.7678...	-73.971...
3	Square	40.7588...	-73.985...
4	Sports	40.81173	-74.067...
5	Empire State	40.7485...	-73.98571
6	Stadium	40.8296...	-73.926...

Travelling Mode: [✓ Car] Walk Bicycle

For Time

A Star

Route To Be Tracked

Or...	Start	Finish	Km	Mn
1	University	Stadium	6,07	18
2	Stadium	Sports	20,59	21
3	Sports	Empire State	14,38	23
4	Empire State	Square	1,58	8
5	Square	Zoo	2,8	11
6	Zoo	University	6,18	21

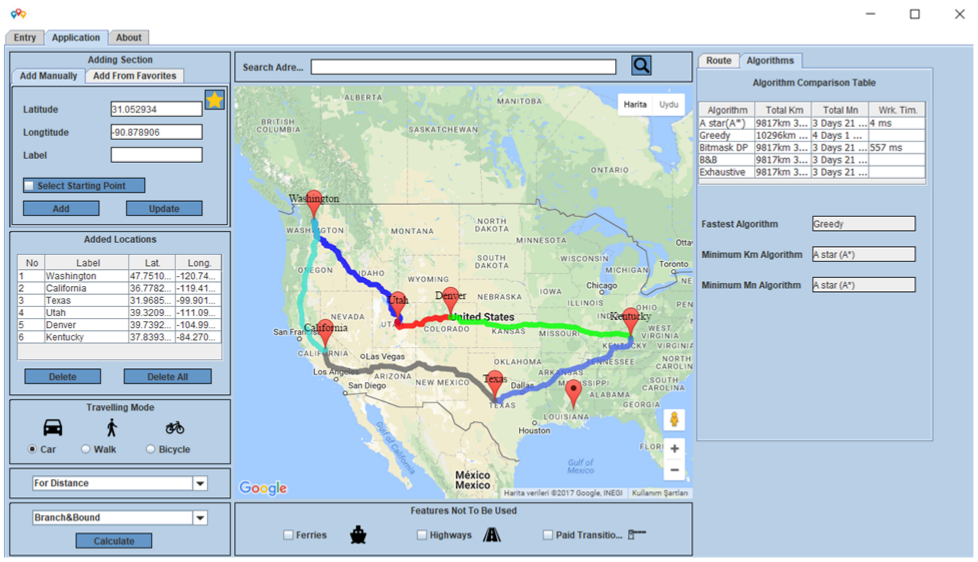
[✓ Show All Route]

Where We Are	University
Remaining Distance	51km 609m
Remaining Time	1 Hours 42 Mn
Total Distance	51km 609m
Total Time	1 Hours 42 Mn

Figure 4. Outputs in the GUI screen shot

The TSP route in different colors for the given scenario is drawn in the Figure 4. The labels of the visiting points are written on the markers. The Route To Be Tracked list gives the travelling list in a sequence. In each line of this list, there are starting and finishing locations, distance in km, and the duration in minutes. Previous and Next buttons help users to tack each of the steps in the map. At the lower right corner, there are some valuable statistics about the TSP results such as the current position, remaining distance, remaining time, total distance and total time. Update The Route button regenerates the TSP route while travelling the nodes.

This option as mentioned previously renews the current route which has been calculated before. While travelling, some sudden obstructions might appear on the route such as an abrupt traffic accident or a municipal excavation work. The road also can be blocked. The traffic intensity might change. A visiting point in the current list may be removed. These factors naturally affect the traffic intensity. In these similar cases, the route ought to be updated by receiving the instant traffic information from Google. According to these reasons, the circuit can be updated by the Update The Route button. As mentioned in the previous chapter, TSP problem transforms into a Hamiltonian Tour.



Latitude: 31.052934
Longtitude: -90.878906
Label:
Select Starting Point [ ]

Added Locations

No	Label	Lat.	Long.
1	Washington	47.7510...	-120.74...
2	California	36.7782...	-119.41...
3	Texas	31.9685...	-99.901...
4	Utah	39.3209...	-111.09...
5	Denver	39.7392...	-104.99...
6	Kentucky	37.8393...	-84.270...

Travelling Mode: [✓ Car] Walk Bicycle

For Distance

Branch&Bound

Features Not To Be Used: [ ] Ferries [ ] Highways [ ] Paid Transitio...

Algorithm Comparison Table

Algorithm	Total Km	Total Mn	Wrk. Tim.
A star(A*)	9817km 3...	3 Days 21 ...	4 ms
Greedy	10296km ...	4 Days 1 ...	
Bitmask DP	9817km 3...	3 Days 21 ...	557 ms
B&B	9817km 3...	3 Days 21 ...	
Exhaustive	9817km 3...	3 Days 21 ...	

Fastest Algorithm: Greedy
Minimum Km Algorithm: A star (A*)
Minimum Mn Algorithm: A star (A*)

Figure 5. A general view of the application window

Get Directions returns detailed route directions for users. In this direction, there are detailed series of waypoints. Optionally this direction can be printed out. Different TSP routes can be observed in case of different choices such as travelling algorithm, travelling mode, travelling features etc. This proposed application additionally can give some possibilities for whom they want to visit by walking or cycling. At the upper right section in Figure 5, there is an algorithm comparison table where there are outputs of each of the methods. Moreover, there are some other statistics such as the fastest algorithm of all in terms of executional time, the algorithm which gives the minimum distance, and finally the method which outputs the shortest time.

4.3 Evaluation of the integer programming methods

The implemented GUI offers a user friendly environment with some extra options to choose a travelling mode by a travelling vehicle. All the experimental tests indicate that there is no need to use strictly an optimization method for the proposed GUI. There are naturally some constraints in the process of the recommended method. The first barrier is the limitation of query numbers to the Google Maps APIs. Maximum 2500 queries a day can be sent and answer of maximum 50 queries can be demanded at a time. Exceeding the limitations results a charge. Due the limitation of max. 50 instant queries at a time, maximum 7 visiting points can be tested lively. A matrix by 77, which contains results of 7 travelling points to each other, is simply under the 50 query limits. The query limitation for a JSON package at a time is 50 at most. There can be some further progressive studies for this introduced blueprint study. Firstly, a mobile application can be implemented in order to adopt into Android and IOS operating systems. Secondly, this project can be moved into a web page by using Java Script so that it can be easily run without requiring a JRE (Java Runtime Environment). This adoption will make the project operating system independent, just needing a simple internet browser. Thirdly, this application can be run with instant GPS signals so that it will get easier to trace the route lively on the GUI application. Therefore, pursuit of the circuit becomes easier. Besides, optimization algorithms such as ACO, ABC, GA, and PSO can be added to be able to run on big number of visiting points. Lastly, the number of instant queries to the Google Maps API service can be increased if purchasing. As a prototype, this study can be improved and enhanced by adding these extra features. There is an experimental output results in Table 3 regarding the scenario of the Figure 4. Total distance and total duration of the trip are the computed outputs of the algorithms after receiving the JSON packages from Google. The computational times of the methods are calculated using an Intel Celeron N3060 CPU having 4GB RAM. All the methods but Greedy have given the same results in different computational times. As mentioned before, since Greedy uses a linear search, it gives the fastest results. However, Greedy fails giving the shortest route. As the computational time of the Bitmask Dynamic Programming method is the smallest one, it has a comparable superiority.

Table III: Experimental outputs of the given scenario in Figure 4.

TSP Algorithm	**Total Distance**	**Total Duration of Trip**	**Computational Time**
A* Search	52.197 km	1 hour 42 minutes	0.21 ms.
Greedy Search	64.760 km	2 hours 9 minutes	0.00 ms.
Bitmask DP	52.197 km	1 hour 42 minutes	0.05 ms.
B&B	52.197 km	1 hour 42 minutes	0.30 ms.
Exhaustive	52.197 km	1 hour 42 minutes	99.2 ms.

Table 4 represents a Hamiltonian Tour with the least coast. In the scenario given in Figure 4, there are 6 nodes to be visited. While visiting the third node of the sport center after the university and stadium, the rout to the destination to the zoo node should be reorganized. The output for the Hamiltonian Tour can be seen in Table 4. Similarly, BitMask DP has given the same shortest travelling distance as the others (apart from Greedy) with the least computational time. Greedy has failed again although it gives a solution in the fastest time.

Table IV: Experimental outputs of the Hamiltonian Tour for the given scenario in Table 3.

Hamiltonian Tour Algorithm	**Total Distance**	**Total Duration of Trip**	**Computational Time**
A* Search	24.94 km	1 hour 3 minutes	0.11 ms.
Greedy Search	29.70 km	1 hour 42 minutes	0.00 ms.
Bitmask DP	24.94 km	1 hour 3 minutes	0.01 ms.
B&B	24.94 km	1 hour 3 minutes	0.20 ms.
Exhaustive	24.94 km	1 hour 3 minutes	44.1 ms.

5. CONCLUSIONS

Some integer programming methods are used to calculate time and distance of a TSP route and additionally Hamiltonian Circuit function are added to the TSP route in this paper. An original GUI design for TSP with some beneficial features has been implemented in this study. A real-world application that shapes a travelling route, using the current traffic intensity information taken from Google, is proposed. This presented study is developed by integer programming techniques. Exhaustive Search, Heuristic A-Star Optimizer, BitMask Dynamic Programming, Branch-and-Bound Algorithm, and Greedy have been implemented and given successful results. Furthermore, a customizable and dynamic route system is added to the classical TSP system to maintain the total travel-cost minimum. A comprehensible interface, showing the shortest route in both distance and time, has been implemented by adding travelling mode options, remaining route distance, and time. This introduced draft study firstly gives a dynamic route control for a user who want to travel some points whereas Google APIs do not supply this kind of a service. Secondly the written program codes are completely original. Thirdly a real-time TSP route is drawn with instantaneous information by API deserializations. Furthermore, the TSP route can be renewed anytime needed. Received directions for the route gives a great benefit to users. As a conclusion, this proposed paper presents nor a complex GUI application neither a tutorial or manual for the GUI. Actually, some features such as integer programming techniques, customized methods, dynamic circuit control units are aggregated in this study to illustrate the proposal. Since this paper proposes a real-time dynamic route control approach by using integer programming methods by realizing of the theoretical approaches in the literature, it should not be considered as a manual of a proposed GUI application. In fact, combining TSP with an application which brings great convenience to real-life makes this study very meaningful.

References

Aygn, S. and Akay, M. 2015. Matlab paralel hesaplama arac ile a* algoritmasnn rota planlama in analizi. *Gen Mhendisler Sempozyumu, Trk Mhendisler Birlii, Mays, Ankara*.

Bulut, F. and Amasyali, M. F. Locally adaptive k parameter selection for nearest neighbor classifier. *one nearest cluster 20(2)*, 415–425. Pattern Analysis and Applications.

Changdar, Chiranjit, Pal, R. K., and Mahapatra, G. S. 2017. A genetic ant colony optimization based algorithm for solid multiple travelling salesmen problem in fuzzy rough environment. *International Soft Computing 21.16*, 4661–4675.

Chen, M. H., Yu, C. W., and Chang, P. C. 2014. Block based imperial competitive algorithm with greedy search for traveling salesman problem. *International Scholarly and Scientific Research Innovation 8*, 793–799.

Choong, S. S., Wong, L.-P., and Lim, C. P. 2017. An artificial bee colony algorithm with a modified choice function for the traveling salesman problem. In *International IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. DOI: 10.1109/SMC.2017.8122629.

Dorigo, Marco, and Gambardella, L. M. 2016. Ant-q: A reinforcement learning approach to the traveling salesman problem. *Proceedings of ML-95, Twelfth Intern. Conf. on Machine Learning*.

Erol, M. H. and Bulut, F. 2017. Real-time application of travelling salesman problem using google maps api. *In Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT)*, 1–5. IEEE.

Goerigk, Marc, and Schbel, A. 2016. Algorithm engineering in robust optimization. Springer International Publishing, 245–279.

Hoffman, L., K., Padberg, M., and Rinaldi, G. 2013. Traveling sales-man problem. In *International Encyclopedia of operations research and management science*. Springer US, 1573–1578.

Kaplan, Haim, and Tarjan, R. E. 1999. New heap data structures. *Department of Computer Science, Princeton University*. Technical Report TR-597-99.

Kullmann, O. 2017. The science of brute force. *Communications of the ACM*.

Lancia, Giuseppe, and Serafini, P. 2018. Traveling salesman problems. compact extended linear programming models. *Springer, Cham*, 155–164.

MUTLU, M. 2010. Kaynak dengeleme problemi iin bir dal ve snr algoritmas (a branch and bound algorithm for resource leveling problem). *Doktora Tezi, Orta Dou Teknik niversitesi Fen Bilimleri Enstits, Ankara*.

OLAK, S. 2010. Genetik algoritmalar yardm ile gezgin satc probleminin zm zerine bir uygulama. *ukurova niversitesi Sosyal Bilimler Enstits Dergisi 19,* 3.

Onder, Gozde, Kara, I., and Derya, T. 2017. New integer programming formulation for multiple traveling repairmen problem. *Transportation Research Procedia 22,* 355-361.

Rao, A. and Hedge, S. K. 2015. Literature survey on travelling salesman problem using genetic algorithms. *International Journal of Advanced Research in Education Technology (IJARET) 2*, 42–45.

Rao, A. and Hegde, S. K. 2015. Literature survey on travelling salesman problem using genetic algorithms. *International Journal of Advanced Research in Education Technology (IJARET) 2.1*, 42.

Urrutia, S., Milans, A., and Lkketangen, A. 2015. A dynamic programming based local search approach for the double traveling salesman problem with multiple stacks. *International Transactions in Operational Research 22,* 1, 61–75.

Voudouris, Christos, and Tsang, E. 1999. Guided local search and its application to the traveling salesman problem. *European journal of operational research 113.2*, 469–499.

Wong, Li-Pei, Low, M. Y. H., and Chong, C. S. 2008. A bee colony optimization algorithm for traveling salesman problem. In *International Modeling and Simulation.* Second Asia International Conference on. IEEE 2008. AICMS 08.

Faruk BULUT was born in Turkey in 1974. He got his bachelors degree in the Computer Education Department at Marmara University in 1998, master degree in the Computer Engineering Department at Istanbul in 2010, and PhD degree in the Computer Engineering Department at Yildiz Technical University in 2015. He has been a lecturer in the Halic University. His major areas of interests are: Image Processing, Machine Learning, Meta Learning and Ensemble Methods.

Mehmet Hamza EROL was born in Izmir, Turkey in 2001. He is now a student in The Private Ugur High School in Izmir. He won a bronze medal in the international competition of IOI (International Olympiad in Informatics) 2018 in Japan and won a silver medal in the National Turkish Olympiad in Informatics in 2017.