Using Smart A* Algorithm to Solve TSP Navigation Problem

Hatem F. Halaoui
Computer Science
Haigazian University
Beirut, Lebanon
Email: hhalaoui@haigazian.edu.lb

Abstract—**Navigation queries are very common among travelers. Moreover, traveling to multiple destinations in one trip is very common as well. The Traveling Salesman Problem (TSP) is one of the most famous multi-destination path problems. Solving TSP efficiently with real-time factors (traffic, distance, real-time delays) is very useful for multiple navigation queries. Google maps, Yahoo maps, and many others are examples of such online navigation applications. Calculating the best driving path between multiple addresses is subject to many factors including distance, road situation, road traffic, speed limitations and others. This paper presents the use of smart heuristic functions, intelligent algorithm A*, traditional graph algorithms like Hamilton circuit, as well as efficient data structures in finding an efficient cycle path between multiple addresses.**

Keywords— Traveling Salesman Problem; Intelligent Navigation Algorithms; Smart Navigation; Hamilton circuit; A* Algorithm.

I. INTRODUCTION

Traveling between places (destinations) is a common task for many people like tourists, sales people, and others. Most of these would like to visit multiple destinations in one trip. This paper proposes a solution for such queries.

This section introduces the main topics behind the proposed approach: the Travelling Salesman Problem (TSP), Spatial Databases (presented as the main data warehouse of our approach), Geographical Information Systems (GIS), and heuristics. We also present an overview of the adopted approach.

A. Traveling Salesman Problem (TSP)

The Travelling Salesman Problem (TSP) [11], which was defined in the 1800s by the Irish mathematician W. R. Hamilton and by the British mathematician T. Kirkman, describes a salesman who needs to travel between a certain numbers of cities. The order in which the cities are to be visited is not important, as long as they are all visited in one trip which ends back at the start city. Cities are connected to each other by roads, railways, airplane paths, or any other means of transportation. Each one of the links between the cities has one or more weights that could represent distance, time, or cost. The main problem is to find the shortest path starting at a source, traveling to all needed destinations, and ending at the source. The TSP is typical of a large class of "hard" optimization problems that have intrigued mathematicians and computer scientists for years.

An optimal solution for the TSP with high number of vertices using traditional algorithms is very time consuming and does not match with real-time problems. Traditional algorithms work well when the number of vertices in low, below 10, so they are better used after decreasing the number of map (graph) vertices. For this reason, our approach will adopt a solution that uses heuristics to decrease the number of graph vertices.

B. Spatial Databases

Spatial databases are the main data warehouses used by Geographical Information Systems. Spatial databases are databases used to store information about geography such as geometry, positions, coordinates, and others [4] [9]. Also, they might include operations to be applied on such data.

C. Geographical Information Systems and Driving Path Applications

A Geographic Information System (GIS) is a collection of computer hardware and software for capturing, managing, analyzing, and displaying all forms of geographical information [6] [7]. Finding the Directions (driving/walking) path is one of the most asked queries in GIS applications. The most important factors that influence the response to such queries include: distance, road situation, road traffic, speed limitations, and others.

D. Heuristic

As an adjective, heuristic pertains to the process of gaining knowledge by making anintelligent guess rather than by following some pre-established formula [2] [3]. Most of what people do in their daily lives involves heuristic solutions. In map problems, when moving from one point to another to reach a certain destination, there are two options. In the first option, the algorithm tries all possible paths from all possible neighbours (next address on the way to destination). It keeps doing this until the destination is reached. Finally, it chooses the best path among all possibilities. In the second option, at each location, the algorithm chooses the next move using some smart evaluation function (called the heuristic function).

E. Navigating Using Heuristic Functions and Hamilton Circuit

This paper addresses the issue of navigating to multiple destinations in any order. The main problem is to find the fastest path starting at a given source and passing over all given destinations, in any order. The importance of the proposed approach is that existing solutions, such as Google

Maps [8], let users select the order of destinations rather than suggesting the fastest path. Moreover, calculating the fastest path using the traditional mathematical algorithms like Hamilton path [1] has a high time complexity for real-time large graphs representing real cities. As a result, the use heuristic algorithms like A*, substantially minimize the graph size and hence minimize the Hamilton algorithm running time for such navigation real-time solutions. The Hamilton circuit definition, algorithm, and examples are presented in Section II.

The paper is organized as follows: Section II presents some related work, including widely used applications. Section III presents the main solution of this paper. Section IV discusses our results and, finally, Section V presents conclusions and future work.

II. BACKGROUND AND RELATED WORK

This section presents the relevant background, including definitions, notations, and algorithms, used in the proposed approach. Some terms used, such as graph, vertex, edge and others assume prior knowledge of these data structures.

A. *Artificial Intelligent Heuristic Algorithm A**

A* [2] is an Artificial Intelligent graph algorithm proposed by Pearl. The main goal of A* is to find a cheap cost (time) graph path between two vertices in a graph using a heuristic function. The goal of the heuristic function is to minimize the selection list at each step. In the graph example, finding the shortest path from a node to another has to be done by getting all possible paths and choosing the best, which is very expensive when having a huge number of nodes. On the other hand, using an evaluation function (heuristic) to minimize the problem choices according to an intelligent criterion would be much faster. In case of A* algorithm, the heuristic function H (S, D) is defined as follows:

Input: a source vertex S and a destination D.
Task: evaluate S based on the destination D using the following heuristic function:
Distance_So_Far + Stright_Line_Distance (S, D)
where:
Distance_So_Far = Distance traveled so far to reach vertex S.
Stright_Line_Distance (S, D) = Straight line distance from source S to destination D calculated by using their coordinates.

A* Algorithm
A*(Graph, Source, Destination)
Task: takes a Graph (Vertices and Edges), Source and Destination (Vertices) and returns the Best path solution (stack of vertices) from Source to Destination.

```
• If Source = Destination then return solution (stack)
• Else expand all neighbours Ni of Source
• Mark Source as Unvisited
• For each Neighbour Ni
    o Get Vi = H(Ni, Destination)
    o Add all (Ni, Vi) to the Fringe (list of all
      expanded Vertices)
    o From the Fringe, Choose an Unvisited
      Vertex V with Least Vi
    o If no more Unvisited return Failure
    o Else Apply A*(V, Destination)
```

The time complexity of A* is O(n^2) [2].

Figure 1 is an example of the A* algorithm behavior to find a path starting from “Arad” to “Bucharest”, cities in Romania [2]. First of all, start at Arad and go to the next neighbor with the best heuristic function (Sibiu). Second, explore all neighbors of Sibiu for the best heuristic function. The algorithm continues choosing the best next step (with the least value of the heuristic function) until it reaches Bucharest. All vertices with values (heuristic function) are kept in the fringe in order to be considered at each step.

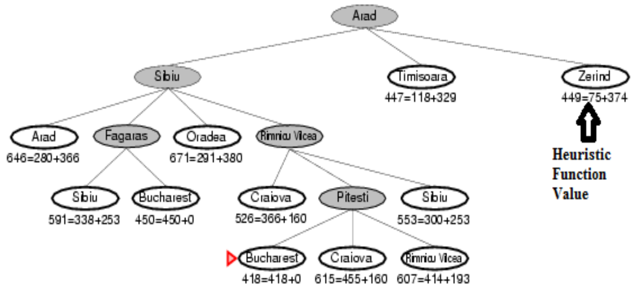


Figure 1. Calculating the path from Arad to Bucharest

B. *Graph Definitions and Notations*

This sub-section presents the graph definitions and algorithms used in the proposed approach. The time-complexities of these algorithms is briefly stated.

Definition 1. Graph G (V,E): where V is the set of vertices and E is the set of edges. Figure 2 illustrates a graph with vertices: 2,3,5,8,9, and 11 and edges: (5,11), (11,2), (11,9), (7,11), (8,9), (3,8).

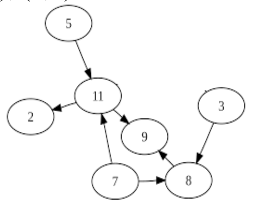


Figure 2. A sample graph

Definition 2. Complete graph: a graph without loops or multiple edges and every vertex is connected to every other vertex. See Figure 3.

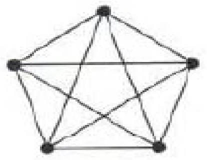


Figure 3. A complete graph

Definition 3. Hamilton circuit [1]: A path in the graph that passes over all vertices once and gets back to the source node where it started. Only the source vertex is visited twice. See Figure 4.

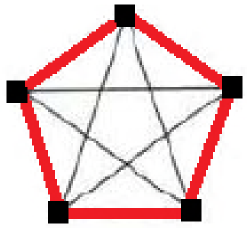


Figure 4. Hamilton Circuit

Definition 4. All permutations: It represents how many ways there are to arrange n different objects out of k objects. The mathematical formula is:

$$^{n}P_{k}=\frac{n!}{(n-k)!}= n\,(n-1)\,(n-2)\ldots(n-k+1).$$

Example: How many ways can 4 students from a group of 15 be lined up for a photograph? Answer: There are $^{15}P_4$ possible permutations of 4 students from a group of 15.

$$^{15}P_4=\frac{15!}{11!}= 15.\ 14.\ 13.\ 12 = 32760.$$

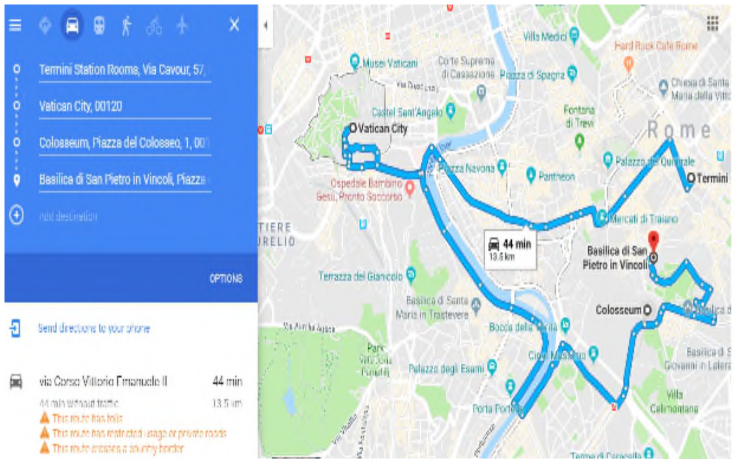
Hence, the permutation of n objects out of n objects (how many different ways to arrange n objects) will be = $\frac{n!}{(n-n)!}$ = n!.

C. *Related Work: Multi-Destinations Using Google Maps*

This subsection presents two existing solutions: Google maps [8], and a previous work A*Multiple [10].

(1) Google Maps

Google Maps [8] is a Web-based service that provides detailed information about geographical regions and sites around the world. In addition to conventional road maps, Google Maps offers aerial and satellite views of many places. Figure 5 shows an example a driving directions query using Google Maps [8]. The query is to get driving directions, over multiple destinations in Rome: Termini station, Vatican City, Coliseum, and Basilica di San Pietro. It also offers real-time traffic information. However, Google Maps [8] does not suggest any order of visiting these sites. The user has to provide Google Maps with the order and the user has to perform multiple trials and look for the best sequence of destinations to be visited. In order to make it a cycle, the user has to provide a path from the last destination to the source (Termini station).



Termini Station Rooms, Via Cavour, 57,

Vatican City, 00120

Colosseum, Piazza del Colosseo, 1, 00

Basilica di San Pietro in Vincoli, Piazza

via Corso Vittorio Emanuele II 44 min

13.5 km

Figure 5. A multi-destination Path by Google Maps where order is chosen by the user

(2) A*Multiple

The main idea behind A*Multiple [10] is to find the best path (shortest in time) to visit multiple destinations in one tour. The algorithm uses a heuristic function to find the next destination.

Algorithm 1. A*Multiple (Source, Destinations)

Task: find an efficient path from source passing over all members in the destinations array.

Returns: 2 lists, namely

1) VSL: Vertices Solution List which is an ordered list vertices that the path follows in the trip.
2) PSL: Path Solution List, which is the list of paths to take each time to each destination (vertex) from a vertex in the VSL list to another in the same list.

```
Pseudo code
If the Destination is Empty return "done".
For all Vertices Vi in Destinations
  Di=H(source, Vi)
  Get the Vs with the Minimum Di
  Remove Vs from Destinations
  Add Vs to the Vertices Solution List VSL
  Add A*Traffic (Source, Vs) to the Path Solution List PSL
 If A*Traffic fails return Failure.
A*Multiple (Vs, Destinations).
```

How does A*Multiple Work?

Next, we present the execution of A*Multiple. To present the proposed approach better, we consider the following problem: suppose the user is at Termini station, Rome and wants to visit the following destinations in Rome: Vatican City, Coliseum and Basilica di San Pietro. If the only priority is time, it means that one can visit them in any order with efficient time. In this case, one has to choose the next destination (at each step) in a smart way.

After creating the Time-Weighted graph (subset of the vertices shown in black in Figure 6) over the map of Rome (from Google Maps), the A*Multiple will return the following:
VSL: Termini station, Vatican City, Coliseum, and Basilica di San Pietro.
PSL: Path1, Path2, Path3.
where VSL is the ordered list of destinations to be visited, PSL is the list of paths from each destination in VSL to the next one, Path1 is Termini station-Vatican City, Path2 is Vatican City- Coliseum, and Path3 is Coliseum, - Basilica di San Pietro. Figure 6 shows these solutions in different colors: orange (Path1), blue (Path2) and pink (Path3). It also gives an estimated time for each path according to current (at time of calculation) traffic situation. However this is not a cycle.

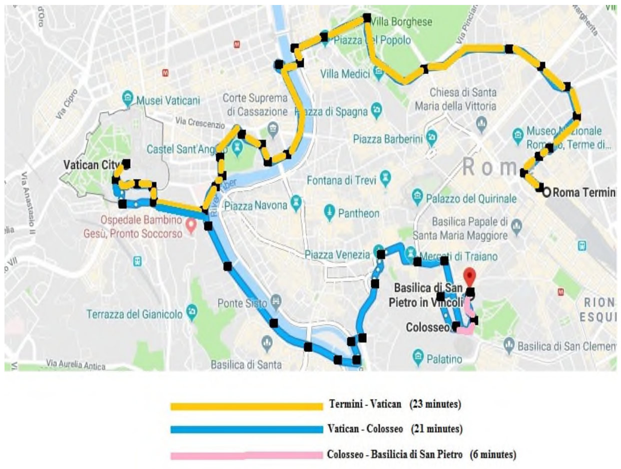

Villa Borghese
Piazza del Popolo
Villa Medici
Musei Vaticani
Corte Suprema di Cassazione
Piazza di Spagna
Chiesa di Santa Maria della Vittoria
Museo Nazionale Romano, Terme di...
Castel Sant'Angelo
Piazza Barberini
Vatican City
Roma
Roma Termini
Fontana di Trevi
Palazzo del Quirinale
Piazza Navona
Pantheon
Basilica Papale di Santa Maria Maggiore
Ospedale Bambino Gesù, Pronto Soccorso
Piazza Venezia
Mercati di Traiano
Basilica di San Pietro in Vincoli
Colosseo
Ponte Sisto
Terrazza del Gianicolo
Basilica di San Clemente
Palatino
Basilica di Santa
Termini - Vatican (23 minutes)
Vatican - Colosseo (21 minutes)
Colosseo - Basilicia di San Pietro (6 minutes)

Figure 6. Paths for Multiple destinations (Termini, Vatican City, Coliseum, Basilica di San Pietro)

III. PROPOSED APPROACH: A* HAMILTON CIRCUIT

This section presents the approach to navigate a multi-destination path starting and ending from/to a certain source. The main idea behind this approach is the following:

- Given: graph G representing the map, destination list L repressing the destinations, and source S the start point.
- Create a new virtual complete graph G1 with vertices V1=L+S and edges E1={(ai,bi),..} where edge (ai,bi) is a path calculated using A* algorithm.
- Find all Hamilton circuits in G1 starting and ending at S
- Choose the shortest

The idea behind building the virtual graph is to dramatically minimize the number of vertices of the graph where Hamilton path algorithm is to be applied. In order to present a formal pseudo-code algorithm of the proposed approach, A*HamiltonCircuit, the following algorithms are presented:

Algorithm 2. Hamilton circuit (G (V, E), S): Finds the shortest Hamilton circuit (see Figure 4) in graph G starting and ending at source S.
G: Graph with vertices V and Edges E.
S: Starting node $S \in V$
Returns L: Ordered List of vertices that form the Hamilton Circuit starting and ending at S.
Algorithm:

```
1. List all permutations ($LSP_i$) of n vertices.
2. Choose permutations that start with S.
3. Add S to the end of each $LSP_i$: it becomes S,….,S
4. Choose the valid permutation from $LSP_i$ where $\forall$ i (vi,vi+1) $\in$ E
5. Choose the shortest
```

The time complexity of Algorithm 2 is as follows:
Step1: n! where n is the number of vertices
Step2: n!
Step3: (n-1)!
Step4: n^2 * (n-1)!
Step5: n

The total time complexity is (n^2+4) n! which is exponential-time algorithm O(n!) and, hence, time consuming for high values of n. Figure 7 shows one result out of many (24 in this case) executions of the Hamilton circuit algorithm starting from vertex v1 all the way back to v1. Later, the shortest path is chosen.

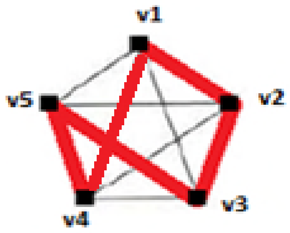


Figure 7. Hamilton circuit starting and ending at v1

Algorithm 3. BuildA*Graph (G(V, E), L): Build a complete virtual graph using the smart A* algorithm

G: graph with vertices V and edges E
L: List of destinations ($L \in V$)
Returns G1(V1, E1): a virtual complete graph with the list of vertices V1 (equal to L) and set of virtual edges E1 where each edge in E1 refer to a path (list of real edges from E) computed using A*.

```
1. For each vertex Vi in L.
2. Using A*, find all paths from Vi to all other destinations and then to E1.
3. Using these paths, build the Virtual Complete Graph G1(V1,E1).
```

The time complexity of Algorithm 3 is as follows:
Step1: O (m*n^2), where m is the number of vertices in the destinations list L and n^2 is A* time complexity.

Step2: O (m^2) (since G1 has m vertices and maximum of m^2 edges.

Step3 is constant.

As a result, the total time complexity will be O (n^2) since m will be considered a constant compared to n (<=10).

Figure 8 shows the actual graph. Figure 10 presents the extraction (using BuildA*Graph) of the virtual graph. The edges in Figure 9 are built using A*. Each one of the edges represents a path with multiple vertices. Each of these paths will be used as a single edge when applying the Hamilton path algorithm on the virtual graph. For example, the path from v1 to v5 is p1, the path from v5 to v2 is p2, and so on. Figure 10 is an example of the virtual graph. For example, the edge (v1, v5) with weight 45 in Figure 10 represents a real path p1 in Figure 9 calculated using A* algorithm. The weights of these edges are the weights of the calculated path. Hence 45, the edge weight of (v1, v5) is the weight of p1 calculated using A*. Note that, for simplicity of examples, the graph in Figure 8 is used as un-directed graph.

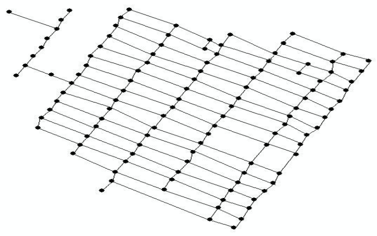


Figure 8. Initial actual graph

Looking at Figure 10, examples of paths starting from v1 (using StartHamilton algorithm) are:

Path1= v1, v2, v3, v4, v5 with weight = 120 +124 +112+ 135 = 491

Path2= v1, v3, v4, v5, v2 with weight = 114+ 112+ 134+221= 581

There will be other 24 options. The option with the lowest weight (shortest) will be chosen.

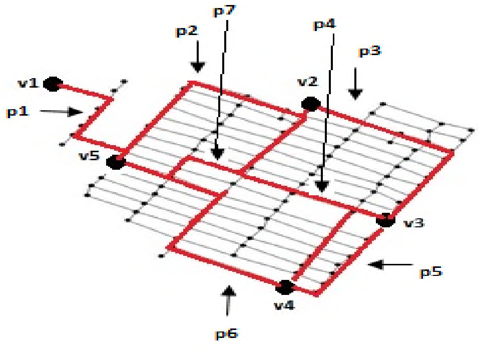


Figure 9. Paths calculated using A*

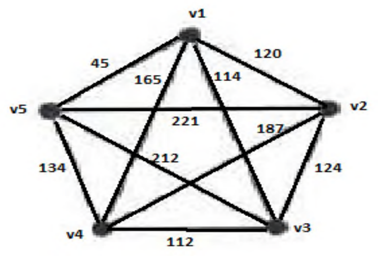


Figure 10. The complete virtual graph extracted from graph is figure 9

Algorithm 4. A*HamiltonCircuit (Graph G (V,E), L, S): Finds the shortest path from a source passing all desired destinations. It uses algorithms 2 and 3 to build a new virtual graph and applies the Hamilton circuit algorithm on it.

```
G: Graph with vertices V and Edges E
S: Start Vertex that belong to V#
L: List of destinations (L∈ V)
1- G1(V1,E1) = BuildA*Graph (G, L)
2- HP = HamiltonCircuit (G1, S ) (Find the shortest
   Hamilton Circuit in G1 that start with S∈V1)
```

The time complexity of Algorithm 4 is as follows:

Step1: O (n^2), where n is the number of vertices in the destinations list

Step2: O (m!), finding all permutations (possible paths) of m vertices out of m vertices.

The total time complexity will be in O (n^2 + m!). If m is a relatively small number (<= 10), its maximum time will be around 3 seconds. Example: 10! = 3,628,800 steps (around 3 seconds to compute), then A*Hamilton will be acceptable.

IV. RESULTS

In this section, we discuss the results of some sample executions using our proposed approach.

A testing tool is developed where 120 samples using 6142 vertices were tested in 2 groups: Group 1 (between 10 and 15 destinations), Group 2 (less than 8 destinations).

Results shown in Table I where:

- Optimal solution represents the absolute best solution.
- Good solution takes maximum of 20% more time than the optimal solution.
- Bad solution takes more than 20% more time than the optimal solution.

TABLE I. PERCENTAGES OF QUALITY OF SOLUTIONS

Number of Destinations	Optimal solution	Good Solution	Bad Solution
Between 10 & 15 destinations Over 6142 vertices	81.6 %	14.3%	4.1%
Less than 8 destinations Over 6142 vertices	97.8%	2.1%	0.1%

Comparing these results to the previous results (81% average) [10] shows a very good progress. Note that existing online solutions do not offer such options and hence comparison with those solutions is not applicable.

V. CONCLUSION AND FUTURE WORK

The approach proposed in this paper offers a TSP solution (full cycle path) with an order of destinations claiming an efficient time. To find a solution the following was done:

- Build a real graph G (V,E) that represents the map.
- Build a complete virtual graph G1 (V1, E1) where V1 is the set of destinations and E1 is the set of edges between these destinations. E1 represents a paths intelligently calculated with the smart algorithm A*[2].
- Calculate the shortest cycle path from the selected source (vertex) using HamiltonCircuit Algorithm (Algorithm 3).

The following are the two main concerns:

1. Knowing that HamiltonCircuit Algorithm's time complexity is exponential, its effect is null when applied on a small number of destinations.
2. The weights of edges are not guaranteed to be the best as A* does not guarantee that.

For future work, finding good heuristic functions is a challenge. This is an open research question and highly dependent on the geography of the surface in the query.

REFERENCES

[1] K. Ross and C. Wright, *"Discrete Mathematics"*. Prentice Hall, Upper Saddle River, New Jersey, 2003.

[2] S. Russell and P. Norving, "*Artificial Intelligence a Modern Approach"*. Prentice Hall, Upper Saddle River, New Jersey, 2003.

[3] J. Pearl, "*Heuristics: Intelligent Search Strategies for computer Problem Solving"*. Addison Wesley, Reading, Massachusetts, 1984.

[4] H. Halaoui, "Smart Traffic Online System (STOS): Presenting Road Networks with time-Weighted Graphs". IEEE International Conference on Information Society (i-Society 2010) London, UK. June 2010, pp. 349-356.

[5] Google Earth Blog Google Earth Data Size, Live Local, New languages coming. Available: http://whatis.techtarget.com/definition/Google-Maps. Retrieved: September, 2015.

[6] H. Halaoui, "Smart Traffic Systems: Dynamic A*Traffic in GIS Driving Paths Applications". Proceeding of IEEE CSIE09, IEEE, Los Angeles, USA. March, 2009, pp. 626-630.

[7] H. Halaoui," *Intelligent Traffic System:* Road Networks with Time-Weighted Graphs*"*. International Journal for Infonomics (IJI), Volume 3, Issue 4, December 2010, pp. 350-359.

[8] Google Maps. Available: https:// Maps.google.com. Retrieved: September, 2015.

[9] H. Halaoui, "Spatial and Spatio-Temporal Databases Modeling: *Approaches for Modeling and Indexing Spatial and Spatio-Temporal Databases"*. VDM Verlag, 2009.

[10] Hatem Halaoui, "SMART NAVIGATION: Using Artificial Intelligent Heuristics in Navigating Multiple Destinations". Proceedings of SOTICS 2015 (The Fifth International Conference on Social Media Technologies, Communication, and Informatics). Barcelona, Spain. November, 2015.

[11] E. L. Lawler, J. K. Lenstra, A. H. G. Rinooy Kan, & D. B. Shmoys (Eds.), "*The traveling salesman problem"* (pp. 145–180). Chichester: John Wile, 1985.