Intelligent Hamilton Path: Using Artificial Intelligent A* Algorithm and Hamilton Path to Navigate Multiple Destinations

Hatem F. Halaoui(✉)

Computer Science, Haigazian University, Beirut, Lebanon
hhalaoui@haigazian.edu.lb

Abstract. Navigation applications are becoming an essential need for travellers holding mobile devices. Finding the best path (time and distance) from one address to another is one of the most asked queries by travellers. A more difficult problem is finding the best path to visit multiple destinations in a single trip, which could be a common query for many including sales people, tourists, and delivery drivers. Google maps, Yahoo maps, and the like are examples of navigation mobile 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 algorithms A*, traditional graph algorithms like Hamilton path as well as efficient data structures in finding efficient path between multiple addresses. It presents graph algorithms and notations, existing smart graph algorithms, heuristics in graph problems, and finally a smart solution, A* Hamilton, to determine the best path between given multiple destinations.

Keywords: Intelligent navigation algorithms · Smart navigation · Hamilton path · A* algorithm

1 Introduction

This section introduces the main topics used in the proposed approach. First, the idea of heuristics is briefly presented. Second, Spatial Databases and the main data warehouse of our approach are briefly presented. Third, a brief introduction to Geographical Information Systems (GIS) and driving path application is given. Finally, a briefing of the adopted approach is also presented.

1.1 Heuristics

As an adjective, heuristic pertains to the process of gaining knowledge by intelligent 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: (1) the algorithm tries all possible paths from all possible neighbours (next address on the way to destination). It keeps doing this until destination is reached. Finally, it chooses the best path among all possibilities; (2) At each location, the algorithm chooses the next move smartly using some evaluation function (called the heuristic function).

The first option is very time consuming and does not match with real-time problems unless the options are little (below 10 graph vertices), hence use it after decreasing the map (graph) vertices. A solution using heuristics is also being adopted to decrease the map (graph) vertices.

1.2 Spatial Databases

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

1.3 Geographical Information Systems and Driving Path Applications

GIS is a collection of computer hardware, software for capturing, managing, analysing, and displaying all forms of geographical information [6, 7]. Finding the direction (driving/walking) is one of the most asked queries in GIS applications. The most important factors influencing such criteria are distance road situation, road traffic, speed limitations, and others.

1.4 Navigating Using Heuristic Functions and Hamilton Path

This paper presents the issue of navigating 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 like Google Maps [8] let the user choose his order of destinations rather than suggesting a fast path.

Moreover, calculating the fastest path with traditional mathematical algorithms like Hamilton path [1] has a high time complexity for real-time large graphs representing real city maps. As a result, use heuristic algorithms like A* to incredibly minimize the graph size and hence minimize the Hamilton algorithm running time for such navigation real-time solutions. Hamilton path definition, algorithm, and examples are presented in Sect. 2.

The paper is organized as follows Sect. 2 presents some related work including widely used applications. Section 3 presents the main solution in this paper. Section 4 discusses some results, conclusions and future work.

2 Background and Related Work

This section presents the main subject's background including definitions, notations, and algorithms used in the proposed approach. Some used terms like graph, vertex, edge and others assume prior knowledge of these data structures. It also presents some related and similar existing work.

2.1 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 intelligent criterion would be much faster. In case of A* algorithm, the heuristic function is 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 taken so far to reach the vertex S
Stright_Line_Distance (S, D) = straight line distance from 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
    – Get Vi = H(Ni, Destination)
    – Add all (Ni, Vi) to the Fringe (list of all expanded Vertices)
    – From the Fringe, Choose an Unvisited Vertex V with Least Vi
    – If no more Unvisited return Failure
    – Else Apply A*(V, Destination)
```

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

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

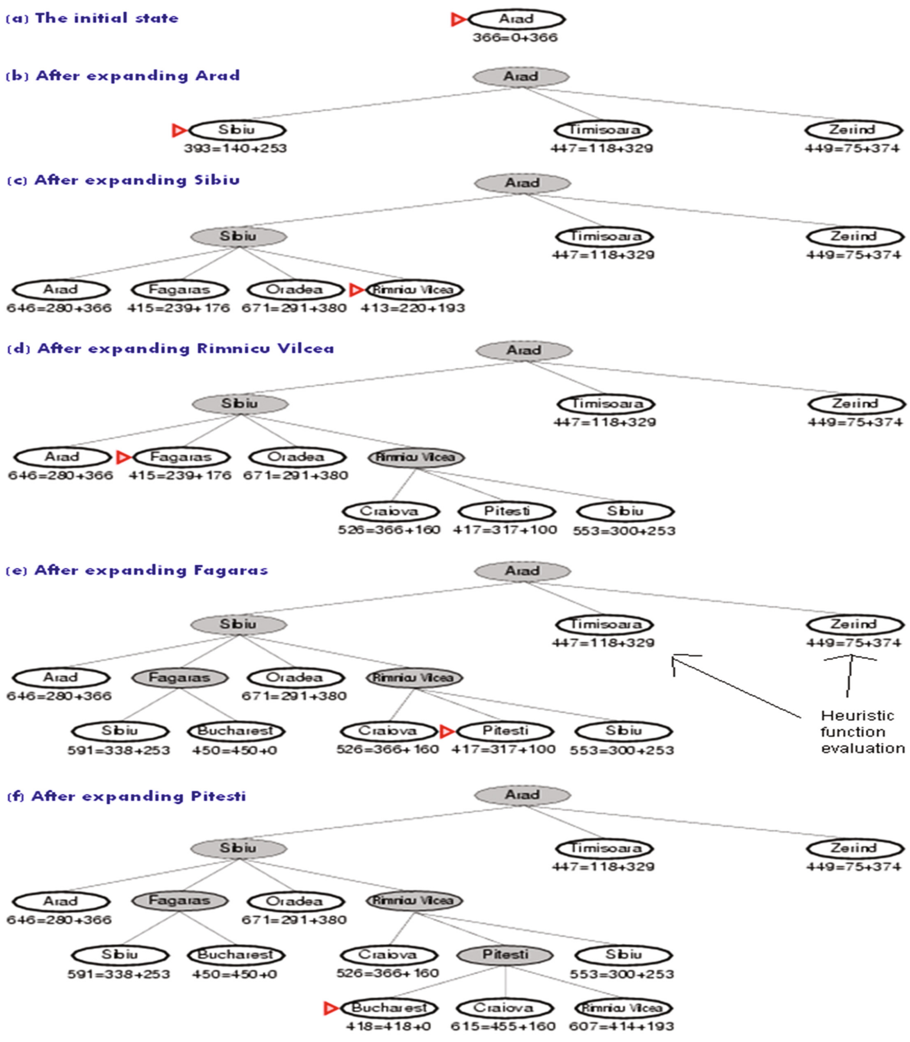


Fig. 1. Calculating the path from Arad to Bucharest

2.2 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).

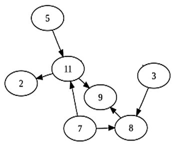


Fig. 2. A sample graph

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

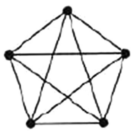


Fig. 3. A complete graph

Definition 3. Hamilton Path [1]: A path in the graph that passes over all vertices once. See Fig. 4.

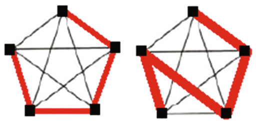


Fig. 4. Examples of Hamilton paths

Definition 4. All permutations: It is how many ways to arrange different n objects out of k objects. The Mathematical proven 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 = $\frac{n!}{(n-n)!} = n!$.

Algorithm 1. Hamilton Path (G (V, E)): Finds a Hamilton path (figure 4) in graph G.

G: Graph with vertices V and Edges E.

Returns L: Ordered List of vertices that form the Hamilton Path.

Algorithm:

1) List all permutation of n vertices in V: vi, vi+1, vi+2, …, vn
2) Choose the valid permutation where $\forall$ i (vi,vi+1) $\in$ E

Algorithm 1 time Complexity:

Step 1: n! Where n is the number of vertices

Step2: n*n!

Total is (n+1) n! Which is an exponential-time algorithm O(n!) and hence time consuming for high values of n.

2.3 Related Work: Multi-destinations Using Google Maps

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

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 of driving directions query using Google Maps [8]. The query is to get driving directions over multiple destinations in London: Paddington station, Harrods, House of Commons, and London Eye. It also offers Real-time Traffic information. However, Google Maps [8] does not suggest any order of visits. The user has to provide Google Maps with the order and he has to make multiple trials and look for the best sequence of destinations to be visited.

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 and then uses the A*Traffic (which also use the same heuristic function) to travel to that destination.

Algorithm 2. A*Multiple (Source, Destinations)

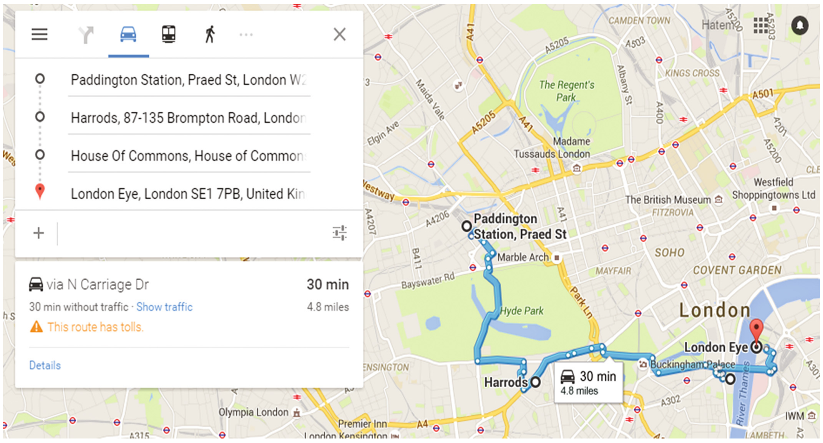
Task: find an efficient path from source passing over all members in Destinations array.

Returns: 2 Lists

VSL: The Vertices Solution List VSL, which is an ordered list vertices that path follows in the trip.

PSL: Path Solution List PSL, which is the list of paths to take each time to each destination (vertex) from a vertex in the list VSL to another in the same list.

```
    Pseudo code
    If 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
```


Paddington Station, Praed St, London W
Harrods, 87-135 Brompton Road, Londo
House Of Commons, House of Common
London Eye, London SE1 7PB, United Ki

via N Carriage Dr 30 min
30 min without traffic · Show traffic 4.8 miles
This route has tolls.
Details

Fig. 5. A multi-destination path by Google Maps where order is chosen by the user

If A*Traffic fails return Failure.
A*Multiple (Vs, Destinations).

- It is time complexity is O(n^2) [10]
- How does A*Multiple Work? [10]

This section presents the execution of A*Multiple. To present the proposed approach better, consider the following problem: Suppose I am at Paddington station and want to visit the following destinations in London: “Eye of London”, “House of Commons”, and “Harrods”. If my only priority is time, means that I can visit them in any order with efficient time. In this case, I have to choose my next destination (at each step) smartly.

After creating the Time-Weighted graph (vertices shown in black in Fig. 7, over 5000 vertices) over the map of London (from Google Maps), the A*Multiple will return the following:

VSL: Harrods, House of Commons, Eye of London.

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: Paddington – Harrods, Path2: Harrods – House of Commons, and Path3: House of Commons – Eye of London.

Figure 6 shows these solutions in different colours: orange (Path1), Blue (Path2) and Pink (Path3). It also gives estimated time of each path according to current (at time of calculation) traffic situation.

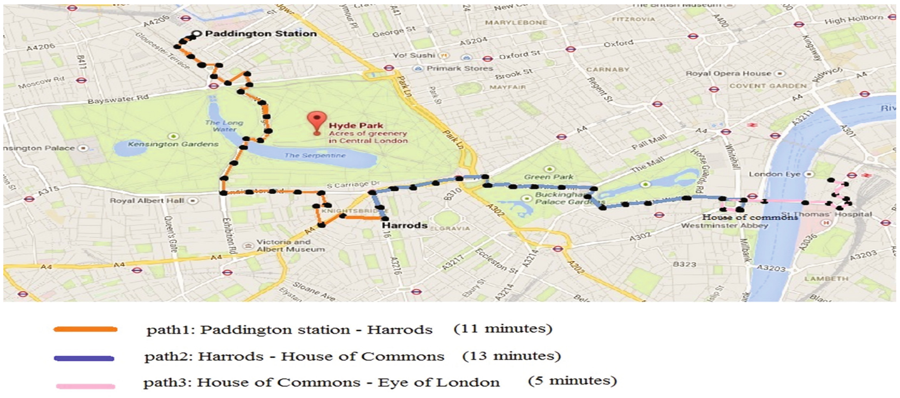

Paddington Station
Hyde Park
Harrods
House of commons
London Eye
path1: Paddington station - Harrods (11 minutes)
path2: Harrods - House of Commons (13 minutes)
path3: House of Commons - Eye of London (5 minutes)

Fig. 6. Paths for multiple destinations (Paddington, Harrods, House of Commons)

3 Proposed Approach: A*Hamilton

This section presents the approach to navigate a multi-destination path starting from 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 paths in G1 starting at S
- Choose the shortest

The main 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*Hamilton, the following algorithms are presented: StartHamilton, BuildA*Graph, and finally the main solution algorithm A*Hamilton.

Algorithm 3. StartHamilton (S, G(V,E)): The algorithm finds all Hamilton paths staring from S passing over all vertices in V.

G: Graph with vertices V and Edges E

S: Start Vertex that belong to V

Returns L: a list of ordered Lists of vertices (Hamilton paths) starting from S

```
1) List all permutation of n vertices in V: vi, vi+1, vi+2, …, vn.
2) Choose all valid permutations $VP_j$ where $\forall$ i (vi,vi+1) $\in$ E
3) For each valid permutation VPj: Choose the ones that start with S and Add them to L
4) Return L
```

Time Complexity:

Step 1: n! where n is the number of vertices

Step 2: n*n!

Step 3: n!

Step 4: Constant

Total is (n+2) n! Which is an exponential-time algorithm O (n!) and hence time consuming for high values of n.

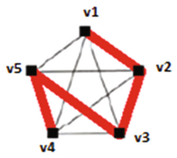


Fig. 7. An example of Hamilton path starting at v1

Figure 7 shows one result out of many (24 in this case) of the execution of the StartHamilton algorithm starting from vertex v1 all the way to v5.

Algorithm 4. 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 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 the Destinations List L
2) Using A*, find all paths from Vi to all other destinations and them to E1.
    a) Using these Paths build the Virtual Complete Graph G1(V1,E1) with v1= L as set of Vertices and E1 calculated in step i.
```

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.

As a result, it will be O (n^2) since m will be considered a constant compared to n (assume between 0 and 10 destinations).

Figure 9 shows the actual graph. Figure 10 presents the extraction (using algorithm BuildA*Graph) of the virtual graph. The edges in Fig. 9 are built using A*. Each of the edges represent a path with multiple vertices. Each of these paths will be used as a single edge when applying Hamilton path algorithm on the virtual graph. Examples: The path from v1 to v5 is p1, the path from v5 to v2 is p2, the path from v2 to v3 is p3, the path from v3 to v4 is p5, the path from v5 to v3 is p7, the path from v5 to v4 is p6, and so on where all paths from each vertex in the destinations list to each other vertex in the same list is calculated and considered as an edge is the virtual graph. The virtual graph will look like the one in Fig. 11 where each edge is a calculated path. Example edge (v1, v5) with weight 45 in Fig. 11 will be the real path p1 in Fig. 10 calculated using A* algorithm. The weight of these edges are the weight 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, graph in Fig. 8 is used as un-directed graph whereas real-time graphs are directed and edges in opposite direction could have different weights.

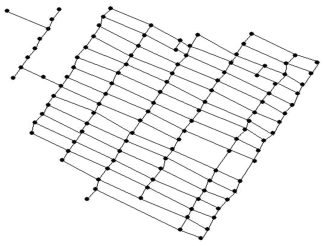


Fig. 8. Initial actual graph

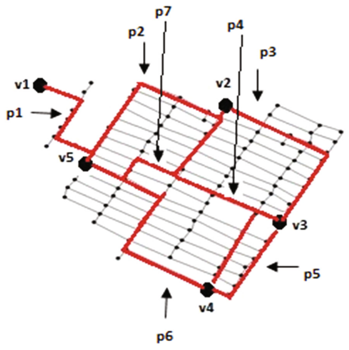


Fig. 9. Paths between vertices calculated using A8

Looking at Fig. 9, 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
Path3 = v1, v5, v4, v3, v2 with weight = 45 + 134 + 112 + 124 = 415

There will be another 24 options. The option with the lowest weight (shortest) will be chosen. Figure 10 shows the virtual graph output of Algorithm 4.

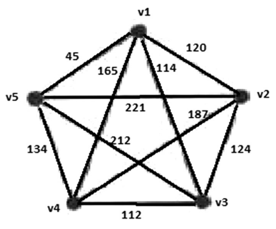


Fig. 10. The complete virtual graph extracted from graph is Fig. 9

Algorithm 5. A*Hamilton (Graph G (V, E), L, S): Finds the shortest Path from a source passing all desired destinations. It uses algorithms 4 and 5 to build a new virtual graph and apply Hamilton algorithm on it.

G: Graph with vertices V and Edges E

S: Start Vertex that belong to V#

L: List of destinations ($L \in V$)

```
1) G1(V1,E1) = BuildA*Graph (G, L)
2) HP = StartHamilton (S, G1 ) (Find all Hamilton Paths
(Set HP) in G1 that start with S∈V1)
3) Choose the shortest in HP
```

Time complexity

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.

Step3: O (m!) (Choosing the best from the m!)

The total will be in O (n^2 + m!)

If m is a relatively a 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.

4 Results, Conclusions and Future Work

Section 4 discusses results of sample executions, some conclusion, and future ideas.

4.1 Results

A testing tool is developed (to test the proposed approach) where 100 samples were tested in 2 groups: Group 1 (Between 7 and 11 destinations Over 5321 vertices), Group 2 (less than 7 destinations Over 5321 vertices) Results showed that the proposed solution is optimal in 88.5%. Table 1 presents the gathered results in each group/each case where:

- Optimal solution: Absolute best solution.
- Good solution: takes maximum of 20% more time than optimal solution.
- Bad solution: Takes more than 20% more time than optimal solution.

Table 1. Percentages of quality of solutions

Distances	Optimal solution	Good solution	Bad solution
More than 7 destinations	81%	13%	6%
Less than 11 destinations			
Over 5321 vertices			
Less than 7 destinations	96%	3%	1%
Over 5321 vertices			
Average	88.50%		

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

4.2 Conclusions

The approach proposed in this paper offers the user a full path solution for a multiple destination trip 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 where V is the set of vertices (real addresses) and E set of edges (real directed pieces of the roads)
- Build a complete virtual graph G1 (V1, E1) where V1 is the set of destinations and E1 is the edges between these destination. Each of these edges represent a path intelligently calculated with the smart algorithm A*[].
- Find all possible paths from a selected source (vertex) using StartHamilton Algorithm. Then choose the best.

The following are the two main concerns:

(1) Even though StartHamilton is exponential, however its effect is null when applied on small number of destinations.

(2) When building the complete virtual graph, the weight between edges is not guaranteed to be the best. The reason for such thing is that A* does not guarantee an optimal path between two edges.

4.3 Future Work

The main concern is that heuristic functions used in A* does not guarantee an optimal (best) solution. For this reason, choosing the heuristic function is an important factor for getting good results. Choosing a good heuristic function in order to choose the series of destination is an open research question and highly dependent on the geography of surface in query.

References

1. Ross, K., Wright, C.: Discrete Mathematics. Prentice Hall, Upper Saddle River (2003)
2. Russell, S., Norving, P.: Artificial Intelligence a Modern Approach. Prentice Hall, Upper Saddle River (2003)
3. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison Wesley, Reading (1984)
4. Halaoui, H.: Smart Traffic Online System (STOS): presenting road networks with time-weighted graphs. In: IEEE International Conference on Information Society (i-Society 2010), London, UK, pp. 349–356, June 2010
5. Google Earth Blog Google Earth Data Size, Live Local, New languages coming. http://whatis.techtarget.com/definition/Google-Maps. Accessed Sep 2015
6. Halaoui, H.: Smart traffic systems: dynamic A*Traffic in GIS driving paths applications. In: Proceeding of IEEE CSIE 2009, pp. 626–630. IEEE, Los Angeles, March 2009
7. Halaoui, H.: Intelligent traffic system: road networks with time-weighted graphs. Int. J. Infonomics (IJI) **3**(4), 350–359 (2009)
8. Google Maps. https://Maps.google.com. Accessed Sep 2015
9. Halaoui, H.: Spatial and spatio-temporal databases modeling. In: Approaches for Modeling and Indexing Spatial and Spatio-Temporal Databases. VDM Verlag (2009)
10. Halaoui, H.: Smart navigation: using artificial intelligent heuristics in navigating multiple destinations. In: Proceedings of SOTICS 2015 (The Fifth International Conference on Social Media Technologies, Communication, and Informatics), Barcelona, Spain, November 2015