Smart Solution for the "Traveling Salesman Problem": Using Artificial Intelligent A* algorithm and Hamilton Circuit

Hatem F. Halaoui
Computer Science and Mathematics Division
Haigazian University
Beirut, Lebanon

Summary

Traveling Salesman Problem (TSP) is one of the famous path problems. Solving it efficiently with real-time factors (traffic, distance, real-time delays) is very useful for multiple navigation queries. Finding the best path (time and distance) to visit multiple destinations in a single trip as in TSP, could be a common query for many including sales people, tourists, delivery drivers and others. 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 circuit as well as efficient data structures in finding efficient cycle for the Traveling Salesman Problem (TSP).

KEYWORDS: Traveling salesman problem, Intelligent Navigation Algorithms; Smart Navigation; Hamilton circuit; A* Algorithm.

I. Introduction

This section introduces the main topics behind the proposed approach. First, the "Travelling Salesman Problem" (TSP) is defined then the idea of Heuristics is presented and finally, a briefing of the adopted approach is presented.

1.1 Traveling Salesman Problem (TSP)

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

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 Navigating Using Heuristic Functions and Hamilton Circuit

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 circuit definition, algorithm, and examples are presented in section 2.

The paper is organized as follows Section 2 presents some related work including widely used applications. Section 3 presents the main solution in this paper. Section 4 discusses some results and finally section 5 presents 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.

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 neighbors Ni of Source
- Mark Source as Unvisited
- For each Neighbor 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.

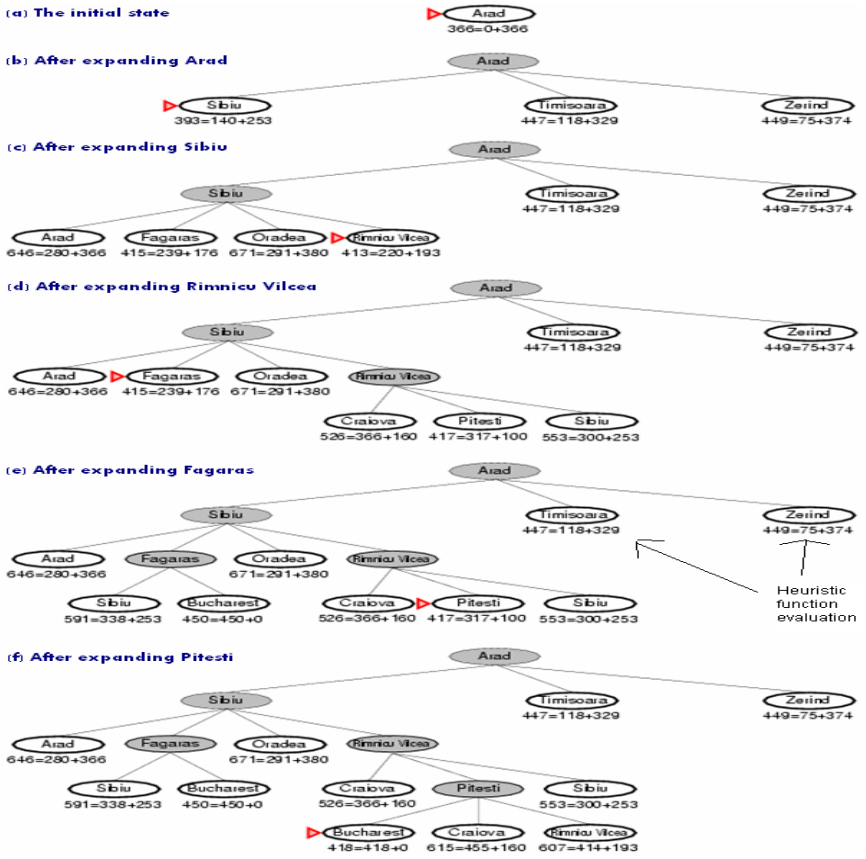

Figure 1 Calculating the path from Arad to Bucharest

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 verticies:2,3,5,8,9, and 11 and Edges: (5,11), (11,2), (11,9), (7,11), (8,9), (3,8).

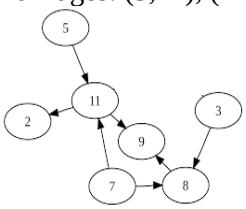

Figure 2 A sample graph

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

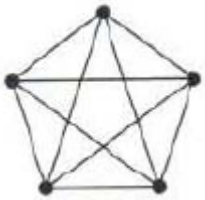

Figure 3 A complete graph

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

Figure 4 Hamilton Circuit

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 $= \frac{n!}{(n-n)!} = n!$.

3. Proposed Approach: A*HamiltonCircuit

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 Circuits in G1 starting and ending 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*HamiltonCircuit, the following algorithms are presented:

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

1. List all permutations of n vertices in V: vi, vi+1, vi+2, …, vn
2. Choose permutations (LSP_i) 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

Algorithm 2 time Complexity:
Step 1: n! Where n is the number of vertices
Step2: n!
Step3: (n-1)!
Step4: n^2 * (n-1)!
Step5: n

Total is around (n^2+4) n! Which is an exponential-time algorithm O (n!) and hence time consuming for high values of n.

Figure 5 shows one result out of many (24 in this case) of the execution of the Hamilton Circuit algorithm starting from vertex v1 all the way back to v1. Later the shortest is chosen out of the 24.

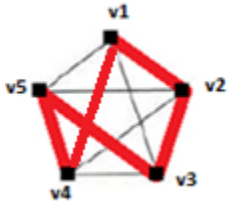

Figure 5 An example of Hamilton circuit starting and ending at v1

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 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 6 shows the actual graph. Figure 8 presents the extraction (using algorithm BuildA*Graph) of the virtual graph. The edges in figure 7 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 circuit 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 figure 11 where each edge is a calculated path. Example edge (v1, v5) with weight 45 in figure 11 will be the real path p1 in figure 9 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 figure 8 is used as un-directed graph whereas real-time graphs are directed and edges in opposite direction could have different weights.

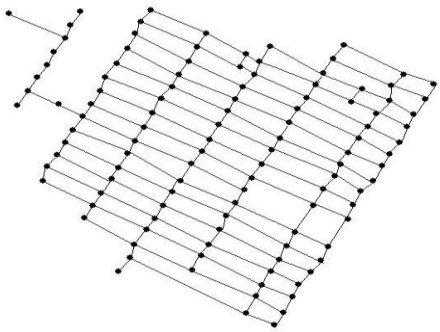

Figure 6 Initial actual graph

Looking at figure 7, 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 8 illustrates the virtual full graph connecting all nodes to each other where each path is calculated using A*.

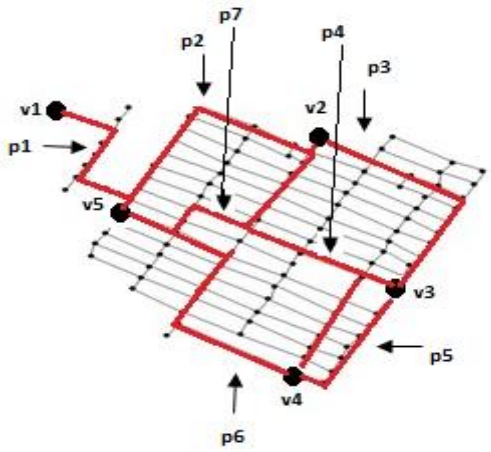

Figure 7 Paths between vertices calculated using A8

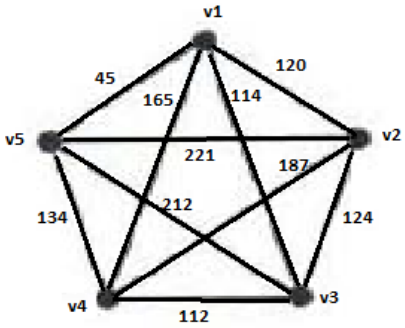

Figure 8 The complete virtual graph extracted from graph is figure 9

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 apply 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)

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.
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

Section 4 discusses results of sample executions, some conclusion, and future ideas.
A testing tool is developed (to test the proposed approach) where 100 samples were tested in 2 groups: Group 1 (Between 7 and 11destinations Over 5321 vertices), Group 2 (less than 7 destinations Over 5321 vertices)

Results showed that the proposed solution is optimal in 88.5%. Table I 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 Less than 11 destinations Over 5321 vertices	81 %	13%	6%
Less than 7 destinations Over 5321 vertices	96%	3%	1%
Average	88.5%		

Comparing these results to the previous results (81% average) [10] will show a very good progress.

References

K. Ross and C Wright (2003). *Discrete Mathematics*. Prentice Hall, Upper Saddle River, New Jersey.

S. Russell and Peter Norving (2003). *Artificial Intelligence a Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey.

J. Pearl(1984). *Heuristics: Intelligent Search Strategies for computer Problem Solving*. Addison Wesley, Reading, Massachusetts.

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

Google Earth Blog Google Earth Data Size (2015). Live Local, New languages coming Available: http://whatis.techtarget.com/definition/Google-Maps.

Hatem Halaoui (2009). *Smart Traffic Systems: Dynamic A*Traffic in GIS Driving Paths Applications*. Proceeding of IEEE CSIE09, pp. 626-630.

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

Google Maps (2015). Available: https:// Maps.google.com.

Hatem Halaoui (2009) *Spatial and Spatio-Temporal Databases Modeling: Approaches for Modeling and Indexing Spatial and Spatio-Temporal Databases".* VDM Verlag.

Hatem Halaoui (2015). "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.