

Intelligent Transportation Systems Traveling Salesman Problem (ITS-TSP) - A Specialized TSP with Dynamic Edge Weights and Intermediate Cities

Jeffrey Miller, Sun-il Kim, Timothy Menard
Department of Computer Systems Engineering
University of Alaska, Anchorage
{jmiller, kim, tsmenard}@uaa.alaska.edu

Abstract – In this paper we present the Intelligent Transportation Systems Traveling Salesman Problem (ITS-TSP), which is a heuristic algorithm loosely based on the traditional TSP with three variations: the edge weights can change constantly, not every node in the graph must be visited, and simple cycles can exist. This problem has direct application to the transportation sector where vehicles leave from a source and need to visit a certain set of locations before returning back to the source. The ITS-TSP algorithm is analyzed to show a worst case running time of $O(V^3)$, assuming that all V nodes in a graph must be visited. There is a pre-processing cost of $O(V^2E)$ that must be incurred, though this must only be performed one time for a graph. The algorithm is a heuristic that provides routes that are optimal based on a snapshot of the graph, though as the edge weights change over time, the solution *may* not be optimal. On a graph of the transportation network in Anchorage, Alaska, we tested the ITS-TSP with live data gathered through vehicle-tracking devices installed in 65 vehicles. With the weight on each edge representing the amount of time to traverse a roadway, the ITS-TSP algorithm always computed the route with minimum cost based on the snapshot of the network.

I. INTRODUCTION

Assume that you are the driver of a delivery vehicle with a certain set of stops that need to be made each day. How would you determine the order in which to make the stops? If you are interested solely in distance, you could create a graph of the transportation network and weight each edge as the distance of the roadway it represents, allowing a solution to the Traveling Salesman Problem (TSP) to determine the *shortest* route. If you are interested solely in time, you could weight each edge in the transportation graph as the amount of time to traverse the roadway it represents, allowing a solution to the TSP to determine the *fastest* route. Unfortunately, finding a solution to the TSP is quite inefficient and only operates on edge weights that do not change. If the weight of any of the edges changes, the non-polynomial time TSP algorithm will have to be run again. However, since the TSP assumes that the start node is also the destination node and that all of the nodes in the graph must be traversed exactly once, the TSP is not able to be re-executed after the driver has begun traveling along the pre-determined route. Further, unless the graph can be redrawn

so that the only vertices that are part of the graph are the ones that must be visited, the TSP algorithm would not provide an appropriate solution for a delivery vehicle driver.

Obvious modifications must be made to the TSP to make it useful for transportation-related problems. The Intelligent Transportation Systems Traveling Salesman Problem (ITS-TSP) addresses the unique aspects of dynamic transportation networks in relation to the TSP. Three variations on the TSP are required in the ITS-TSP:

1. Edge weights can change constantly in transportation-based graphs when they are based on the time to traverse the roadway they represent.
2. Not every node in a transportation graph will need to be visited on every route, so a route that visits only a predefined subset needs to be determined.
3. A node can be visited more than once if that provides a faster route, since the edge weights can change while traveling.

Since the weights change in real-time, the solution to the TSP at a specific snapshot in time may not be the solution determined by the ITS-TSP, which will change based on the current edge weights and the current location of the vehicle. As the entire set of intermediate nodes must be traversed (which will be a subset of the total set of nodes in the graph), local routing decisions must take into account the remaining nodes that must be traversed. To determine the efficiency of the final path, the optimal path with respect to the changing edge weights would need to be determined. As we cannot determine future roadway conditions to prove the optimality of the ITS-TSP, we will compare the route determined from the ITS-TSP algorithm with the optimal route as determined using brute force on the historical edge weights during the time the route was being traversed.

In this paper, we are basing our analysis and explanation on a live roadway section in Anchorage, though we have modeled it as a graph with the edge weights representing both distance and time to traverse the roadway. The data we are using is based on live data, though it has been selected to meet the specific scenarios we describe. The remainder of the paper is organized as follows. Section II provides an overview of the related work on variations of the TSP and other ITS algorithms. Section III provides an in-depth explanation of the ITS-TSP. Section IV analyzes the running time and discusses the correctness of the algorithm. Section V explains the practical optimizations that can be made to the ITS-TSP algorithm, and the conclusion and future work are provided in Section VI.

II. RELATED WORK

The development and improvement of graphing algorithms has been going on for decades, with the search for being able to calculate the shortest path between two points being the most prominent pursuit. Well known is Dijkstra's algorithm [7], where, given a graph with a set of vertices (or nodes) and edges and each edge having its own weight of a non-negative value, the output will be the shortest path to all of the nodes from a single node. Allowing negative edge weights, Bellman-Ford's algorithm [8,9] will also compute a single-source shortest path. In order to determine the shortest path from each node to every other node within a graph, an all-pairs shortest path algorithm would be required, such as Johnson's [10] or Floyd-Warshall's [11] algorithms. All of these algorithms work only with a static graph in which none of the edge weights is changing with time. If an edge weight were to change, the algorithm would have to be executed again. In order to reduce the time, dynamic fastest path (DynFast) algorithms such as those in [13,14] improve on the time and processing power of updating an edge to determine the current shortest path.

With respect to ITS applications, the edges of the graph representing a transportation network are relatively static. This enables all of the paths between all of the nodes in the graph to be precomputed, which will allow edge weights to be updated or fastest paths to be retrieved in constant time, as shown by the All Pairs All Paths Precomputed class of algorithms [3]. The ITS-TSP algorithm uses this precomputation step to utilize fast edge updates to compute the shortest, or fastest, path quickly.

The Traveling Salesman Problem (TSP) [12,16,17] is based on determining the Hamiltonian cycle with minimum cost, which only allows visiting each node once. The Dynamic Fastest Path with Multiple Unique Destinations (DynFast-MUD) algorithm [15] and ITS-TSP are loosely based on the traditional TSP with differences explained in the following section.

The Dynamic Traveling Salesman Problem (D-TSP) is a particular case of the TSP, in which additional nodes may be added to the graph during a traversal [4,5]. Another variation of D-TSP allows for updated edge weights to be considered for recalculating the shortest path [18]. The D-TSP with Time Windows [6] follows these variations, in which each node must be visited within a certain time period, and visiting a random subset of cities was discussed in [1]. Although these algorithms have similarities to the ITS-TSP, there are a few distinctions that do not allow the existing algorithms to be used without modification.

III. ITS-TSP ALGORITHM

The Intelligent Transportation Systems Traveling Salesman Problem (ITS-TSP) is loosely based on the traditional Traveling Salesman Problem (TSP). In the TSP, there is a graph $G = (V, E)$, where $V = \{v_1, \dots, v_n\}$ and $E =$

$\{e_1, \dots, e_p\}$. Each edge $e_i \in E$ connects exactly two distinct vertices from V and has a weight $w_i > 0$ associated with it. A start node, which is one of the nodes in V , represents the start and end of the overall route. The traveling salesman needs to traverse all of the vertices in V exactly once in a manner that minimizes the overall cost of the route, which is determined by summing the weight of each edge traversed.

In the ITS-TSP, there are a few variations from the traditional TSP. First of all, the weights on the edges can change constantly, so a solution to the TSP at a specific instance of time may not be the optimal solution at a different instance of time. Secondly, in the ITS-TSP, only a subset of vertices needs to be traversed rather than all of the nodes in the graph. Lastly, the traditional TSP requires the cycle to be Hamiltonian, meaning that each node can only be visited once. In the ITS-TSP, a node may be traversed more than once (though only visited once) if the optimal route requires traveling through the same node again.

These specific variations originate from the nature of a transportation network. If the vertices represent intersections and the edges represent roadways, the weight on each edge can represent the amount of time to traverse that roadway given the current traffic conditions. As traffic conditions change, the amount of time to traverse the roadways change, and the edge weights must be updated.

With one such application of the ITS-TSP concerning delivery vehicles, the fact that not every node must be visited in the graph represents that a delivery vehicle does not stop at every location to which it delivers every day. In fact, the vehicle will only stop at the locations for which it has a delivery. Each day, the number and location of stops will change, and the ITS-TSP is extensible enough to allow optimal routes to be determined based on a changing subset of intermediate nodes to visit.

The traditional TSP requires the cycle that is found to be Hamiltonian, which is also a simple cycle. With transportation networks, it may be faster to drive along a roadway that has already been traversed to arrive at another destination in a shorter amount of time. Because of this, the ITS-TSP is not limited to disallowing simple cycles, but allows cycles to exist in the route, as long as the start node of the route is the same as the destination node of the route. While traversing from one intermediate node to another, however, simple cycles are not allowed.

The ITS-TSP algorithm is provided in Figure I, with the pseudo-code provided in Figure III. An explanation of the variable types used in the pseudo-code is provided in Figure II, and an explanation of the helper functions used in the pseudo-code is in Figure IV.

IV. ITS-TSP RUNNING TIME AND CORRECTNESS

To determine the running time of the ITS-TSP algorithm, we need to analyze the loops and function calls that are included in Figure III. We will assume that the number of nodes in the entire graph is V , the number of edges is E , and the number of intermediate nodes is n .

FIGURE I. ITS-TSP ALGORITHM

INPUT = Graph, Set of intermediate nodes to visit, Start node, End node

OUTPUT = Route of minimum cost beginning at the start node, visiting all intermediate nodes, terminating at the end node

Step 1 – Create a sub-graph of the original graph with nodes comprised of the start node, intermediate nodes, and end node. The weights on the edges will correspond to the minimum cost path between the nodes

Step 2 – Iterate over all of the intermediate nodes

- a) Determine the minimum weight path from the start node to the intermediate node and add it to the potential fastest route
- b) From the original set of paths found in step 1, remove all paths that have a source of the start node
- c) From the original set of paths found in step 1, remove all paths that have a destination of the intermediate node
- d) As long as the set of paths still contains more paths
 1. Determine the path with the minimum overall cost that still remains
 2. As long as that path does not create a cycle in our potential fastest route, add it to the potential fastest route
 3. From the original set of paths found in step 1, remove all paths that have a source of the source node
 4. From the original set of paths found in step 1, remove all paths that have a destination of the destination node
- e) Order the paths in the potential fastest route from the start node where the destination of one path is the source of the next path
- f) Add the fastest path from the destination of the route back to the start node to the potential fastest route
- g) Store the potential fastest route in an array and reset the potential fastest route to be empty for the next iteration

Step 3 – From all the routes found in Step 2, find the route with minimum weight and return it

FIGURE II. VARIABLE TYPES USED IN ITS-TSP ALGORITHM

Variable Type	Description
NODE	vertex in a GRAPH
EDGE	edge connecting two NODEs in a GRAPH with a weight
GRAPH	set of NODEs and set of EDGEs
NODE_SET	set of NODEs where adjacent NODEs do not have to be connected by an edge
PATH	set of NODEs where adjacent NODEs are connected by an EDGE
PATH_SET	set of PATHs
ROUTE	set of PATHs where the destination NODE of each PATH is the same as the source NODE of the next PATH

The loop on lines 8-17 iterates $n+1$ times, with the nested loop on lines 9-16 iterating n times. The call to the `fastest_path` function on line 14 takes $O(Vm)$ time, where m is the number of paths that are considered in the All Pairs All Paths algorithm [3]. The overall running time of the loop on lines 8-17 is therefore $O(V^3m)$, assuming that in the worst case that $n=V$, meaning that there are V intermediate nodes.

The loop on lines 21-47 iterates n times, the loop on lines 25-35 iterates $n-1$ times (since we will have one path per pair of nodes in our intermediate node set), and the loop on lines 39-42 iterates n times. All of the lines within those loops execute in $O(n)$ time, except getting the minimum path, which will execute in $O(Vm)$ time if the All Pairs All Paths Precomputed – Constant Update (APAP-PC Constant Update) algorithm is used [3]. This gives an overall worst case running time for the loop on lines 21-47 of $O(V^3m)$, assuming in the worst case that $n=V$.

The loop on lines 50-54 iterates n times, and finding the weight of a route takes $O(n)$ time, giving a running time of $O(n^2)$ for that loop.

Adding up those running times, since the loops are in series, gives an overall running time of $O(V^3m)$. As explained in [3], m should be a constant value, giving the worst case running time of the ITS-TSP algorithm as $O(V^3)$.

Although the ITS-TSP is a variation on the traditional TSP, we were able to reduce the running time to polynomial-time. The reason is based on the variations and

the pre-processing step that is required, which takes $O(V^2E!)$ to compute all paths between all pairs of nodes. Since the ITS-TSP algorithm is executed whenever an intermediate node is visited due to constantly changing edge weights, the algorithm may not produce the optimal results if the manner in which the edge weights change were known a priori. The algorithm makes the best decision it can based on the current conditions and the remaining intermediate nodes to visit. Therefore, the ITS-TSP algorithm should be considered a heuristic algorithm rather than an algorithm that will always compute the optimal route based on unknown changing edge weights.

V. DISCUSSION: HEURISTIC OPTIMIZATION FOR PRACTICAL APPLICATIONS OF ITS-TSP

Given that we are solving a practical problem in the ITS domain (as opposed to a pure theoretical problem), there are several optimizations that can be implemented in practice. The graph on which real-time routing is performed can be reduced to a clique consisting of only the nodes that we are interested in visiting, with an edge between every pair of nodes (step 1 in the ITS-TSP algorithm). We will term this sub-graph D . Note that, in practice, it is highly unlikely that a delivery truck, for example, will visit every node in a city. Furthermore, n , the subset of nodes that we need to visit, will likely be significantly less than V ($n \ll V$).

FIGURE III. ITS-TSP PSEUDO-CODE

```

1  -- returns the ROUTE that starts at start_node, ends at end_node, and visits all
2  -- of the nodes in int_nodes, with the minimum overall cost
3  -- NOTE: start_node may not be the same as end_node, so we could be starting at a
4  -- different node than our desired destination, meaning we are in the middle of our route
5  ROUTE ITS-TSP(GRAPH g, NODE_SET int_nodes, NODE start_node, NODE end_node) {
6    int_nodes = int_nodes - {start_node} -- in case int_nodes contains start_node
7    PATH_SET original_path_set = {}
8    for each (NODE src ∈ int_nodes U {start_node}) {
9      for each (NODE dest ∈ int_nodes) {
10       if (src != dest) {
11         -- using All_Pairs_All_Paths algorithm from [3]
12         -- this variable contains all of the fastest paths between all of the
13         -- nodes in start_node and int_nodes
14         original_path_set = original_path_set U fastest_path(g, src, dest)
15       }
16     }
17   }
18   -- find the fastest route from the original_path_set, which contains all of
19   -- the fastest paths between all of the start_node, int_nodes, and end_node
20   PATH_SET path_set = original_path_set
21   for each (NODE u ∈ int_nodes) {
22     ROUTE route = {} U path_set.get_minimum_path(start_node, u)
23     path_set.remove_all_paths_with_src(start_node)
24     path_set.remove_all_paths_with_dest(u)
25     while (path_set != {}) {
26       PATH p = path_set.get_minimum_path()
27       path_set.remove_path(p)
28       -- NOTE: cycles can exist in the overall route traversed over time, but
29       -- not in a route determined based on a snapshot in time
30       if (!route.contains_cycle_with_path(p)) {
31         route = route U p
32         path_set.remove_all_paths_with_src(p.src)
33         path_set.remove_all_paths_with_dest(p.dest)
34       }
35     }
36     -- order the paths where the dest from one path is the src of the next path
37     fin_route[u] = {}
38     PATH curr_path = route.get_path_from_src(start_node)
39     while (curr_path != NULL) {
40       fin_route[u] = fin_route[u] U curr_path
41       curr_path = route.get_path_from_src(curr_path.dest)
42     }
43     -- complete the route with the path from the last node to end_node by
44     -- calling fastest_path function from [3]
45     fin_route[u] = fin_route[u] U fastest_path(g, fin_route[u].dest, end_node)
46     path_set = original_path_set
47   }
48   -- find route with minimum cost from the fin_route array
49   ROUTE minimum_route = {}
50   for each (t ∈ int_nodes) {
51     if (minimum_route == {} OR minimum_route.weight > fin_route[t].weight) {
52       minimum_route = fin_route[t]
53     }
54   }
55   return minimum_route
56 }

```

To translate G to D , every edge in D must map to the minimum cost path in G . Given that the edge weight updates occur in real-time, the cost of all of the pre-computed paths between all pairs of nodes being considered must be recomputed. The step required to create D incurs the cost $O(mn^2)$, which is the cost of finding the minimum cost path among the paths for all node pairs in D . Note that we are assuming that m is precomputed offline. The

performance of this update can vary significantly depending on the value of m (the number of paths between a pair of nodes). In a reasonably well-connected graph, m can be large, adversely affecting the performance of ITS-TSP's online computation. However, in practice, m can be limited to a small constant based on a few realistic assumptions.

Consider an example where, due to congestion, the physically shortest path (15 miles) will take a driver 55

FIGURE IV. DESCRIPTION OF HELPER FUNCTIONS USED IN ITS-TSP ALGORITHM

Function	Description	Running Time
PATH PATH_SET.get_minimum_path(NODE src, NODE dest)	Returns the PATH with minimum weight between the NODE src and NODE dest that is in the PATH_SET	$O(Vm)$ if using the APAP-PC Constant Update algorithm*
PATH PATH_SET.get_minimum_path()	Returns the PATH with minimum weight out of all the PATHs in the PATH_SET	$O(Vm)$ if using the APAP-PC Constant Update algorithm*
void PATH_SET.remove_path(PATH p)	Removes the PATH p from the PATH_SET	$O(n)$ in the worst case if the PATHs are not ordered
void PATH_SET.remove_all_paths_with_src(NODE src)	Removes all of the PATHs from PATH_SET that have NODE src as the source node	$O(n)$ where n is the number of intermediate NODEs that need to be traversed
void PATH_SET.remove_all_paths_with_dest(NODE dest)	Removes all of the paths from PATH_SET that have NODE dest as the destination node	$O(n)$ where n is the number of intermediate NODEs that need to be traversed
boolean ROUTE.contains_cycle_with_path(PATH p)	Returns true if the PATH p completes a cycle with the other PATHs already in the ROUTE	$O(n)$ where n is the number of PATHs currently in ROUTE
PATH ROUTE.get_path_from_src(NODE src)	Returns the PATH from the NODE src that is part of the ROUTE	$O(n)$ where n is the number of PATHs currently in the ROUTE
PATH fastest_path(GRAPH g, NODE src, NODE dest)	Returns the fastest PATH in the GRAPH g from the NODE src to the NODE dest	$O(Vm)$ if using the APAP-PC Constant Update algorithm*
ROUTE.weight	Gets the weight of the ROUTE, which is the sum of the weights of all of the PATHs in the ROUTE	$O(n)$ where n is the number of PATHs in the ROUTE

* V is the number of NODEs in the graph and m is the maximum number of PATHs considered between any pair of NODEs

minutes to traverse, but an alternative route exists (55 miles) with no traffic allowing the driver to get to the same destination in 45 minutes. It is not clear whether the second route is worth the 267% extra distance traveled. Many of the ITS-TSP problems involve visiting destinations that have some notion of locality, where many of the destinations sit inside a certain radius. This fact suggests that a radius-based selection of m -relevant paths is a promising practical optimization [3]. We can then reason that the route required to take a much longer detour is not likely to appear as a solution from the ITS-TSP algorithm since another route by passing some other destination first is likely to be found. (Again, note that these are practical, real-application policy and operation decisions that are used to guide the heuristic rather than an approach based on purely theoretical optimization considering properties of mesh topologies).

The frequency of edge weight updates, which in turn translate to path swapping on D , can also be limited by setting realistic thresholds. First, when edge weights increase on paths that are not the current minimum, the corresponding edge in D need not be changed. Second, by defining a threshold, we can ignore small increases in the weight of an edge currently sitting on D or small decreases on an edge that is not the current minimum cost edge. This approach can be used to limit the online computation time, if necessary, but has not been tested yet. It is important to note, however, that the edge weights in practice are not expected to fluctuate frequently. In addition, with the

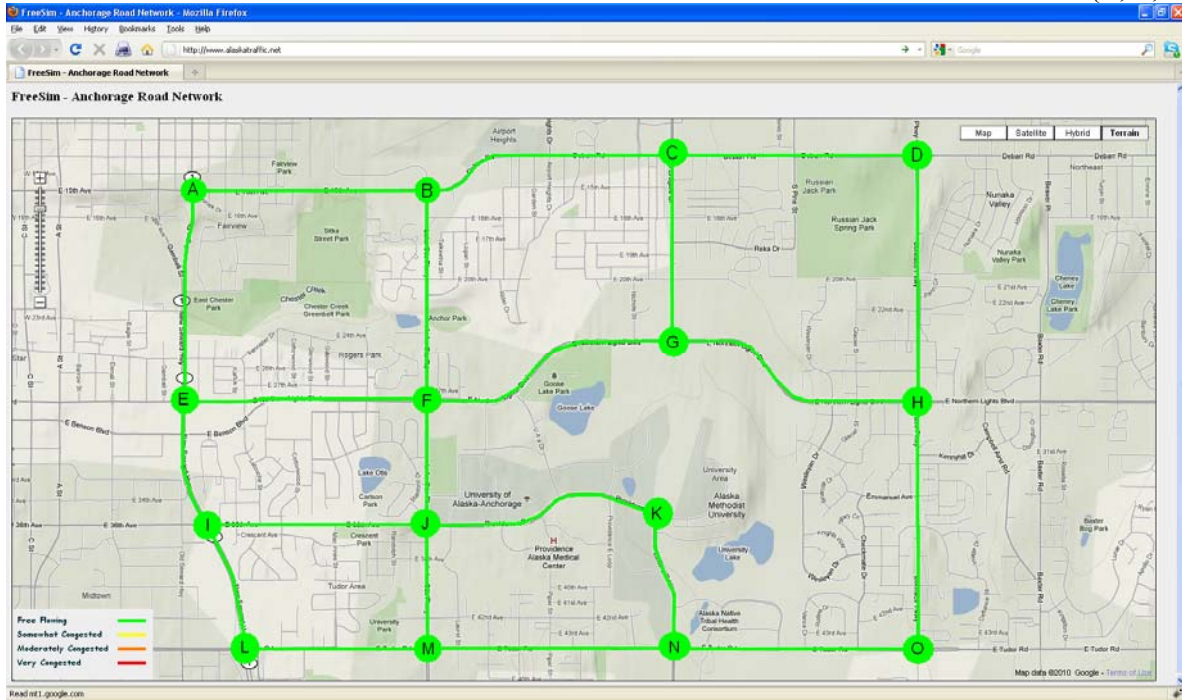
exception of major accidents, the fluctuations that do occur are likely to be small and gradual over time.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented an algorithm for determining the fastest route from a start node through an intermediate set of nodes and returning to the start node. Although this problem resembles the traditional Traveling Salesman Problem, the application of the ITS-TSP is specifically in transportation. The weights of the edges can change constantly, so the route needs to be updated or recomputed during traversal. The set of intermediate nodes that need to be visited does not necessarily include all of the nodes in the graph. And lastly, cycles can exist in the ITS-TSP, for as the edge weights change, it may become faster to travel back through a node that was already visited.

The running time of the ITS-TSP, based on a pre-processing time of $O(V^2E!)$ is $O(V^3)$ in the worst case, if every node in the graph must be visited. Because the weights on the edges are changing constantly, the algorithm is based on recomputing the route based on different snapshots in time. The ITS-TSP algorithm is a heuristic that will compute fastest routes based on those snapshots, but may not compute the optimal route without knowing how the edge weights will change in the future. In Anchorage, 65 vehicles have tracking devices and are reporting their location to a central server. From this data, we have tested the ITS-TSP algorithm based on the FreeSim

FIGURE V. FREESIM EXAMPLE MAP WITH START NODE A AND INTERMEDIATE NODES {B, H, K, P}



screenshot [2] in Figure V, with the start node of A and intermediate nodes of {B, H, K, P}. The ITS-TSP algorithm accurately directs vehicles along fastest routes based on taking snapshots of the edge weights determined by live data during the computation. In the future, we will compare the routes computed from the ITS-TSP with those obtained using the brute force method after knowing how the edge weights change. This will provide us with a means of determining the prediction algorithms to add into the ITS-TSP in the future.

We also have plans of notifying drivers while they are in route as to the paths to traverse to minimize the overall time on the road. This application is of interest to delivery companies as well as to all drivers who travel to more than one destination before returning to their original sources. The ITS-TSP algorithm has many applications in different industries and has the potential to minimize the overall delay experienced by drivers.

VII. REFERENCES

- [1] Jaillet, Patrick. "A Priori Solution of a Traveling Salesman Problem in which a Random Subset of the Customers are Visited." *Operations Research*, Volume 36, Number 6, November 1988.
- [2] Miller, Jeffrey, Ellis Horowitz. "FreeSim - A Free Real-Time Traffic Simulator." *IEEE 10th International Intelligent Transportation Systems Conference*, September 2007.
- [3] Miller, Jeffrey, Ellis Horowitz. "Algorithms for Real-Time Gathering and Analysis of Continuous-Flow Traffic Data." *IEEE 9th International Intelligent Transportation Systems Conference*, September 2006.
- [4] Ghiani, Gianpaolo, Antonella Quaranta, Chefi Triki. "New Policies for the Dynamic Traveling Salesman Problem." *Optimization Methods and Software*, Volume 22, Issue 6, December 2007.
- [5] Lu, Xiangwen, Amelia C. Regan, Sandra Irani. "The Dynamic Traveling Salesman Problem: An Examination of Alternative Heuristics." *Transportation Science*, 2001.
- [6] Larsen, Allan, Oli B.G. Madsen, Marius M. Solomon. "The A-Priori Dynamic Traveling Salesman Problem with Time Windows." *Transportation Science*, Volume 38, Issue 4, November 2004.
- [7] Dijkstra, E.W. "A note on two problems in connexion with graphs." *Numerische Mathematik*, 1959.
- [8] Bellman, Richard. "On a Routing Problem", *Quarterly of Applied Mathematics*, Volume 16, Issue 1, 1958.
- [9] Ford Jr., Lester R., D.R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
- [10] Johnson, Donald. "Efficient Algorithms for Shortest Paths in Sparse Networks." *Journal of the ACM*, 1977.
- [11] Floyd, Robert. "Algorithm 97 (SHORTEST PATH)." *Communications of the ACM*, 1977.
- [12] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms - 2nd Edition*. The MIT Press, 2001.
- [13] Demetrescu, Camil and Giuseppe Italiano. "A New Approach to Dynamic All Pairs Shortest Paths." *ACM Symposium on Theory of Computing*, June 2003.
- [14] Misra, S, B.J. Oommen. "New Algorithms for Maintaining All-Pairs Shortest Paths." *IEEE 10th Symposium on Computers and Communications*, June 2005.
- [15] Miller, Jeffrey, Muhammad Ali. "Dynamic Fastest Paths with Multiple Unique Destinations (DynFast-MUD) - A Specialized Traveling Salesman Problem with Intermediate Cities." *IEEE 12th Intelligent Transportation Systems Conference*, St. Louis, Missouri, USA, October 2009.
- [16] Applegate, David L., Bixby, Robert E., Chavátal, Vasek, Cook, William J. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.
- [17] Lawler, Eugène L., Lenstra, Jan Karel, Kan, A. H. G. Rinnooy, Shmoys, D. B. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons Ltd., 1985.
- [18] Dreyfus, Stuart E., Law, Averill M. Law. *The Art and Theory of Dynamic Programming - Mathematics in Science and Engineering*, Volume 130, Elsevier Science & Technology Books, 1977.