Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

EFFICIENT STORAGE OF TRANSPORT NETWORK ROUTES FOR SIMULATION MODELS

Ramon Alanis

Alberta Health Services
10030 107 Street
Edmonton, AB, T5J 3E4, Canada

ABSTRACT

A heuristic approach is presented which, combined with the use of data structures, reduces storage requirements to store full optimal paths on transportation networks. The goal is to allow the use of full road networks on the implementation of simulations for the operation of a fleet of vehicles. The approach is based on the representation of routing information as a routing matrix, where we exploit structural properties of routing information to reduce storage requirements.

1 INTRODUCTION

A heuristic approach is proposed which, in combination with basic data structures, allows the efficient storage of travel routes from any node to any other node on a road network. Our motivation is the need to implement a simulation model for a fleet of ambulances and the desire to model vehicle travel realistically. This leads to the use of actual road networks and the need to quickly find the best route to follow when an ambulance receives a call.

A simulation model of a fleet of vehicles in a city-sized road network presents challenges derived from the size of the network. We need to model a vehicle fleet, such as ambulances, police cars, fire engines, or taxis, and at any given moment the vehicles can be directed to move to a new destination. To simulate these vehicle moves we need to know the route that the vehicles should follow, and we can potentially have routes from any node to any other node, on a road network that can easily have tens of thousands of nodes. To model this transportation network we must either precompute and store the routes or compute shortest paths on demand using Dijkstra's or a similar shortest path algorithm. Since we may have to find tens or hundreds of thousands of routes for a single simulation run, performance is of a high priority. As example, using a road network with 15,227 nodes, the complexity of finding a route using Dijkstra's algorithm would be of $O(n\log(n)) \approx 63,689$, while extracting a precomputed route will be proportional to the average route length, which assuming a squared road network would be of $O(\sqrt{n}) \approx 123$. We therefore decided that precomputing and storing all the possible routes would produce a fast simulation model, but the price to pay would be high storage requirements. We recognize a significant precomputing cost, but having to run many different simulation scenarios we considered the improvement on performance significant.

Many other applications require the real-time computation or retrieval of optimal routes. With an effective tool, mapping and route-planning web services similar to Google Maps could be easily and efficiently implemented. One example is a local vehicle-dispatch system that manages a fleet of vehicles in a city. The methodology in this paper could potentially simplify the development of route planning services for local areas.

We give a detailed description of the problem in Section 2, where we define the data structures used to store routing information and the properties that allow compression as well as how we can alter the level of compression. In Section 3 we describe how our basic problem can be transformed into a traveling salesman problem (TSP). Section 4 reviews some of the most significant developments in the search for shortest

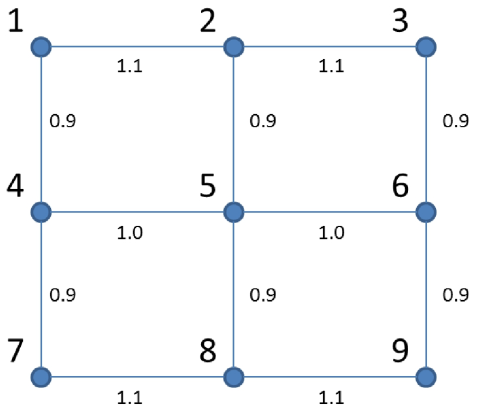
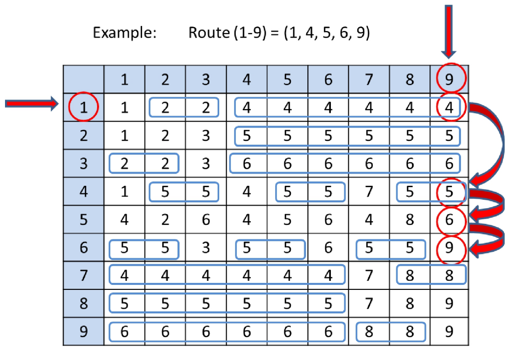


Figure 1: Road Network.



Example: Route (1-9) = (1, 4, 5, 6, 9)

	1	2	3	4	5	6	7	8	9
1	1	2	2	4	4	4	4	4	4
2	1	2	3	5	5	5	5	5	5
3	2	2	3	6	6	6	6	6	6
4	1	5	5	4	5	5	7	5	5
5	4	2	6	4	5	6	4	8	6
6	5	5	3	5	5	6	5	5	9
7	4	4	4	4	4	4	7	8	8
8	5	5	5	5	5	5	7	8	9
9	6	6	6	6	6	6	8	8	9

Figure 2: Routing Matrix.

routes and the solution of the TSP. Section 5 shows how to find optimal solutions for small problems using a well-known TSP formulation or a publicly available TSP solver. Section 6 presents a heuristic approach that allows for efficient storage of information and can be used for large road networks. Finally, Section 7 summarizes the results obtained and presents conclusions.

2 Problem Description

A road network can be defined as a graph $G = (V, A)$ where V is a set of vertices or nodes, with $n = |V|$, and A is a a set of arcs or road segments. There are distances $d_{i,j}$ associated with each arc, where the word distance is used generically, since it may represent the travel cost or travel time. The small nine-node road network shown in Figure 1 will be used to illustrate concepts in the rest of this chapter.

For this road network and any two nodes i and j, the shortest route $p_{i,j} = (p_{i,j,0}, p_{i,j,1}, \ldots, p_{i,j,k})$, where $p_{i,j,0} = i$ and $p_{i,j,k} = j$, is the path for which the sum of the distances along the path is minimized over all possible paths connecting i to j. We assume that the shortest paths have been computed by Dijkstra's or a similar shortest path algorithm.

To represent the shortest paths from any node to any other node we define the matrix $R = \{r_{i,j} = p_{i,j,1}\}$. In this square matrix of dimension $|V| \times |V|$ element $r_{i,j}$ stores the first node after i that would be visited along the shortest path from node i to j. Once at node $l = r_{i,j}$ we proceed to the following node, which is the first node after l on the path from l to j, $r_{l,j}$, and we repeat this until we arrive at $r_{k,j} = j$. For example, in Figure 2 we show how to recover the route from node 1 to node 9 for the network in Figure 1.

Unfortunately, the road network for a medium to large city can easily have tens of thousands of nodes, and storing a route for all possible pairs of nodes would require storage space of the order of $O(n^2)$. For example, the road network for the city of Edmonton, Alberta, Canada, has 15,227 nodes. This leads to a matrix with 231,861,529 elements of data. This pushes the limit on the amount of main memory allocated to a single program on a personal computer.

Fortunately, there is a simple fact that can be exploited: if you stand on a typical street intersection you can travel in only four possible directions. Therefore, for most intersections or nodes, the corresponding row in our routing matrix R has at most four possible entries, not counting the diagonal entries $r_{i,i} = i$. We observe this in Figure 2. In actual road networks, the number of possible directions is not always four, but it is usually small. For four directions, if we alter the order of the columns in R we can obtain four long sequences of repeated values. There are many compression schemes that take advantage of this data replication to store the rows in a more compact way. For example, if there are 10,000 nodes, and the columns are reordered into four sequences of repeated values, they can be stored using four slightly more complex data elements instead of the 10,000 required for the original row.

For example, we could represent our nine-node road network using the routing matrix shown in Figure 3, where each data element (a,b) represents a sequence of value a repeated b times. The worst case occurs for row 5 where the number of data elements is 9; for rows $1,3,7$, and 9 we require only 3 data elements per row.

Based on this, we define c_i to be the storage cost for row i, equal to the number of sequences of identical values found in row i. Unfortunately, if the columns are reordered to reduce the storage cost for row i, there is no guarantee that this ordering will also minimize the storage cost for other rows.

Based on this explanation, our problem is to find the column ordering in our routing matrix that minimizes the sum of the costs c_i for all rows.

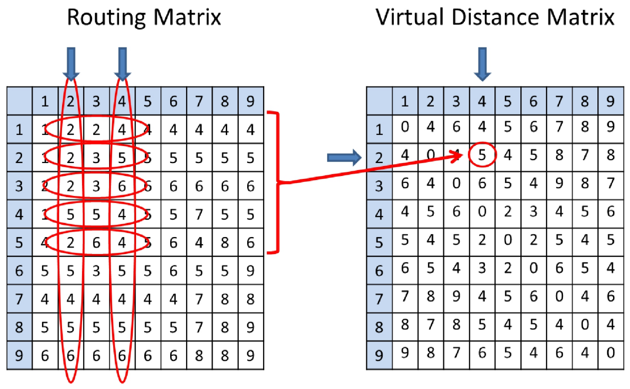
										Cost
1	(1,1)	(2,2)	(4,6)							3
2	(1,1)	(2,1)	(3,1)	(5,6)						4
3	(2,2)	(3,1)	(6,6)							3
4	(1,1)	(5,2)	(4,1)	(5,2)	(7,1)	(5,2)				6
5	(4,1)	(2,1)	(6,1)	(4,1)	(5,1)	(6,1)	(4,1)	(8,1)	(6,1)	9
6	(5,2)	(3,1)	(5,3)	(6,1)	(5,2)	(9,1)				6
7	(4,6)	(7,1)	(8,2)							3
8	(5,6)	(7,1)	(8,1)	(9,1)						4
9	(6,6)	(8,2)	(9,1)							3

Figure 3: Compressed Routing Matrix.

3 Problem Transformation

Since the problem is to find the ordering of the columns that minimizes the total cost, we need a way to compute the cost of having two columns i and j next to each other. By analyzing a single row k we see that if $r_{k,i} = r_{k,j}$ then the existing sequence continues, and the additional cost of having $r_{k,i}$ next to $r_{k,j}$ is 0. On the other hand, if $r_{k,i} \neq r_{k,j}$ then the existing sequence ends at $r_{k,i}$ and a new sequence starts at $r_{k,j}$, which implies the creation of a new data element and therefore an additional storage cost of 1.

To extend the previous analysis to a pair of columns, we define $d^v_{i,j}$ to be the cost of having column i next to column j, such that it is equal to the number of rows for which the two columns have different values. This is also known as the Hamming distance (Hamming 1950) between the columns, and we will refer to this as the "virtual distance" between columns i and j to distinguish it from physical road-network distances. We represent this by a virtual distance matrix D^v as illustrated in Figure 4.



	1	2	3	4	5	6	7	8	9
1	1	2	2	4	4	4	4	4	4
2	1	2	3	5	5	5	5	5	5
3	2	2	3	6	6	6	6	6	6
4	1	5	5	4	5	5	7	5	5
5	4	2	6	4	5	6	4	8	6
6	5	5	3	5	5	6	5	5	9
7	4	4	4	4	4	4	7	8	8
8	5	5	5	5	5	5	7	8	9
9	6	6	6	6	6	6	8	8	9

	1	2	3	4	5	6	7	8	9
1	0	4	6	4	5	6	7	8	9
2	4	0	4	5	4	5	8	7	8
3	6	4	0	6	5	4	9	8	7
4	4	5	6	0	2	3	4	5	6
5	5	4	5	2	0	2	5	4	5
6	6	5	4	3	2	0	6	5	4
7	7	8	9	4	5	6	0	4	6
8	8	7	8	5	4	5	4	0	4
9	9	8	7	6	5	4	6	4	0

Figure 4: Virtual Distance Matrix.

The problem can be transformed by changing our perspective, from a geographical space to a "virtual space" in which the columns in R can be seen as "virtual nodes," with the distances between each pair of virtual nodes defined by $D^v = \{d^v_{i,j}\}$.

The problem is now to find a route or path visiting all the virtual nodes, without any restriction on which node is first or last, such that all the nodes are visited exactly once and the total virtual distance is minimized.

Finding a route visiting all nodes exactly once is also known as the Hamiltonian path problem. Similarly, a Hamiltonian cycle problem involves finding a path visiting each node exactly once, except that the route must be closed into a cycle. If we start from a Hamiltonian path problem there is a simple transformation that converts it into a Hamiltonian cycle problem: we add a dummy virtual node and connect it to every node in our problem, with all the new arcs having the same very small distance, smaller than that of any existing arc. This ensures that a minimum-distance Hamiltonian cycle is also a minimum-distance Hamiltonian path. The solution to the cycle problem can be transformed into a path by eliminating the dummy node and its associated arcs.

Our problem is a minimum-distance Hamiltonian path problem and it is equivalent (via the transformation mentioned above) to a minimum-distance Hamiltonian cycle problem, or in other words, the well-known traveling salesman problem (TSP), one of the most studied problems in graph theory.

4 Literature Review

The current literature is reviewed with a focus on two areas: algorithms to find optimal or minimum-cost paths between two nodes and algorithms for the TSP.

4.1 Shortest Path algorithms

Even when for our purpose we assume that shortest paths are a pre-computed input, the basic problem is to provide somehow shortest paths between any pair of nodes, and the solutions, as we will see, go from computation of shortest paths on demand, to different levels of pre-computing and storage of partial routing information, with our approach being at the extreme by storing all possible paths.

Finding an optimal or minimum-cost route between two nodes in a graph is a problem whose solution is widely applied today; however, researchers continue to develop improved algorithms. The classical solution was proposed by Dijkstra (1959). The original algorithm stores tentative distances from the source node to each node in the network. We start at the source node and gradually construct a minimum-distance spanning tree from the source to each node by adding, one by one, directly connected nodes to the tree and revisiting the minimum distance to the source, until we finally include the destination in our minimum spanning tree. The execution time for Dijkstra's original algorithm is of the order $O(n^2)$. However, a continental road map for the US or Western Europe has millions of nodes, so this original algorithm is probably not being used in your typical GPS device.

The improvements on Dijkstra's algorithm can be classified into exact algorithms (those that return an optimal solution) and approximate algorithms, such as those used in some GPS units with limited hardware. We will focus on exact algorithms.

The first improvement involves performing a bidirectional search. Instead of starting at the source and gradually constructing a shortest-distance tree, the search starts simultaneously at the source and at the destination. In the original Dijkstra's algorithm the search forms an approximately circular shape covering the nodes around the source, until the destination is included. In a bidirectional search, two circles start to grow, one around the source and the other around the destination, until some node is visited from both directions (Dantzig 1962). The number of nodes visited is reduced and a speed-up of approximately a factor of two is observed.

Another improvement can be achieved by implementing Dijkstra's algorithm using $O(n)$ priority queues, as described by Thorup (2004). With this method we can achieve execution times of $O(n\log(n))$ without significantly altering the idea behind the original algorithm.

Another approach is based on the intuitive idea of giving a higher priority to arcs that lead toward the destination, rather than searching equally in all directions. This is the A* algorithm as proposed by Hart, Nilsson, and Raphael (1968). In this algorithm we estimate a lower bound on the distance from every node to the destination, such as the Euclidean distance. Based on this we modify the weight of an arc (u,v) to $w(u,v)-\pi(u)+\pi(v)$ where $\pi(v)$ is the lower bound on the distance to the destination. With this adjustment we shrink arcs leading toward the destination while preserving the selection of the optimal path.

Another step on the search for efficiency comes from the use of hierarchical approaches and the removal or selection of arcs and nodes to simplify the network. Examples include the work of Jing, Huang, and Rundensteiner (1998) and Cagigas (2005). Jing, Huang, and Rundensteiner (1998) propose a HiTi graph model where a high graph is divided into smaller subgraphs, leaving at the higher level only those nodes on the boundary of the subgraphs. This can be extended to multiple levels. At each subgraph we compute the shortest paths between all possible pairs of boundary nodes, and we replace all the internal nodes by arcs directly connecting the boundary nodes.

A different approach is reach-based routing (Gutman 2004), based on the concept of "reach." A reach metric is computed for each node based on the smaller of the distance from that node to the source and the distance to the destination on a critical path. These values can be used to eliminate nodes from consideration in a shortest path algorithm when their reach is too short to reach either the source or the destination. This approach requires significant precomputation.

Another hierarchical approach, highway hierarchies (Sanders and Schultes 2005), creates a hierarchy of highway levels. It uses the concept of neighborhood to classify the nodes on any optimal path into the neighborhood of the source, the neighborhood of the destination, and the other nodes (called highway nodes). Based on this classification a highway graph is constructed and then simplified by contracting and removing edges to create a higher-level graph in a hierarchical network. This process can be repeated to create a multilevel hierarchy.

Transit node routing is a hierarchical approach based on the observation that for long routes there are important (transit) nodes, such as major intersections that are visited by a large number of shortest-path routes. If we identify a set of transit nodes in a graph, most route searches consist of finding a route to a nearby transit node, then finding a route on a higher-level network of transit nodes to a transit node near the destination, and finally finding a local route on to the destination.

These are just a few examples of recent hierarchical approaches, and some algorithms combine several of the techniques described. Most hierarchical approaches have in common the creation of a hierarchical structure to represent the road networks or graphs, with a distinction between local neighborhoods and higher levels of simplified road networks. We must consider not only the complexity of finding a shortest-path route, but also the complexity of performing precomputation and the space required to store the precomputed results that allow for a faster search.

Our approach has a narrower scope than some of the techniques mentioned because we need a simple approach that can be easily incorporated into a simulation, and some of the techniques above are too complex to be easily implemented. Nevertheless, our efficient storage of precomputed shortest paths could be useful in some of the hierarchical methods.

4.2 The Traveling Salesman Problem

The TSP is one of the most widely studied problems in graph theory and combinatorial optimization. Consider a graph $G=(V,A)$ where V is a set of vertices and A is a set of arcs connecting some of those vertices, with costs or distances defined by a matrix $D=(d_{i,j})$. The TSP is defined as the problem of finding a minimum-distance circuit that visits each vertex once and only once. Two common scenarios include the case where D is symmetric ($d_{i,j}=d_{j,i}$), as in our column-ordering problem, and the case where

D satisfies the triangle inequality, i.e., $d_{i,j} + d_{j,k} \geq d_{i,k} \forall i, j, k \in V$. The triangle inequality is a property of graphs on a plane with Euclidean distances, but it does not hold for our column-ordering problem. This is a significant drawback since we cannot use the methods that rely on the assumption of Euclidean distances and the triangle inequality.

The TSP is NP-hard (Laporte 1992), which suggests that finding an optimal solution might be intractable for our target problem size. One of the earliest approaches was an integer linear programming formulation by Dantzig, Fulkerson, and Johnson (1954). This original formulation became the foundation for many subsequent models such as those proposed by Miller, Tucker, and Zemlin (1960) and Desrochers and Laporte (1991). Branch-and-bound approaches have been proposed by Carpaneto and Toth (1980) and Miller and Pekny (1991) to solve large TSPs. Some of these methods, in particular the one proposed by Miller and Pekny (1991), have been able to solve instances with thousands of nodes. However, sometimes the same methods are unable to solve even relatively small problems.

As already mentioned, this is by no means an exhaustive survey of all the methods available. In our application exact solutions are not easily found for the large problems that we want to analyze, and heuristic methods based on tour improvements are slow for our problems. We therefore consider tour-construction methods, which have the potential to be faster at the cost of suboptimal solutions.

5 Exact Solutions

As a first step we propose the use of the well-known TSP integer programming formulation proposed by Dantzig, Fulkerson, and Johnson (1959). Unfortunately, it is limited to small problems, with typically tens to hundreds of nodes, much smaller than the problems that we want to solve.

It is worth mentioning the Concorde TSP Solver, a solver freely available for academic use (http://www.tsp.gatech.edu/concorde.html), which uses some of the most advanced exact and heuristic methods to search for optimal solutions. It is considered the state-of-the-art solver for the TSP problem. Benchmark results are available from the same website for a set of problems, most of which are characterized as being neither easy nor hard. From these it is clear that current TSP algorithms cannot be used in real-life applications to consistently solve problems beyond 10,000 nodes. Therefore, heuristic or approximate algorithms are the only practical approach to our problem.

6 Approximate Solutions

There are many heuristic algorithms for the TSP; Laporte (1992) in his review mentions many of them. Before selecting one, we analyze some properties of our problem:

1. We have introduced virtual nodes and presented the problem as a TSP, but in reality we do not have geographically located nodes in a two-dimensional space. Therefore, we must be careful because geometric assumptions, such as Euclidean distances, usually associated with TSP problems can be violated for our virtual distances.
2. A typical road network is sparse, with a low ratio of arcs to nodes. In our virtual space we have a fully connected network, where each virtual node (column) has a direct connection to every other node. This, together with the previous observation, could have an unexpected impact on performance if we attempt to use spanning trees or other heuristics that traverse all possible paths or that take advantage of the geometrical properties of planar networks.
3. If we have two trips that start from the same source but travel to different destinations that are far from the source but close to each other, then there is a high probability that the two routes will initially be identical. As an extreme example consider traveling to two different addresses in a distant town. You are likely to follow the same route for most of the trip, except at the end. Thus, in our routing matrix it would be convenient to have neighboring nodes in adjacent columns.

4. If we could find a Hamiltonian path joining all the original geographical nodes we could satisfy our goal of having neighboring column nodes that correspond to actual geographical neighboring nodes.

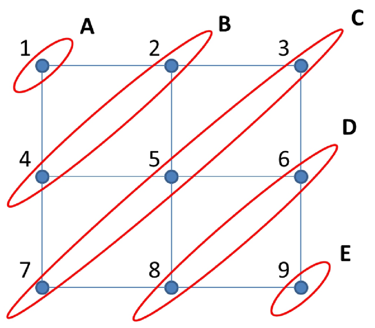


Figure 5: Ordering or Layers on Processing of Nodes.

Based on the previous observations we decided to implement a heuristic approach loosely based on the family of tour-construction insertion algorithms but taking into account the properties above. The detailed heuristic is as follows:

1. Choose arbitrarily a single geographical node as our starting solution. In Figure 5 we could select node 1 at stage A.
2. Define the set of geographical nodes to be added at the next iteration by choosing all the nodes not already in the solution but directly connected to nodes in the solution. We refer to this as the next stage or the next layer to add. In Figure 5 the stages are A, B, C, D, and E.
3. Take a geographical node in the new layer but not already added. For demonstration purposes we proceed from left to right.
4. Search in the current solution for the geographical nodes directly connected to the node under consideration and consider adding the corresponding virtual node or column in a position adjacent to any of the directly connected geographical nodes. We insert the new virtual node in such a way that the total cost after the insertion is minimized. If we add node k between nodes i and j, then the cost of the insertion is $(d^v_{i,k} + d^v_{k,j} - d^v_{i,j})$.
5. Repeat the last two steps until all the nodes in the new layer have been added.
6. If there are still nodes that are not in the solution, return to step 2.

Table 1 illustrates the sequence of steps required to solve our nine-node example.

With the use of efficient data structures this algorithm is of order $O(n)$ based on the number of operations, since we insert the nodes one by one, and for each node we analyze at most four possible insertion points.

The solution found and displayed in Figure 7 is, in this particular case, a Hamiltonian path, which is consistent with the intuition indicating that directly connected nodes should have low “virtual distances.” It therefore leads to the intuition that in an optimal solution adjacent columns are likely to correspond to geographical nodes with a direct connection, resulting in the formation of geographic paths. In other examples we have observed optimal solutions composed of several interconnected geographic paths, which shows that the optimal solution is not always a Hamiltonian path in the geographical network.

We used the same data to formulate a TSP. We added the dummy node 10 to our virtual distance matrix as shown in Figure 8, and since the shortest node-to-node distance is 2, we assigned a distance of 1 from node 10 to every other node. We solved the problem using the NEOS Concorde Online Solver (http://www.neos-server.org), and we obtained the solution shown in Figure 6, which matches our heuristic solution.

Table 1: Heuristic Solution Process.

Layer	Node to Add	Alternative Paths	Cost of Addition	Selected
A	1	(1)	$n = 9$	(1)
B	4	(4-1)	$d^v_{4,1} = 4$	(4-1)
		(1-4)	$d^v_{1,4} = 4$	
	2	(4-2-1)	$d^v_{4,2} + d^v_{2,1} - d^v_{4,1} = 5$	
		(4-1-2)	$d^v_{1,2} = 4$	(4-1-2)
C	7	(7-4-1-2)	$d^v_{7,4} = 4$	(7-4-1-2)
		(4-7-1-2)	$d^v_{4,7} + d^v_{7,1} - d^v_{4,1} = 7$	
	5	(7-5-4-1-2)	$d^v_{7,5} + d^v_{5,4} - d^v_{7,4} = 3$	(7-5-4-1-2)
		(7-4-5-1-2)	$d^v_{4,5} + d^v_{5,1} - d^v_{4,1} = 3$	
		(7-4-1-5-2)	$d^v_{1,5} + d^v_{5,2} - d^v_{1,2} = 5$	
		(7-4-1-2-5)	$d^v_{2,5} = 4$	
	3	(7-5-4-1-3-2)	$d^v_{1,3} + d^v_{3,2} - d^v_{1,2} = 6$	
		(7-5-4-1-2-3)	$d^v_{2,3} = 4$	(7-5-4-1-2-3)
D	8	(8-7-5-4-1-2-3)	$d^v_{8,7} = 4$	
		(7-8-5-4-1-2-3)	$d^v_{7,8} + d^v_{8,5} - d^v_{7,5} = 3$	(7-8-5-4-1-2-3)
		(7-5-8-4-1-2-3)	$d^v_{5,8} + d^v_{8,4} - d^v_{5,4} = 7$	
	6	(7-8-6-5-4-1-2-3)	$d^v_{8,6} + d^v_{6,5} - d^v_{8,5} = 3$	(7-8-6-5-4-1-2-3)
		(7-8-5-6-4-1-2-3)	$d^v_{5,6} + d^v_{6,4} - d^v_{5,4} = 3$	
		(7-8-5-4-1-2-6-3)	$d^v_{2,6} + d^v_{6,3} - d^v_{2,3} = 5$	
		(7-8-5-4-1-2-3-6)	$d^v_{3,6} = 4$	
E	9	(7-9-8-6-5-4-1-2-3)	$d^v_{7,9} + d^v_{9,8} - d^v_{7,8} = 6$	
		(7-8-9-6-5-4-1-2-3)	$d^v_{8,9} + d^v_{9,6} - d^v_{8,6} = 3$	(7-8-9-6-5-4-1-2-3)
		(7-8-6-9-5-4-1-2-3)	$d^v_{6,9} + d^v_{9,5} - d^v_{6,5} = 7$	

From	To	Cost
1	2	4
2	3	4
3	10	1
10	7	1
7	8	4
8	9	4
9	6	4
6	5	2
5	4	2
4	1	4

Figure 6: Solution.

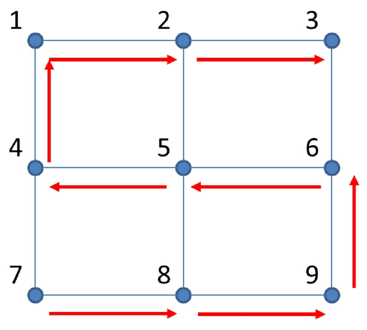
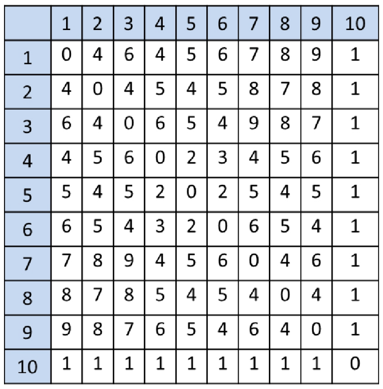


Figure 7: Graphic Solution.



	1	2	3	4	5	6	7	8	9	10
1	0	4	6	4	5	6	7	8	9	1
2	4	0	4	5	4	5	8	7	8	1
3	6	4	0	6	5	4	9	8	7	1
4	4	5	6	0	2	3	4	5	6	1
5	5	4	5	2	0	2	5	4	5	1
6	6	5	4	3	2	0	6	5	4	1
7	7	8	9	4	5	6	0	4	6	1
8	8	7	8	5	4	5	4	0	4	1
9	9	8	7	6	5	4	6	4	0	1
10	1	1	1	1	1	1	1	1	1	0

Figure 8: Modified Virtual Distances for TSP.

Because the solution represents a circuit, and node 10 is a dummy, we can remove node 10 and reconnect the start and the end to form a single path: $(7-8-9-6-5-4-1-2-3)$. The total storage cost can be computed via

- The number of rows, 9, for the initial cost of the first sequence in each row.
- Plus the cost of the TSP solution, 30, for each time that a sequence was broken and a new data element was required.
- Minus the cost of connecting the dummy node, $(1+1)=2$.

Therefore, the total number of data elements is 37, compared to the initial cost of 81 elements in the original uncompressed routing matrix. The compression rate usually becomes more significant as the number of nodes grows, as we will see in Section 7.

7 Results and Conclusions

We created two versions of a 100-node road network, a 9 block by 9 block perfect grid (10×10 nodes). In one version we created an artificially regular routing pattern, such that to travel between any two nodes the rule is to first travel horizontally and then vertically. In the second version we allowed randomness in the length of each block, so that each arc had a base length of 10 plus or minus 1, with an equal probability of $+1$ or -1. This resulted in optimal routes that were less predictable. In both cases the solution was obtained by our heuristic in 0.11 s, while the Concorde NEOS Server took 0.13 s for the regular case and 1.39 s for the random case. Both methods found the optimal cost of 460 data elements for the regular case. For the random case the heuristic method found a cost of 1,058 elements while Concorde found an optimal cost of 905 elements. Given the original size of the routing matrix (10,000 elements) both solutions provide excellent results: the optimal TSP-based solution requires 9.05% of the original storage requirement while the heuristic approach requires 10.58%. From this experiment we learned that:

1. The level of compression depends on other characteristics of the road network in addition to its size.
2. The heuristic solution, as expected, can be faster and provide good results.
3. The gains in compression seem to increase with the size of the road network.

The results for different networks are summarized in Table 2.

We used eight different road networks to test our models. The networks had 9, 12, 100, 1,936, 2,025, 4,900, and 15,227 nodes. We display 3 of the networks in Figure 9. All the networks are artificial, except for the last one, which corresponds to the road network for the city of Edmonton, Alberta, Canada.

Table 2: Summary of Performance Results.

Nodes	Original Size	Heuristic			Concorde/IP		
		Cost	Time (s)	Ratio	Cost	Time (s)	Ratio
9	81	37	0.02	45.68%	37	< 0.01	45.68%
12	144	52	0.02	36.11%	52	< 0.01	36.11%
100 Reg	10,000	460	0.11	4.6%	460	0.13	4.6%
100 Rnd	10,000	1,058	0.11	10.58%	905	1.39	10.37%
1,936 Reg	3,748,096	9,504	2.09	0.25%	9,504	4.23	0.25%
2,025 Reg	4,100,625	9,945	2.57	0.24%			
4,900 Reg	24,010,000	24,220	18.01	0.10%			
15,227 Real	231,861,529	574,993	187.68	0.25%			

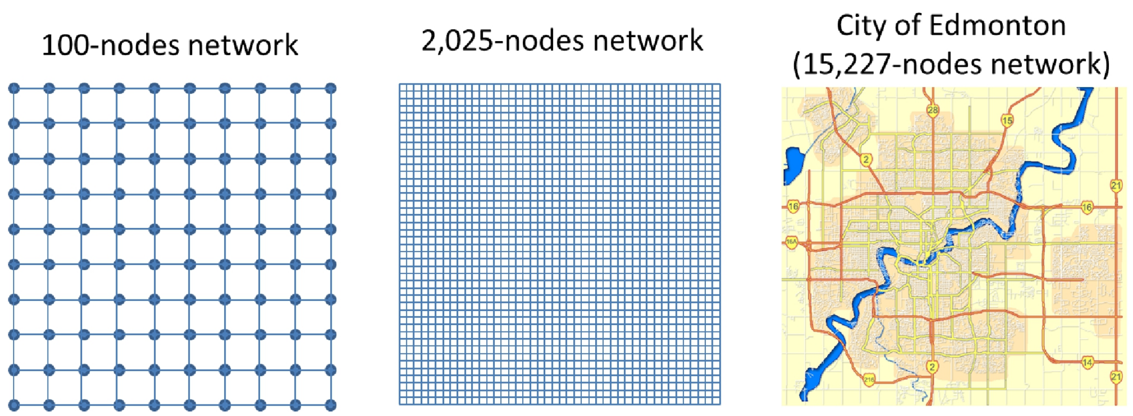


Figure 9: Road Networks.

The first solution method attempted was the Dantzig, Fulkerson, and Johnson (1959) IP formulation already discussed, and it was tested on the models with 9 and 12 nodes. It obtained an optimal solution, but it is suitable only for very small networks. On larger networks (22 nodes) it failed to find an integer solution in 2.5 h. It was deemed inadequate for real road networks. The use of Concorde allows the solution of larger problems. A version of Concorde is available for personal computers, but it is limited to Euclidean distances, which makes it unsuitable for our application. Another version is available online at the NEOS Server (http://www.neos-server.org/neos/), and it supports symmetric distance matrices. This service has a limit on the size of uploaded files that limited our tests to an approximate maximum of 1,936 nodes. This is roughly equivalent to a grid of 43 blocks by 43 blocks. Given the time to solve the problem (4.23 s) we expect that the Concorde solver could deal with larger problems.

We tested our insertion method on the networks with 9, 12, 100, 1,936, 2,025, 4,900, and 15,227 nodes. For 9, 12, 100, and 1,936 nodes we found the optimal solution. For larger problems we found solutions with high compression ratios but we do not know if they are optimal. It is worth mentioning that our computer-generated networks are highly regular. On tests with 100 nodes and randomness in the length of the arcs, and therefore with less predictable routing, our heuristic achieved significant compression but was not able to find an optimal solution. The storage size required was 17% higher than that of the optimal solution, and the compression rate achieved was within 0.21 percentage points of that achieved by the optimal solution.

Our experiments show that the heuristic can handle realistically sized networks in a relatively short time. The level of compression achieved in such networks is impressive: we observed a reduction from 231,861,529 data elements to just 574,864 for the network with 15,227 nodes. This represents a compression of 99.75% and makes storing the full routing information entirely feasible and not demanding at all for a city the size of Edmonton. We used these data in a simulation of the Edmonton EMS system, and the amount of data was easily handled. We achieved fast performance for both the data preparation and the simulation itself.

The complexity of the storage requirements is not easy to estimate, mostly because, as seen in the two 100-node examples, there is a wide variation depending on the characteristics of the road network. We approximate the complexity by plotting the number of nodes against the observed number of data elements required to store the compressed routing matrix. The plot is shown in Figure 10; both axes are transformed under a log_{10} transformation, and the plot displays a power trend line that corresponds to the equation $y = 3.1949n^{1.1205}$.

This storage formula implies that for a continental-size network of 4,000,000 nodes we could require $79,811,503$ data elements. When translated into data structures this indicates that every route in a whole continent could easily fit into a small flash drive or a small portion of the memory available on a smartphone.

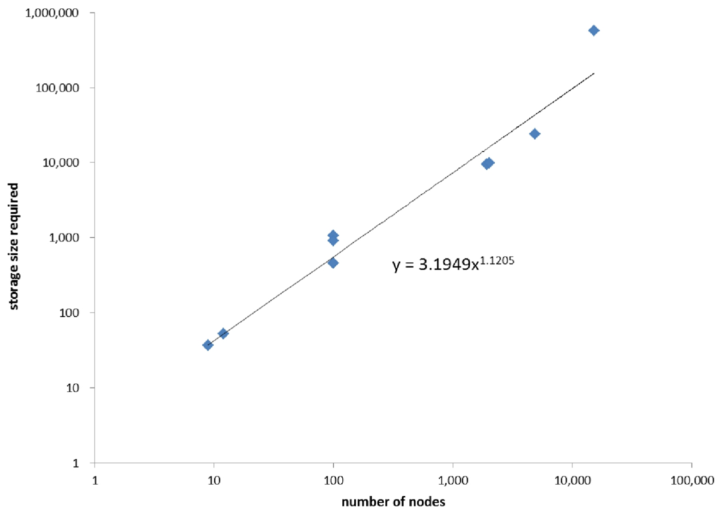


Figure 10: Storage Requirements vs Network Size.

To conclude:

- We have developed a simple heuristic that allows a very significant reduction in the storage requirements for routing information for a road network.
- The complexity of the heuristic is linear in the number of operations.
- The level of compression achieved increases with the size of the road network. Empirically, we estimate that it is of order $O(n^{1.1205})$.

REFERENCES

Cagigas, D. 2005, 8/31. “Hierarchical D* algorithm with materialization of costs for robot path planning”. *Robotics and Autonomous Systems,* 52 (2–3): 190–208.

Carpaneto, G., and P. Toth. 1980. “Some new branching and bounding criteria for the asymmetric travelling salesman problem”. *Management Science* 26 (7): 736–743.

Dantzig, G. B. 1962. *Linear Programming and Extensions*. Princeton: Princeton University Press.

Dantzig, G. B., D. R. Fulkerson, and S. M. Johnson. 1959. “On a linear-programming, combinatorial approach to the traveling-salesman problem”. *Operations Research* 7 (1): 58–66.

Dantzig, G. B., R. Fulkerson, and S. Johnson. 1954. “Solution of a large-scale traveling-salesman problem”. *Journal of the Operations Research Society of America* 2 (4): 393–410.

Desrochers, M., and G. Laporte. 1991. “Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints”. *Operations Research Letters* 10 (1): 27–36.

Dijkstra, E. W. 1959. “A note on two problems in connexion with graphs”. *Numerische Mathematik* 1:269–271.

Gutman, R. J. 2004. “Reach-based routing: A new approach to shortest path algorithms optimized for road networks”. In *ALENEX/ANALC’04*, 100–111.

Hamming, R. W. 1950. “Error detection and error correction codes”. *Bell System Technical Journal* 29 (2): 147–160. MR0035935.

Hart, P. E., N. J. Nilsson, and B. Raphael. 1968, july. “A formal basis for the heuristic determination of minimum cost paths”. *IEEE Transactions on Systems Science and Cybernetics* 4 (2): 100–107.

Jing, N., Y. W. Huang, and E. A. Rundensteiner. 1998, MAY-JUN. “Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation”. *IEEE Transactions on Knowledge and Data Engineering* 10 (3): 409–432.

Laporte, G. 1992, June. “The Traveling Salesman Problem - An overview of exact and approximate algorithms”. *European Journal of Operational Research* 59 (2): 231–247.

Miller, C. E., A. W. Tucker, and R. A. Zemlin. 1960, October. “Integer programming formulation of traveling salesman problems”. *Journal of the ACM* 7:326–329.

Miller, D. L., and J. F. Pekny. 1991. “Exact solution of large asymmetric traveling salesman problems”. *Science* 251 (4995): 754–761.

Sanders, P., and D. Schultes. 2005. “Highway hierarchies hasten exact shortest path queries”. In *ESA’05*, 568–579.

Thorup, M. 2004. “Integer priority queues with decrease key in constant time and the single source shortest paths problem”. *Journal of Computer and System Sciences* 69 (3): 330–353. Special Issue on STOC 2003.

AUTHOR BIOGRAPHIES

RAMON ALANIS is Senior Operations Researcher at Alberta Health Services in Edmonton, AB, Canada and Instructor at the School of Business in Grant MacEwan University. He holds a Ph.D. in Operations and Information Systems from the University of Alberta and a M.S. in Management Science from Queen’s University. His email address is ramon.alanis@AlbertaHealthServices.ca.