Heuristics
for
Sparse General Travelling Salesman Problems

L. R. Foulds
Department of Management Systems
University of Waikato
Hamilton, New Zealand

Derek R. O'Connor
Department of Management Information Systems
University College
Dublin, Ireland

Abstract : The well-known travelling salesman problem can be expressed in graph-theoretic terms as follows : find a minimum-weight Hamiltonian cycle in a connected edge-weighted graph $G(N, A)$. We concentrate here on the problem where G is large and sparse, *i.e.*, where $|N|$ is 500 or more and $|A|$ is $O(|N|)$. Such graphs arise frequently in practice, where G represents a road network, and each node (city) is connected directly to 3 or 4 other nodes. Sparse graphs may not have Hamiltonian cycles and hence there may be no solution to the problem as stated. Whether this is the case or not, the requirement that the cycle is Hamiltonian is often unnecessarily restrictive. It is sufficient in many practical problems, such as traffic planning, to address only what is called the General Travelling Salesman Problem, which requires merely a *minimum-weight, closed spanning walk*, in which any node can be repeated if this lowers the total weight. Such a walk can, of course, be found in any connected graph. We report on the results of an experimental comparison of some heuristic methods for this problem.

1. Introduction

Consider a travelling salesman who, starting from his home city, must visit, exactly once, each city on a given list and then return home. It is assumed that a cost (usually given terms of distance or time) of travelling between each pair of cities is known. The problem is to find the least-cost tour. An early seminal paper on the problem was presented by Dantzig, Fulkerson, and Johnson [3] and the problem was popularized by Flood [5]. A comprehensive treatise on research into it has been compiled by Lawler, Lenstra, Rinnooy-Kan, and Shmoys [12], reflecting the fact that the problem has become one of the most famous in combinatorial mathematics, operations research, and computer science. It has many direct and indirect applications, some that have nothing to do with physical movement on a graph. Some examples of these have been given by Garfinkel [6] : vehicle

Australasian Journal of Combinatorics 5(1992), pp.285-292

routing, computer wiring, wallpaper cutting, job sequencing, and clustering a data array. Our present work is motivated by the need to find closed tours in graphs representing general road graphs. Typically, these graphs represent road networks of rural areas with up to 10,000 towns, with relatively few pairs of towns *directly* connected by a road. The paper reports on the results of an experimental comparison of solution methods that were developed for such practical problems.

We begin in the next section by introducing the necessary notation and terminology. We then proceed to state the relevant combinatorial optimization problems to be studied. We then devise some solution methods for the problem and end with a comparison of the performance of these methods.

2. Definitions

Consider a connected, undirected, arc-weighted graph $G = (N, A)$, with *node* set N and with an *arc* set A, an element of which is denoted by (u, v) if it directly connects nodes u and v. Each arc (u, v) has a non-negative real-valued weight denoted by $d(u, v)$. Let $|N| = n$, and $|A| = m$. For details on the undefined concepts arising in this paper the reader is referred to Harary [8], and Lawler, *et al.* [12]. Naturally $m \leq n(n-1)/2$, with equality occuring when G is complete. A graph is said to be *dense* if m is $O(n^2)$ and *sparse* if m is $O(n)$.

A Hamiltonian Cycle of G is a simple cycle in G in which every node of G appears exactly once. If a graph has a Hamiltonian cycle it is called a Hamiltonian graph.

The *Travelling Salesman Problem* (TSP) is : find in G a Hamiltonian cycle of least weight.

The weight of a cycle is the sum of the weights of the arcs of the cycle. The correspondence between the TSP and the problem described earlier is apparent when the cities, roads, and travel costs are represented by the nodes, arcs, and arc-weights, respectively, of a graph. When G is dense it is very likely to be Hamiltonian and thus it is likely to have at least one feasible solution to the corresponding TSP. However, if G is sparse it is far less likely to be Hamiltonian and thus possess a feasible solution for the TSP.

Many of the graphs representing practical problems are sparse. Such graphs may not have any Hamiltonian cycles, and even if they do they may be difficult to find. To complicate matters further, a minimum weight Hamiltonian cycle may not be optimal for the underlying practical problem. This is because a minimum weight Hamiltonian cycle is constrained to visit each node exactly once, whereas an optimal tour may visit nodes more than once and achieve a shorter tour. A simple example that illustrates this is the complete graph of 3 nodes u,v,w and arc lengths $d(u, v) = d(u, w) = 1$, and $d(v, w) = 3$. The minimum weight Hamiltonian cycle (u, v, w, u) has length 5, and the optimum tour (u, v, u, w, u) has length 4. For this reason we study the following problem defined below :

Definition (*The General Travelling Salesman Problem* (GTSP)) : Given an edge-weighted undirected graph G, find a closed spanning walk of least weight in G

We call a closed spanning walk a *tour* and a solution to the GTSP an *optimal tour*. Our purpose here is to devise effective solution procedures for the GTSP where G is relatively large (n = 500 to 10000) and sparse. The GTSP is *NP-Hard* and so we concentrate on the development of simple heuristic algorithms that are fast and, we hope, give tours that are not too far above the optimum.

3. Review of Literature

Most, if not all work on practical algorithms for the TSP has been confined to relatively small ($n < 500$) complete graphs. Golden, et al. [7] devotes just one paragraph to sparsity, in which TSP heuristic methods are tested on one 100-node problem with the density of G varying from 100% down to 20%. Most of these algorithms could not find a solution when the density of G is less than 40%.

Some heuristic methods have been devised for relatively large Euclidean (straight-line) TSP's. Karp's Dissection-Patching methods [9], Simulated Annealing [11], Litke [13], Bland & Shallcross [2], Padberg & Rinaldi [15], have given promising results. The testing (and design) of these methods have been on complete graphs. That is, the problem is given as n coordinates in the Euclidean plane and it is assumed (implicitly) that the salesman can move from any coordinate to any other coordinate (in a straight line). This means that large problems can be handled without storing the very large distance matrix that underlies the problem.

Miller & Peckney [14] have reported the results of experiments on random asymmetric problems with up to 500,000 nodes and random symmetric problems with up to 20,000 nodes. Both classes of problem have been solved in up to 13,000 secs on a Cray-2.

Bentley [1] described data structures and algorithms for million-node problems which find solutions within a few percent of optimal in a few hours of midi computer time.

As Garfinkel [6], among others, has pointed out, any numerical instance of a GTSP can be transformed into a TSP by replacing each arc weight $d(u, v)$ by the length of the shortest path from u to v.

4. Solution Techniques for the GTSP

The methods mentioned above work well in certain problem contexts, e.g., minimizing plotter pen travel, but they are impracticable in other contexts, e.g., actual road graphs. The road graphs that motivated the present research are large, sparse and do not obey the triangle inequality. They have n (nodes) in the range [500-10,000] and m (arcs) in the range [1,000-30,000]. Thus their arc densities are typically less than 1%. The methods we propose to use are based on some standard heuristic methods, modified to handle large sparse graphs. Because of the large size of these graphs we consider only those heuristic methods that have low-order complexity. Hence we do not consider methods whose complexity is $O(n^3)$ or greater.

4.1 Data Structures

Large sparse graphs cannot be represented and stored as adjacency matrices because a typical graph matrix would have 1M to 25M elements. Hence, we use an adjacency list representation of the graph which requires $O(n+m)$ storage. In this representation only arcs that are actually present in the graph are stored. This means that any method that implicitly assumes the existence of all arcs must be modified to check explicitly for the existence of each arc that is currently being considered by the method.

4.2 Algorithms

The weight of the minimum spanning tree (MST), which we denote by $L(MST)$, is a lower bound on the minimum tour weight, while $2L(MST)$ is an upper bound. Prim's MST algorithm [16] for finding a minimum spanning tree is easily implemented for large sparse graphs and can be used to obtain crude lower and upper bounds on the optimal solution to the GTSP. The Assignment Problem lower bound is not used because it gives very poor lower bounds for undirected graphs.

We now discuss modifications of two standard heuristic algorithms for the TSP that run in $O(n^2)$ time. These algorithms are chosen because modifications enabling them to solve the GTSP seem natural and are easy to implement.

The *Nearest Neighbour Heuristic* is heuristic for the TSP which has the following form, where $Tour$ is an ordered set of nodes representing the partly-constructed tour :

Algorithm Nearest-Neighbour

Pick a starting node s and mark it *visited*
$Tour := \{s\}$
$u := s$
while there exists an *unvisited* node **do begin**
    $v :=$ unvisited v that minimizes $d(u,v)$
    $Tour := Tour + \{v\}$
    Mark v *visited*
    $u := v$
end while
$Tour := Tour + \{s\}$, i.e., return to start.

This algorithm assumes that G is complete and thus it is always possible to travel directly between any pair of nodes u,v via arc (u,v). In sparse graphs the arc (u,v) may not exist and so we must modify the algorithm so that it proceeds from u to the nearest unvisited node that minimizes $Dist(u,v)$, where $Dist(u,v)$ is the shortest path distance from u to v.

To find the shortest path distance $Dist(u,v)$, we use a slight modification of Dijkstra's algorithm [4] for finding the shortest path between nodes u and v in G. Dijkstra's algorithm requires a start node s and, optionally, a target node t. Once the target node has been labelled *permanent* the algorithm can be terminated because the shortest path from s to

t, and $Dist(s,t)$, have been calculated. In our modification we start Dijkstra's algorithm at u without a target node t and terminate it after the *first unvisited* node v has been labelled permanent. Thus we get a path which we denote by $P(u,v)$ from u to the nearest unvisited node v. The heuristic is as follows :

Algorithm Nearest-Neighbour-Shortest-Path

```
Pick a starting node s and mark it visited
Tour := {s}
u := s
while there exists an unvisited node do begin
    P(u,v) := Path from u to unvisited v
              that minimizes Dist(u,v)
    Tour := Tour + P(u,v)
    Mark v visited
    u := v
end while
Tour := Tour + P(u,s), i.e., return to start by shortest path.
```

The *Minimum Spannning Tree Heuristic* is a TSP heuristic that starts by constructing an MST of G. Then, starting at some node s, it traverses the tree in a *Depth-First* manner. When it reaches a leaf node u, instead of backing up the tree it moves directly to the next node v in depth-first order that has not been visited. When the last node of the tree has been visited the tour is completed by returning directly to the start node s. This algorithm assumes that the graph is complete, that the triangle inequality holds, and hence the direct moves are possible and shortest.

In sparse graphs direct moves are not always possible and so we must modify this heuristic as follows : when a leaf node u is reached move *on the shortest path* to the next node v in depth-first order that has not yet been visited. The heuristic is as follows :

Algorithm Minimum-Spanning-Tree-Shortest-Path

```
Construct a Minimum Spanning Tree MST.
Construct a Depth-First Ordering of the nodes of MST
starting at node s.  Call this ordered set DFO.
Tour := {s}; DFO := DFO - {s}
u := s
while DFO is not empty do begin
    v := next node on DFO
    DFO := DFO - {v}
    if arc (u,v) ∈ MST then
        Tour := Tour + {v}
    else
        P(u,v) := Path from u to v that minimizes Dist(u,v)
        Tour := Tour + P(u,v)
    u := v
end while
Tour := Tour + P(u,s), i.e., return to start by shortest path.
```

In general, the tours generated by the heuristics above depend on the start node. To guarantee the best tour for each heuristic, all nodes of the graph must be used as start nodes. This can be very expensive for large graphs. Alternatively a small, randomly-chosen, subset of the nodes can be used as start nodes, and the best of these tours chosen.

Examples

The following graph of 5 nodes and 6 arcs is taken from Kelly [10]. The arc list $\{(u, v, d(uv)\}$ is

$$\{(1,4,15),(1,5,90),(2,3,100),(2,4,25),(2,5,60),(3,4,100),(3,5,50),(4,5,80)\}.$$

The only Hamiltonian tours of this graph are $\{1,5,2,3,4,1\}$ and $\{1,5,3,2,4,1\}$, which have lengths 365 and 280, respectively. The *non*-Hamiltonian tour $\{1,4,3,5,2,4,1\}$ has length 265, and this is the optimum tour. Both the *Nearest Neighbour Shortest Path* and the *Minimum Spanning Tree Shortest Path* algorithms generate the following tours, using the start nodes 2 and 5 : $\{2,4,1,5,3,2\}$ with length 280, and $\{5,2,4,1,5,3,5\}$ with length 265.

4.3 Analysis of the Heuristics

The **while**-loop of each heuristic is performed n times. The main work performed in each loop is the application of Dijkstra's algorithm to find the shortest path between two nodes u and v. Dijkstra's algorithm takes $O(m + n \log n)$ time, if properly implemented [17]. For sparse graphs with $m = O(n)$ this reduces to $O(n \log n)$ time. Constructing a minimum spanning tree requires $O(n \log n)$ time, and Depth-First Search requires $O(n)$ time. Hence the complexity of each algorithm is $O(n^2 \log n)$.

5. Testing and Results

We tested the heuristics on two sets of sparse road networks :

Type	Name	n/m
Small	Ireland1	67/116
	Ireland1	100/179
Large	Sri-Lanka	981/1327
	Ireland3	1797/2809
	Urban 1	1830/3139
	Rural 1	4923/5679

In all cases we constructed a minimum spanning tree to get crude lower and upper bounds, *viz.*, $L(MST)$ and $2L(MST)$.

To obtain the best possible tour from each heuristic it is necessary to start the tour at each of the n nodes of G. This was done for the small graphs. For the large graphs 20 nodes chosen at random were used as the starting point and the best of the 20 tours generated was chosen. This limitation was necessary because of the size of the graphs and the computer used. The compiler-machine details are as follows : 12MHz AT using Turbo Pascal 5.0 with range-checking turned OFF.

Tour Lengths and Computation Times (12MHz AT secs)

	Ireland1 67/116	Ireland2 100/170	SriLanka 981/1327	Ireland3 1797/2809	Urban1 1830/3139	Rural1 4923/5679
MST	1338	1572	70939	55460	5760	14805
SP N-N	1912	2197	115721	91050	9589	29557
	(0.2)	(0.2)	(3.5)	(8.5)	(10.0)	(47.0)
MST-SP	2135	2477	123970	98320	10286	28940
	(0.2)	(0.3)	(5.5)	(14.0)	(16.0)	(102.0)
2 MST	2676	3144	141878	110920	11520	29610

[Note: The times shown in parentheses are for the generation of 1 tour]

It can be seen in all cases except Rural1 (which has an underlying graph that is extremely sparse), that the Nearest Neighbour heuristic gives better results than the MST heuristic.

6. Summary

We have posed heuristics for the general travelling salesman problem, where its underlying graph is relatively large and sparse. Such graphs require carefully-designed data structures and heuristics to support the implementation of solution techniques if they are to solve realistically-sized problems in reasonable computational time. Further, the sparsity may require repeated visits to certain cities to obtain an optimum tour. We have devised methods for actual road networks which gave rise to this research. They are modifications of existing methods for the TSP which assume that the underlying graph is complete. It has been shown that the methods are capable of solving, in a reasonable amount of time, large-scale practical instances of problems for which they were designed.

References

[1] Bentley, J.L : 'Experiments on Geometric Travelling Salesman Heuristics', *AT&T Computing Science Technical Report*, No. 151, 1990.

[2] Bland, R.G., and Shallcross, P.F. : 'Large TSP's Arising from Experiments in X-Ray Crystallography : A Preliminary Report on Computation', *OR Letters*, Vol 8, 125–128, 1989

[3] Dantzig, G.B., Fulkerson, D.R., and Johnson S.M. : 'Solution of a Large-Scale Traveling-Salesman Problem'. *Operations Research*, Vol 2, 393–410, 1954.

[4] Dijkstra, E.J. : 'A Note on Two Problems in Connexion with Graphs', *Numerische Mathematik*, Vol 1, 269–271, 1959.

[5] Flood, M.M. : 'The Traveling Salesman Problem'. *Operations Research*, Vol 4, 61–75, 1956.

[6] Garfinkel, R.S. : 'Motivation and Modeling'. in [Lawler, *et al.*].

[7] Golden, B., Bodin, L., Doyle, T., and Stewart, W. : 'Approximate Traveling Salesman Algorithms', *Operations Research*, Vol 28, No.3. Part II, 694–711, 1980.

[8] Harary, F. : *Graph Theory*, Addison-Wesley, 1969.

[9] Karp, R.M., and Steele, J.M. : 'Probabilistic Analysis of Heuristics', in [Lawler, *et al.*].

[10] Kelly, C. : 'Monte Carlo Methods for the Sparse Travelling Salesman Problem', Unpublished M.Mangt.Sc. Dissertation, MIS Dept., University College, Dublin, Ireland, 1988.

[11] Laarhoven, P.J.M., and Aarts, E.H.L. : *Simulated Annealing : Theory and Applications*, Reidel, 1987.

[12] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.S. : *The Traveling Salesman Problem*, Wiley, 1985.

[13] Litke, J.D. : 'An Improved Solution to the TSP with Thousands of Nodes', *Computing Practices*, Vol 27, No 12, 1227–1236, 1984.

[14] Miller, D., and Pekney, J. : 'The Travelling Salesman Problem', *Science*, Vol 251, 754–761, 1991.

[15] Padberg, M., and Rinaldi, G. : 'Facet Identification for the Symmetric TSP', *Mathematical Programming*, Vol 47, 219–257, 1990.

[16] Prim, R.C. : 'Shortest Connection Networks and Some Generalizations', *Bell System Technical Journal*, Vol 36, 1389–1401, 1957.

[17] Tarjan, R.E. : *Data Structures and Network Algorithms*, CBMS 44, SIAM, 1983.