1.203J, 6.281J, 15.073J, 16.76J, etc.
Logistical and Transportation Planning Methods

Some Important Heuristics for the TSP

We summarize below some of the principal characteristics of a number of the best-known heuristic algorithms for the TSP. The worst-case results cited apply to TSPs which have symmetrical distance matrices that satisfy the triangular inequality, but some of the heuristics can also be used in problems that do not satisfy these conditions. Additional details can be found in the following three outstanding references:

Jünger, Michael, G. Reinelt and G. Rinaldi, "The Traveling Salesman Problem", Chapter 3 in *Network Models* (*Volume 7* of *Handbooks in Operations Research and Management Science* series) M.O, Ball, T.L. Magnanti, C.L. Monma and G. Nemhauser (eds.) Elsevier Science, Amsterdam (1995) pp. 225 -330.

Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (editors) *The Traveling Salesman Problem*, John Wiley and Sons, Chichester (1985).

Rosenkrantz, D.J., R.E. Stearns and P.M. Lewis, "An Analysis of Several Heuristics fot the Traveling Salesman Problem", *SIAM Journal of Computing, 6* (1977) pp. 563-581.

A. Construction Heuristics

1. Nearest Neighbor Heuristic:

Begin at a random node or at a pre-specified node and build a tour by always moving to the nearest neighbor (among the nodes not yet connected to the tour) of the latest node to join the tour. When all nodes have been connected to the tour, join the last node to the origin to complete the tour.

Worst-case performance:

$$\frac{L(\text{NEARNEIGHBOR})}{L(\text{TSP})} \leq \frac{1}{2}\lceil \log_2 n \rceil + \frac{1}{2}$$

Computational Complexity: $O(n^2)$.

2. Insertion Heuristics:

All the heuristics in this class, initiate the tour-construction process with a "seed tour" consisting either of a single node with a self-loop or of a loop involving just two nodes. (The method of selecting these two initial nodes is itself a matter of choice.)

It is useful to define the cost, c(i, k, j) of inserting a new node, k, in a tour between two nodes, i and j, which were previously adjacent in the tour:

$c(i, k, j) = d(i, k) + d(k, j) - d(i, j)$.

We then have:

2a. Random Insertion Heuristic: The next node to join the tour, T, is selected randomly among the nodes still in (N - T) where N is the set of nodes of the network. The location where the selected node is inserted is the one that minimizes c(i, k, j). The procedure is repeated until all nodes have been inserted into T.

2b. Nearest Insertion Heuristic: The next node to join the tour, T, is the one that minimizes d(k, T) among all the nodes $k \varepsilon (N-T)$. (We use the notation d(k, T) for the distance between a node k and the node in the set T which is closest to k.) The location where the selected node is inserted is the one that minimizes c(i, k, j). The procedure is repeated until all nodes have been inserted into T.

2c. Farthest Insertion Heuristic: The next node to join the tour, T, is the one that maximizes d(k, T) among all the nodes $k \varepsilon (N-T)$. The location where the selected node is inserted is the one that minimizes c(i, k, j). The procedure is repeated until all nodes have been inserted into T.

2d. Cheapest Insertion Heuristic: The next node to join the tour, T, is the one that minimizes c(i, k, j) among all the nodes $k \varepsilon (N-T)$ and for all consecutive pairs of nodes $(i, j) \varepsilon T$. The location where the selected node is inserted is, of course, the one that minimizes c(i, k, j). The procedure is repeated until all nodes have been inserted into T.

Worst-case performance:

2a: $\frac{L(RANDOMINSERT)}{L(TSP)} < \log_2 n + 1$

2b: $\frac{L(NEARINSERT)}{L(TSP)} < 2$

2c: $\frac{L(FARINSERT)}{L(TSP)}$: unknown

2d: $\frac{L(CHEAPINSERT)}{L(TSP)} < 2$

Computational complexity:

Heuristics 2a, 2b, and 2c are all $O(n^2)$; heuristic 2d is $O(n^3)$ --but with careful programming it can be $O(n^2 \log n)$.

3. Minimum Spanning Tree (MST) Heuristic:

(i) Find the MST of the n nodes to be visited.
(ii) Double all the edges of the MST to obtain an Eulerian graph; the Euler tour of that graph is a solution to TSP.
(iii) Improve the tour obtained in (ii) by not re-visiting any node that has already been visited.

Worst-case performance:

$$\frac{L(MST\text{-}TOUR)}{L(TSP)} < 2$$

Computational complexity: $O(n^2)$.

4. Christofides Heuristic:

Please see section 6.4.6 in Larson and Odoni (pp. 398-404)

Worst-case performance:

$$\frac{L(CHRISTOFIDES)}{L(TSP)} < \frac{3}{2}$$

Computational complexity: $O(n^3)$.

5. Convex Hull Heuristic:

Note: This is a heuristic for Euclidean TSPs only.

(i) Find the convex hull of the set of n nodes to be visited (please also see section 6.4.7 in Larson and Odoni). The convex hull provides the starting point for the construction of the tour.
(ii) Use any insertion heuristic to add the rest of the nodes, one at a time, to the tour.

Worst-case performance: Unknown.

Computational complexity: $O(n^3)$ if a cheapest insertion heuristic is used to add points to initial (convex hull) tour; $O(n^2)$ if a nearest neighbor or farthest neighbor or insertion heuristic is used to add points to initial (convex hull) tour

6. Savings Algorithm

Please see discussion of Clarke-Wright Savings Algorithm (Algorithm 6.7) in section 6.4.12 in Larson and Odoni (pp. 417-422). This discussion refers to the more general application of the Savings Algorithm to the Vehicle Routing Problem (VRP). The TSP can, of course, be viewed as a special case of the VRP in which any capacity restrictions have been removed. Thus, to apply the Savings Algorithm to the TSP one proceeds until all nodes (i.e., points to be visited) have been merged into a single tour.

Worst-case performance: Unknown.

Computational complexity: $O(n^3)$.

B. Tour Improvement Heuristics

All tour improvement heuristics must begin with a complete TSP tour T, which is used by the heuristics as the initial solution.

1. Improvement through Node Insertion

Take a node i, currently adjacent to nodes p and q in T. Attempt to improve tour by placing i between two other nodes k and j which are currently adjacent in T. Repeat for all pairs (k,j) until an improvement is found. Repeat for all nodes in the tour until an improvement is found.

Computational complexity: $O(n^2)$.

2. 2-exchange (or 2-opt) Heuristic

Perform the following until failure for every node i occurs:

(i) Select a node i in the current T.

(ii) Examine all possible "2-opt" moves involving the edge between i and its successor node in T. (Make sure to maintain a single connected tour.) If it is

possible to decrease the tour length this way, then choose the best such move and call the resulting tour T. Otherwise, declare failure for node i.

Computational complexity: $O(n^2)$.

3. 3-exchange (or 3-opt) Heuristic

Same idea as 2-exchange heuristic but examines all possible "3-opt" moves.

Computational complexity: $O(n^3)$.

4. Variable Depth Search

This is also known as the "Lin-Kernighan Search Heuristic". It employs a combination of "improvement by node insertion", "2-opt" and "3-opt" moves. Occasionally may accept a tour modification that increases slightly the length of the tour. Designed to make many changes initially and few later on. Many variations of the heuristic exist.

Computational complexity: Depends on choices made with regard to depth of search.