2.3. TSP—HEURISTIC APPROACHES

Heuristics examined. The heuristics we examine fall into three broad classes—tour construction procedures, tour improvement procedures, and composite procedures. *Tour construction procedures* generate an approximately optimal tour from the distance matrix. *Tour improvement procedures* attempt to find a better tour given an initial tour. *Composite procedures* construct a starting tour from one of the tour construction procedures and then attempt to find a better tour using one or more of the tour improvement procedures. Most of these procedures are described in the literature and, hence, will be sketched only briefly; but newer procedures will be studied in more detail. We assume for the sake of simplicity that the costs are symmetric, (i.e. $c_{ij} = c_{ji}$) satisfy the triangle inequality, and are defined for each (i, j) pair, unless otherwise specified.

2.3.1 *Tour construction procedures*

(*a*) *Nearest neighbor procedure* (Rosenkrantz, Stearns, and Lewis[568]).

Step 1. Start with any node as the beginning of a path.

Step 2. Find the node closest to the last node added to the path. Add this node to the path.

Step 3. Repeat step 2 until all nodes are contained in the path. Then, join the first and last nodes.

Worst case behavior:

$$\frac{\text{length of nearest neighbor tour}}{\text{length of optimal tour}} \leq \frac{1}{2}\left\lceil \lg(n) \right\rceil + \frac{1}{2}$$

where lg denotes the logarithm to the base 2, $\lceil X \rceil$ is the smallest integer $\geq X$, and n is the number of nodes in the network.

Number of computations. The nearest neighbor algorithm requires on the order of n^2 computations.

Comments. In a computational setting, the procedure outlined above may be repeated n times, each time with a new node selected as the starting node. The best solution obtained would then be listed as the answer. Notice that this strategy runs in an amount of time proportional to n^3.

(b) *Clark and Wright Savings* (Clark and Wright[145], Golden [291]).

Procedure

Step 1. Select any node as the central depot which we denote as node 1.

Step 2. Compute savings $s_{ij} = c_{1i} + c_{1j} - c_{ij}$ for $i, j = 2,3, \ldots, n$.

Step 3. Order the savings from largest to smallest.

Step 4. Starting at the top of the savings list and moving downwards, form larger subtours by linking appropriate nodes i and j. Repeat until a tour is formed.

Worst case behavior. The worst case behavior for this approach is known for both a sequential and concurrent version. Golden[289] demonstrates that for a sequential version of this algorithm where at each step we select the best savings from the last node added to the subtour, the worst case ratio is bounded by a linear function in $\lg(n)$. Ong[517] has derived a similar result for the concurrent version.

Number of computations. The calculation of the matrix $S = [s_{ij}]$ in step 2 requires about cn^2 operations for some constant c. Next, in step 3, savings can be sorted into nonincreasing order via the "Heapsort" method of Williams[674] and Floyd[226] in a maximum of $cn^2 \lg(n)$ comparisons and displacements. Step 4 involves at most n^2 operations since there are that many savings to consider. Thus, the Clark and Wright savings procedure requires on the order of $n^2\lg(n)$ computations.

Comments. Each node may be selected as node 1 for the above procedure (yielding $n^3\lg(n)$ computations). The best solution obtained would be listed as the answer. In practice one can exploit geometric properties when distances are Euclidean; as a result, only a fraction of the almost n^2 saving are calculated which reduces running times by an order of magnitude (see Golden, Magnanti and Nguyen[306] for details).

(c) *Insertion procedures* (Rosenkrantz, Sterns and Lewis [568]).

An insertion procedure takes a subtour on k nodes at iteration k and attempts to determine which node (not already in the subtour) should join the subtour next (the selection step) and then determines where in the subtour it should be inserted (the insertion step). For the first four insertion procedures we discuss, each node in the network can be used as a starting node. Notice that when each node is used as a starting node, the complexity of the entire procedure increases by an order of magnitude (that is, the number of computations is multiplied by n).

(c1) *Nearest insertion*

Procedure

Step 1. Start with a subgraph consisting of node i only.

Step 2. Find node k such that c_{ik} is minimal and form the subtour i-k-i.

Step 3. *Selection step.* Given a subtour, find node k not in the subtour closest to any node in the subtour.

Step 4. *Insertion step.* Find the arc (i, j) in the subtour which minimizes $c_{ik} + c_{kj} - c_{ij}$. Insert k between i and j.

Step 5. Go to step 3 unless we have a Hamiltonian cycle.

Worst case behavior. $\dfrac{\text{length of nearest insertion tour}}{\text{length of optimal tour}} \leq 2.$

Number of Computations. The nearest insertion algorithm requires on the order of n^2 computations.

(c2) *Cheapest insertion*

Procedure

Same as for nearest insertion except that steps 3 and 4 are replaced by the following step.

Step 3'. Find (i, j) in subtour and k not, such that $c_{ik} + c_{kj} - c_{ij}$ is minimal and, then, insert k between i and j.

Worst case behavior: Same as for nearest insertion.

Number of computations. The cheapest insertion algorithm requires on the order of $n^2\lg(n)$ computations.

(c3) *Arbitrary Insertion*

Procedure

Same as for nearest insertion except that in step 3, arbitrarily select node k not in the subtour to enter the subtour.

Worst case behavior.

$$\frac{\text{length of arbitrary insertion tour}}{\text{length of optimal tour}} \leq \lceil \lg(n) \rceil + 1.$$

Number of computations. The arbitrary insertion algorithm requires on the order of n^2 computations.

(c4) *Farthest Insertion*

Procedure

Same as for nearest insertion except that in step 3 replace "closest to" by "farthest from" and in step 2 replace "minimal" by "maximal".

Worst case behavior. Same as for arbitrary insertion.

Comments. Since the worst case result holds for any arbitrary ordering of the nodes, it holds in particular for the ordering induced by the farthest insertion procedure. It is, however, quite possible that a tighter worst case bound can be derived.

(c5) *Quick Insertion or Nearest Addition*

Procedure

Step 1. Pick any node as a starting circuit T, with one node (and 0 edges).

Step 2. Given the k-node circuit T_k, find the node z_k not on T_k that is closest to a node, call it y_k, on T_k.

Step 3. Let T_{k+1} be the $k+1$—node circuit obtained by inserting z_k immediately in front of y_k in T_k.

Step 4. Repeat Steps 2 and 3 until a Hamiltonian circuit (containing all nodes) is formed.

Worst case behavior. Same as for nearest insertion. *Number of Computations*: Same as for nearest insertion.

(c6) *Convex hull insertion*

It has been shown that if the costs c_{ij} represent Euclidean distance and H is the convex hull of the nodes in two-dimensional space, then the order in which the nodes on the boundary of H appear in the optimal tour will follow the order in which they appear in H (see Eilon, Watson–Gandy and Christofides[201] for a discussion of the implications of this result). This observation serves as impetus for the convex hull heuristic procedure (which is discussed along with a host of variants in Stewart's doctoral dissertation[627]). See also the work of Golden and Stewart[308].

Procedure

Step 1. Form the convex hull of the set of nodes. The hull gives an initial subtour.

Step 2. For each node k not yet contained in the subtour decide between which two nodes i and j on the subtour to insert node k. That is, for each such k, find (i, j) such that $c_{ik} + c_{kj} - c_{ij}$ is minimal.

Step 3. From all (i, k, j) found in Step 2, determine the (i^*, k^*, j^*) such that $(c_{i^*k^*} + c_{k^*j^*})/c_{i^*j^*}$ is minimal.

Step 4. Insert node k^* in subtour between nodes i^* and j^*.

Step 5. Repeat Steps 2 through 4 until a Hamiltonian cycle is obtained.

Worst case behavior. Unknown.

Number of computations. This procedure involves, in the worst case, about as much computational complexity as the cheapest insertion algorithm—on the order of $n^2 \lg(n)$ computations.

Comments. Wiorkowski and McElvain[685] introduced an insertion algorithm also based on convexity ideas a number of years ago. However, their procedure performed with rather poor accuracy. More recently, Norback and Love[514] have proposed two geometric methods closely related to algorithm (c6).

(c7) *Greatest Angle Insertion*

Procedure

Step 1. From the convex hull of the set of nodes. The hull gives an initial subtour.

Step 2. Choose the node k^* not in the subtour and the arc (i^*, j^*) in the subtour such that the angle formed by the two arcs (i^*, k^*) and (k^*, j^*) is the largest possible.

Step 3. Insert node k^* in subtour between nodes i^* and j^*.

Step 4. Repeat Steps 2 and 3 until a Hamiltonian cycle is obtained.

Worst case Behavior. Unknown. However, Yang[696] has demonstrated the negative result that there does not exist a constant that bounds the ratio of "greatest angle" tour length to optimal tour length.

Number of computations. Same as for cheapest insertion. *Comments*: Norback and Love[514] seem to have first suggested this approach as well as a related procedure known as the eccentric ellipse method.

(c8) *Difference × ratio insertion* (*Or* [518])

Procedure

Same as for greatest angle insertion except that Step 2 is replaced by the following step.

Step 2'. Choose the node k^* not in the subtour and the arc (i^*, j^*) in the subtour such that the product $\{c_{i^*k^*} + c_{k^*j^*} - c_{i^*j^*}\} \times \{(c_{i^*k^*} + c_{k^*j^*})/c_{i^*j^*}\}$ is smallest possible.

Worst case behavior. Unknown

Number of computations. Same as for cheapest insertion.

(d) *Minimal spanning tree approach* (Kim [387])

Procedure.

Step 1. Find a minimal spanning tree T of G.

Step 2. Double the edges in the minimal spanning tree (MST) to obtain an Euler cycle.

Step 3. Remove polygons over the nodes with degree greater than 2 and transform the Euler cycle into a Hamiltonian cycle.

Worst case behavior.

$$\frac{\text{length of MST approach tour}}{\text{length of optimal tour}} \leq 2.$$

Number of computations. This approach requires on the order of n^2 computations.

(e) *Christofides' heuristic* Christofides[132] recently proposed the following interesting technique for solving TSP's.

Procedure

Step 1. Find a minimal spanning tree T of G

Step 2. Identify all the odd degree nodes in T. Solve a minimum cost perfect matching on the odd degree nodes using the original cost matrix. Add the branches from the matching solution to the branches already in T, obtaining an Euler cycle. In this subgraph, every node is of even degree although some nodes may have degree greater than 2.

Step 3. Remove polygons over the nodes with degree greater than 2 and transform the Euler cycle into a Hamiltonian cycle.

Worst case behavior.

$$\frac{\text{length of Christofides' tour}}{\text{length of optimal tour}} \leq 1.5.$$

Cornuejols and Nemhauser[151] have improved this bound slightly (although not asymptotically) in obtaining a tight bound for every $n \geq 3$.

Number of computations. Since the most time-consuming component of this procedure is the minimum matching segment which requires $O(n^3)$ operations, this heuristic is $O(n^3)$. In most cases, the number of odd nodes will be considerably less than n.

(f) *Nearest merger* (Rosenkrantz *et al.* [568])

The nearest merger method when applied to a TSP on n nodes constructs a sequence $S_1, \ldots, S_n$ such that each S_i is a set of $n - i + 1$ disjoint subtours covering all the nodes.

Procedure

Step 1. S_1 consists of n subtours, each containing a single node.

Step 2. For each $i < n$, find an edge (a_i, b_i) such that $c_{a_i b_i} = \min \{c_{xy}$ for x and y in different subtours in $S_i\}$. Then, S_{i+1} is obtained from S_i by merging the subtours containing a_i and b_i. (At each step in the procedure, the two subtours closest to one another are merged.)

Worst case behavior.

$$\frac{\text{length of nearest tour}}{\text{length of optimal tour}} \leq 2.$$

Number of computations. This approach requires on the order of n^2 computations.

Comments. To merge two tours T_1 and T_2 when one or both consist of a single node is trivial. If T_1 and T_2 each contain at least two modes, let (a, b) be an edge in T_1 and (d, e) an edge in T_2 such that

$$c_{ad} + c_{be} - c_{ab} - c_{de}$$

is minimized. Then MERGE (T_1, T_2) is the tour derived from T_1 and T_2 by deleting (a, b) and (d, e) and adding (a, d) and (b, e).

Other tour construction algorithms have been proposed recently by Stinson[634] and Karp[376]. Stinson's heuristic is a modification of Vogel's approximation method which has been used extensively in obtaining an initial feasible solution to the transportation problem.

Karp[376] presents a partitioning algorithm for the TSP in the plane and performs a probabilistic analysis in order to obtain some interesting theoretical results. From a practical point of view, his procedure is a decomposition algorithm which should be capable of solving extremely large TSP's. The key idea is to partition a large rectangle into a number of subrectangles and solve a TSP in each subrectangle. The outcome is an Euler cycle which can be transformed into a tour over all the nodes with minimal effort. A simpler, but somewhat related approach, is described by Platzman and Bartholdi[546] they include a BASIC program listing in the appendix.

2.3.2 *Tour improvement procedures*

Perhaps the best known heuristics for the TSP are branch exchange heuristics. The 2-opt and 3-opt heuristics were introduced by Lin[444] in 1965 and the k-opt procedure, for $k \geq 3$, was presented by Lin and Kernighan[446] more recently. These branch exchange heuristics work as follows:

Step 1. Find an initial tour. Generally, this tour is chosen randomly (this need not, however, be the case) from the set of all possible tours.

Step 2. Improve the tour using one of the branch exchange heuristics.

Step 3. Continue Step 2 until no additional improvement can be made.

The branch exchange procedure terminates at a local optimum. For a given k, if we define a k-change of a tour as consisting of the deletion of k branches in a tour and their replacement by k other branches to form a new tour, then a tour is k-optimal if it is not possible to improve the tour via a k-change. Steps 2 and 3 would generate this k-optimal tour. Since the 2-opt exchange procedure is weaker than the 3-opt exchange procedure, it will generally terminate at an inferior local optimum. Similarly, a k-opt exchange procedure will generally terminate with a better local optimum than will a 3-opt exchange procedure for $k \geq 4$. Figure 2.10 illustrates a 2-opt exchange. Note that the exchange is well-defined at each step. Once we have decided to drop arcs (A, B) and (E, F) from the current solution in order to reduce tour length, we must introduce arcs (A, E) and (B, F), since introducing (A, F) leads to a subtour solution and introducing (A, B) leads to the original solution. Note that symmetry is required here since the direction on arcs (B, C), (C, D), and (D, E) is reversed.

These branch exchange procedures are important since they illustrate a general approach to heuristics for combinatorial optimization problems. In addition, they have been used to generate excellent solutions to large-scale traveling salseman problems in a reasonable amount of time.

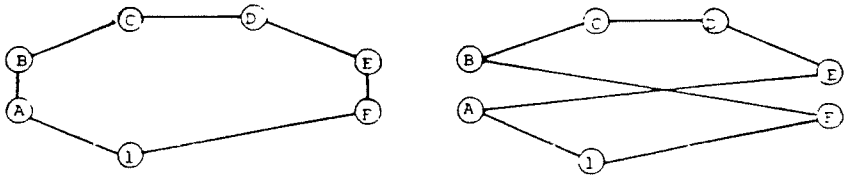

Fig. 2.10. Initial tour and feasible 2-opt swap.

In terms of worst case performance, Rosenkrantz, Stearns, and Lewis[568] give some partial results. For example, they show (assuming the triangle inequality) that a 2-optimal tour can be almost twice the length of an optimal tour.

Roughly speaking, a 3-opt procedure requires n times the work that a 2-opt procedure needs. To obtain close approximations to the length of the optimal tour using this strategy, one should repeat a 3-opt procedure for a number of starting tours[568]. Computationally, this approach becomes burdensome quickly. We, therefore, consider a class of heuristic methods which is computationally less expensive but as accurate as the repeated use of the 3–opt procedure. These are composite procedures.

2.3.3 *Composite procedures*

The basic composite procedure can be stated as follows:

Step 1. Obtain an initial tour using one of the tour construction procedures.

Step 2. Apply a 2-opt procedure to the tour found in Step 1.

Step 3. Apply a 3-opt procedure to the tour found in Step 2.

The composite procedure is relatively fast computationally and gives excellent results.

The idea behind the composite procedure is to get a good initial solution rapidly and hope that the 2-opt and 3-opt procedures will then find an almost-optimal solution. In this way, the 3-opt procedure which is the computationally most expensive step of the three, need only be used once.

The following are some possible variants of the composite procedure outlined above:

Variant 1. Run the composite procedure without Step 2.

Variant 2. Run the composite procedure without Step 3. This variant gives very fast running times and very accurate results but would not be expected to perform as accurately as the three-step composite procedure.

Variant 3. Run the composite procedure a few times, using different tour construction algorithms in Step 1. This variant, although more expensive computationally than the three-step composite procedure, is expected to give better results.

Variant 4. In Step 1 of the composite procedure, only run the tour construction procedure from a small number of starting nodes. Then perform Steps 2 and 3 from the best of these solutions.

All of the heuristic algorithms described in this section are intended for symmetric TSP's (although some can be applied to asymmetric problems as well). Algorithms specifically designed for asymmetric TSP's have been proposed by Akl[4], Frieze *et al.*[236], Karp[373], Van Der Cruyssen and Rijckaert[659], and Kanellakis and Papadimitriou[370]. We discuss only the first of these methods here.