Heuristics for the TSP

I. An α-approximation heuristic: MST 2-approximation

Steps:

1. Generate a minimum spanning tree MST^* of the nodes N.
2. Let $LIST$ contain two copies of each arc in the minimum spanning tree, MST^*.
3. Generate a $WALK$ of nodes of G using each arc in $LIST$ exactly once; $\{v_1, v_2, ..., v_{2(n-1)}, v_1\}$. Note that v_i are not unique.
4. Create a heuristic tour T^H by following the nodes in the order specified by $WALK$, but skipping repeat visits to any node (the "taking shortcuts" process):
   (a) Let $t(i) = 0$ for all $i \in N$; $T^H = \emptyset$.
   (b) for $i = 1$ to $2(n-1)$:
      i. if $t(v_i) = 0$, then $T^H \leftarrow T^H + v_i$ and $t(v_i) = 1$.
5. $T^H \leftarrow T^H + v_1$.

In this method, recognize that $WALK$ does not fit with our definition of a tour, since it may contain nodes that are visited more than once. So, we simply eliminate repeat visits in our WALK to identify T^H, sometimes referred to as an embedded tour.

Generating a walk of the nodes
Generating the walk of the nodes above is always possible, since the arcs in $LIST$ form what is known as an Eulerian graph. Walks in such graphs can be identified using a simple recursive procedure. Pick an arbitrary node v_1 as a starting point, and create a walk of some (but possibly not all) of the nodes: $WALK_1 = \{v_1, v_2, v_3, v_2, v_1\}$. Now remove the arcs you just used from $LIST$, and examine the remaining structure. You now may have additional disconnected walks that may need to be attached to nodes in the main walk. For example, perhaps you find a new walk attached to v_3: $\{v_3, v_4, v_5, v_4, v_3\}$. Again, remove these arcs from $LIST$. Then, you update $WALK_2 = \{v_1, v_2, v_3, v_4, v_5, v_4, v_3, v_2, v_1\}$. Keep proceeding recursively in this fashion until no arcs are left in $LIST$.

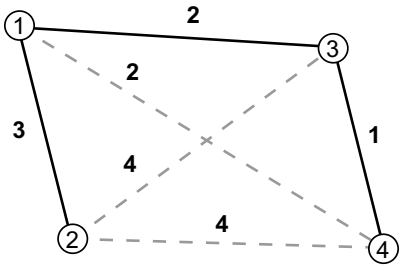

Figure 1: The dark black lines represent the minimum spanning tree.

For example, in Figure 1 suppose a TSP tour is started at node 1. One example of a $WALK$ that could be generated would be $WALK = \{1, 2, 1, 3, 4, 3, 1\}$. This tour contains unnecessary return visits to nodes, and has a total cost of $12 = 2 * (3 + 2 + 1)$. Applying the shortcut procedure, a tour would begin at 1 and move to 2. The walk specifies a return to 1, so we look ahead to the next unvisited node, which is 3. The shortest path from 2 to 3 is along arc $(2, 3)$, so we add this arc to the tour. T is now $T = \{(1, 2), (2, 3)\}$. Next, $WALK$ tells us to visit node 4 and since it has not yet been visited, we move along the MST: $T = \{(1, 2), (2, 3), (3, 4)\}$. Next, the $WALK$ specifies a return to 3. Since it has already been visited, we move ahead to the next unvisited node. Since all nodes have been visited, we return to the start node 1 along the shortest path, which happens to be arc $(4, 1)$. The final tour is $T = \{(1, 2), (2, 3), (3, 4), (4, 1)\}$, with a cost of 10.

Proof of the 2-approximation

The MST 2-approximation heuristic will generate a TSP tour with a cost no greater than double the optimal cost for networks satisfying the triangle inequality. This is not good, but at least it's a guarantee!

Proof:

First, we know that the heuristic tour we generated has a cost less than twice the minimum spanning tree, since we started with a walk with cost exactly equal to $2C(MST)$ and then reduced its cost by taking shortcuts due to the Δ-inequality:

$$C(T^H) \leq 2C(MST)$$

Now, suppose that you were to remove a single arc from the optimal traveling salesman tour; the result is a tree. Let's call it $TSPTREE$. It's clear that the minimum spanning tree cost is always no greater than the cost of any $TSPTREE$:

$$C(MST) \leq C(TSPTREE)$$

and of course, the cost of any $TSPTREE$ is no greater than the cost of the optimal TSP tour for problems with nonnegative arc costs:

$$C(TSPTREE) \leq C(TSP^*)$$

The result now follows:

$$C(T^H) \leq 2C(MST^*) \leq 2C(TSPTREE) \leq 2C(TSP^*)$$

B. Christofides' Heuristic

Christofides' heuristic has the best known worst-case performance bound for the traveling salesman problem on complete networks satisfying the triangle-inequality. Similar to the Twice-Around MST heuristic, Christofides' heuristic utilizes the concept of minimum spanning trees. In addition, the heuristic relies on the idea of minimum cost perfect matchings (MCPM). Consider the following definitions and facts:

Definition 3 *(Node Degree)*
The degree *of a node $i \in N$ with respect to a set of arcs T is the number of arcs in T that are adjacent, or touching, node i.*

Lemma 1 *(An Euler Lemma)*
Given a set of nodes N and a set of arcs representing a spanning tree T, the number of nodes with odd degree with respect to the arc set T is even.

Definition 4 *(Perfect Node Matching)*
Given a set of nodes S such that $|S|$ is even, a perfect node matching *W is a set of $\frac{|S|}{2}$ node pairings such that each node in S is in exactly one pairing. In a complete network, each node pairing (i, j), $i, j \in S$ represents an arc in the network. The cost of matching W is given by $C(W) = \sum_{(i,j)\in W} c_{ij}$.*

Lemma 2 *(Another Lemma from Euler)*
Suppose given a complete network, you find a spanning tree T. Then, you find a perfect node matching W of the nodes in N with odd degree with respect to T. Now consider the set of arcs $R = T \cup W$. Now, the nodes in N each have an even degree with respect to R and are all connected. Thus, a traveling salesman tour of the nodes can always be constructed using only the arcs of R (with no duplicates).

The above definitions and lemmas motivate Christofides' heuristic. Here is a sketch of the method:

Christofides' Heuristic

Steps:

1. Given an undirected, complete network G satisfying the Δ-inequality, find a minimum spanning tree MST on G.

2. Let S be the nodes of G with odd degree with respect to MST.

3. Find a perfect node matching W^* of the nodes in S with minimum total cost. (This problem can be solved in $O(n^3)$ time by the method of Lawler).

4. Generate a tour of the nodes T^C using arcs in the set $MST \cup W^*$ exactly once. (T^C can again be improved by introducing shortcuts to avoid revisiting nodes).

It remains to prove the worst-case performance bound for this heuristic.

Theorem 1 *(Christofides Worst-case Performance Bound) Christofides heuristic is* 1.5*-optimal for every complete network G satisfying the Δ-inequality, that is:*

$$C(T^C) \leq \frac{3}{2} C(TSP^*) \tag{1}$$

where TSP^* is an optimal traveling salesman tour on G.

Proof:

1. First, it is clear that $C(T^C) \leq C(MST) + C(W^*)$, since T^C uses only arcs in the set $MST \cup W^*$ and may include shortcuts.
2. It is additionally clear that $C(MST) \leq C(TSP^*)$. (This was shown using earlier arguments: if you remove one arc from TSP^* you create a spanning tree, which has a cost necessarily no less than MST, the minimum spanning tree.)
3. Thus, we now have shown from (1) and (2) that $C(T^C) \leq C(TSP^*) + C(W^*)$.
4. We now claim that $C(W^*) \leq \frac{1}{2} C(TSP^*)$. The proof is as follows:
   (a) Let $t_1, t_2, ..., t_{2k}$, where $k = \frac{|S|}{2}$, be the nodes in S listed in the order they would be visited by the optimal tour TSP^* on G. Note that we need not know what this tour is for the proof. And further note that TSP^* visits all the nodes in N, not just the nodes in S. As a quick example, suppose we have 7 nodes and $TSP^* = \{3, 4, 7, 2, 5, 1, 6, 3\}$. Suppose that $S = \{2, 4, 6, 7\}$. Then, $t_1 = 4, t_2 = 7, t_3 = 2, t_4 = 6$.
   (b) Consider the following two feasible perfect matchings on S:
   $M_1 = \{(t_1, t_2), (t_3, t_4), ..., (t_{2k-1}, t_{2k})\}$ and $M_2 = \{(t_2, t_3), (t_4, t_5), ..., t_{2k}, t_1)\}$.
   Note that the union of M_1 and M_2 describes a tour that visits only the nodes in S in the order of the optimal tour TSP^*.
   (c) First, it should be clear that $C(M_1) + C(M_2) \leq C(TSP^*)$. To see why, recognize that since $M_1 \cup M_2$ visits nodes in S in the *same order* in which they would be visited by TSP^*, $M_1 \cup M_2$ is a tour that includes some "shortcuts" by skipping nodes not in S (*i.e.*, in $N \setminus S$). By the Δ-inequality, therefore, the expression holds.
   (d) Second, it is also true that $C(W^*) \leq \frac{1}{2}(C(M_1) + C(M_2))$. This is well-known; the cost of an optimal solution to any minimization problem is always no greater than the average cost of two feasible solutions.
   (e) From (c) and (d), it follows that $C(W^*) \leq \frac{1}{2}(C(M_1) + C(M_2)) \leq \frac{1}{2} C(TSP^*)$
5. Thus, from (3) and (4) the result follows.

□

II. Construction Heuristics

Nearest-Neighbor heuristic

The nearest-neighbor heuristic is a greedy local search method for TSP which maintains at each step a current path. Starting at an arbitrary starting node, it scans "neighbors" of either one or both current path endpoints and adds the closest neighbor to the tour. The last step of the method connects the remaining two endpoints together.

Initialization:

Given an undirected network $G = (N, A)$, with costs c_{ij} associated with each $(i, j) \in A$.

Alternative 1: $P = \{i, j\}$ where arc (i, j) has minimum cost among all arcs
Alternative 2: Choose an arbitrary start node i, and find $j \in N \setminus \{i\}$ such that c_{ij} is minimized. Let $P = \{i, j\}$.

Steps:
while $|P| < n$:

1. Let $s = \mathit{begin}(P)$ and $t = \mathit{end}(P)$, the nodes at the beginning and end of the current path.

2. Choose the node $k \in N \setminus P$ to enter the path:

Alternative 1: Let k be such that c_{tk} is minimized. $P = P + \{k\}$.

Alternative 2: Let k be such that c_{tk} or c_{ks} is minimized. The new path P is either $P + \{k\}$ or $\{k\} + P$ respectively.

end;
Let $T = P + \{\mathit{begin}(P)\}$.

Tour Generation: The nodes in T now define a TSP-tour when followed in order.

This heuristic is simple to understand without resorting to precise pseudocode. At each step, the current tour T contains two *endpoints*, that is, nodes connected to only one arc in the tour. At each step of the algorithm, the arc with minimum cost connecting an endpoint to a node not yet visited by the tour is added to the tour. If no unvisited nodes remain, a connection is formed between to the two endpoints to complete the tour.

Nearest insertion heuristic

Initialization:

Given an undirected network $G = (N, A)$, with costs c_{ij} associated with each $(i, j) \in A$, where c_{ij} satisfies the Δ-inequality.

Find the arc (i, j) with minimum cost in A. Let $T = \{i, j, i\}$

Steps:

while $|T| < n + 1$:

1. Find the node $j \notin T$ that is *closest* to a node currently in T. That is, find $j \notin T, i \in T$ such that c_{ij} is minimized.

2. Find the *insertion location* for node j into the existing tour. To do so, find the arc (ℓ, k) in the current tour with the minimum insertion cost: $c_{\ell j} + c_{jk} - c_{\ell k}$. Insert j between ℓ and k in T.

end;

The nearest-insertion heuristic is a $2 - \frac{2}{n}$-optimal heuristic.

Farthest insertion heuristic

Initialization:

Given an undirected network $G = (N, A)$, with costs c_{ij} associated with each $(i, j) \in A$, where c_{ij} satisfies the Δ-inequality.

Find the arc (i, j) with maximum cost in A. Let $T = \{i, j, i\}$

Steps:

while $|T| < n + 1$:

1. Let $C_j(T)$ be $\min_{i \in T} c_{ij}$ for all nodes $j \notin T$. The node j to be inserted into the tour is one with maximum $C_j(T)$.

2. Find the *insertion location* for node j into the existing tour. To do so, find the arc (ℓ, k) in the current tour with the minimum insertion cost: $c_{\ell j} + c_{jk} - c_{\ell k}$. Insert j between ℓ and k in T.

end;

In English, Step 1 finds the node j not on the tour for which the minimum distance to a vertex on the tour is maximum, and adds that node to the current tour.

Cheapest insertion heuristic

Initialization:

Given an undirected network $G = (N, A)$, with costs c_{ij} associated with each $(i, j) \in A$, where c_{ij} satisfies the Δ-inequality.

Alternative 1: Find the arc (i, j) with maximum cost in A. Let $T = \{i, j, i\}$

Alternative 2: Use some procedure to develop an initial subtour, T. Frequently, convex hulls (or close approximations) are used.

Steps:

while $|T| < n + 1$:

1. Let $C_{j\ell}(T)$ be $c_{\ell j} + c_{jnext(\ell)} - c_{\ell next(\ell)}$ for all nodes $j \notin T$.

2. Let j^* and ℓ^* be the indices of the minimum cost $C_{j\ell}$. Insert node j^* after node ℓ^* in T.

end;

In this heuristic, we simply calculate an *insertion cost matrix* at each step representing the cost of inserting node $j \notin T$ after node $\ell \in T$. We choose the minimum cost insertion at each iteration.

III. Improvement Heuristics

Given that a tour T has already been created via some method, it may be possible to improve the tour with an *improvement heuristic*. One often employed improvement heuristic is called λ-opt. A λ-opt heuristic is based on the concept of λ-optimality, which is a local neighborhood condition. Essentially, a tour is said to λ-optimal if there is no way to replace λ arcs in T with λ arcs not in T to create a new feasible tour with lower cost:

Definition 5 *(λ-optimality)*
Given a CUNDI $G = (N, A)$ and a tour T_i, let $A(T_i)$ be the set of arcs defined by the nodes in T_i: $A(T_i) = \{(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n), (v_n, v_1)\}$. Then, tour T_i is λ-optimal if and only if there does not exist two sets of arcs, $X \in A(T_i)$, $Y \in A \setminus A(T_i)$ such that:

1. $|X| = |Y| = \lambda$
2. $\sum_{(i,j)\in X} c_{ij} - \sum_{(i,j)\in Y} c_{ij} > 0$
3. *The set $A(T_i) + Y \setminus X$ allows a new feasible TSP tour sequence T_{i+1}.*

These ideas motivate the following generic heuristic. Note that technically, the Δ-inequality is not necessary for the validity of this heuristic.

λ-opt heuristic

Initialization:
Given some initial traveling salesman tour T^0
$k = 0$

Iterations:

1. Identify a feasible λ-exchange with positive savings. That is, find two sets $X \in A(T_k)$ and $Y \in A \setminus A(T_k)$ such that $|X| = |Y| = \lambda$, $\sum_{(i,j)\in X} c_{ij} - \sum_{(i,j)\in Y} c_{ij} > 0$, and that the set of arcs $T_k + Y \setminus X$ admits a feasible tour sequence. If no sets can be identified, STOP. T_k is λ-optimal.
2. Let T^{k+1} be the new tour sequence generated from X and Y.
3. Repeat from 1.

Analysis:
How many options for removing λ arcs in T_k and replacing them with λ new arcs exist? In a network with n nodes, each tour contains n arcs. Thus, there are $\binom{n}{\lambda}$ ways of removing λ arcs from a tour. If removing a set of λ arcs results in exactly x nodes that are no longer adjacent to any arc in the tour, the number of new ways to reconnect the nodes is $[(\lambda - 1)!2^{\lambda-1-x}] - 1$.

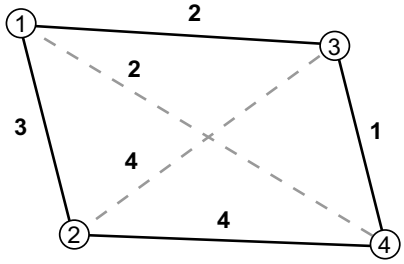

Figure 2: The dark black lines represent the TSP tour. Note that in this case, $\binom{4}{2} = 6$, but 4 of these choices leave one node no longer adjacent to any arc in the tour. The removal of arcs $(1, 2)$ and $(1, 3)$ is such an example.

The two most commonly employed heuristics are 2-opt and 3-opt. Let's analyze how many new tours are created in each iteration in these cases.

2-opt: In this case, there are $\binom{n}{2}$ ways of removing 2 arcs. Additionally, in n of these cases, one node is left no longer adjacent to any arc. In these n cases, there is no new way to connect the nodes: $[(2-1)!2^{2-1-1}] - 1 = [1!2^0] - 1 = 0$. In the remaining cases, there is exactly one new way to reconnect the nodes: $[(2-1)!2^{2-1}] - 1 = [1!2^1] - 1 = 1$. Therefore, the total number of possible 2-opt exchanges is:

$$\left(\binom{n}{2} - n\right)$$

3-opt: In this case, there are $\binom{n}{3}$ ways of removing 3 arcs. In n of these cases, two nodes are left no longer adjacent to any arc. In these n cases, there is exactly one new way to connect the nodes: $[(3-1)!2^{3-1-2}] - 1 = [2!2^0] - 1 = 1$. In $n(n-4)$ cases, one node is left no longer adjacent to an arc. In these cases, there are $[(3-1)!2^{3-1-1}] - 1 = [2!2^1] - 1 = 3$ new ways to reconnect. In the remaining cases, there are $[(3-1)!2^{3-1}] - 1 = [2!2^2] - 1 = 7$ new ways to reconnect. Therefore, the total number of new tours to be examined at each iteration is:

$$7\left(\binom{n}{3} - n - n(n-4)\right) + 3n(n-4) + n$$

Using these equations for a network 50 nodes, the 2-opt heuristic would examine at most 1175 exchanges each iteration while 3-opt would examine no more than 127,700 exchanges each iteration.

Implementation Issues

Implementing an efficient 2-opt or 3-opt procedure efficiently requires addressing a number of issues. Most implementations attempt to identify the set of removed arcs X and added arcs Y as follows. Consider first the case with 2-opt:

1. Choose a node t_1 arbitrarily in N.
2. Choose t_2 such that T includes $\{..., t_1, t_2, ...\}$ or $\{..., t_2, t_1, ...\}$. Thus, (undirected) arc $x_1 = (t_1, t_2)$.

3. Choose now an arc y_1 that will be added to the tour following the removal of x_1. It is clear that x_1 and y_1 must share an endpoint! So, suppose $y_1 = (t_2, t_3)$: the choice required is to choose node t_3!

4. Choose now another arc to leave the tour. There are two choices dictated by t_3, since the leaving arc must be adjacent! $x_2 = (t_3, t_4)$ such that T contains $\{..., t_3, t_4, ...\}$ or $\{..., t_4, t_3, ...\}$.

5. Finally, y_2 is given since the tour must be reconnected: $y_2 = (t_4, t_1)$.

This method is used to identify a potential exchange with reduction of $c_{y_1} - c_{x_1} + c_{y_2} - c_{x_2}$; many implementations will simply find an exchange with negative reduction, make the exchange and then search again.

The Lin-Kernighan heuristic

Since a blend of different λ-opt exchanges is likely to lead to the most improvement to a TSP tour, a systematic method for performing variable λ-opt exchanges would likely be an effective practical heuristic. This is the idea behind the LK heuristic, which is widely recognized as the best-performing practical heuristic for solving TSP problems. For the small (fewer than 50 node) geographic network problems that we tend to find in transportation and logistics, most implementations of LK will actually find optimal solutions! In fact, in the most recent DIMACS TSP Challenge conducted in 2000, a publicly-available LK code solved every random 1,000 node instance to within 0.2% of optimality (verified with an optimal code) in no longer than 20 seconds on standard personal computers.

In LK, we consider *sequential* exchanges where we identify a pair of arcs at each step that we will swap; by definition, they must have an endpoint in common. We will choose at least two exchanges such that the sequence of exchanges defines a new feasible tour, and the reduction of *each* exchange improves the tour. Here's how the generic form works. Define the reduction $r_i = c_{y_i} - c_{x_i}$, and let $R_i = \sum_{k=1}^{i} r_k$:

Initialization:

Given some initial tour T_0, let $T \leftarrow T_0$

Iterations:

1. Let $i = 1$. Choose t_1.

2. Choose $x_1 = (t_1, t_2) \in T$.

3. Choose $y_1 = (t_2, t_3) \notin T$ such that $R_1 < 0$. If no such exists, go to Step 11

4. Let $i = i + 1$

5. Choose $x_i = (t_{2i-1}, t_{2i}) \in T$ such that:

   (a) If $y_i = (t_{2i}, t_1)$ is added, a feasible tour T_N results.

(b) $x_i \neq y_s$ for all $s < i$.

If $R_k < 0$, then T_N is a better tour than T. Let $T \leftarrow T_N$ and go to Step 1.

6. Choose $y_i = (t_{2i}, t_{2i+1}) \notin T$ such that:
   (a) $R_i < 0$
   (b) $y_i \neq x_s$ for all $s < i$.
   (c) x_{i+1} exists; that is, there is a possible feasible choice for the next arc to leave.

   If such y_i exists, go to Step 4
7. If there is an untried alternative for y_2, let $i = 2$ and go to Step 6.
8. If there is an untried alternative for x_2, let $i = 2$ and go to Step 5.
9. If there is an untried alternative for y_1, let $i = 1$ and go to Step 3.
10. If there is an untried alternative for x_1, let $i = 1$ and go to Step 2.
11. If there is an untried alternative for t_1, go to Step 1.