An Excel-based, Rotating Constellation Heuristic for Solving the Travelling Salesman Problem

Richard J. Perle
Department of Finance and Computer Information Systems, Loyola Marymount University, Los Angeles, CA, USA

Abstract - The Travelling Salesman Problem (TSP) is a well-known combinatorial problem that finds many applications in operational settings such as product distribution and manufacturing. Finding exact solutions to TSPs can be difficult, so heuristic methods are sometimes implemented. This paper develops and tests the performance of a new and novel, Excel-based heuristic algorithm for solving Euclidian plane, symmetric TSPs where the X-Y coordinates of all nodes are known. Test results show that the method works well for small problems. The method has an error function that varies linearly with the problem size. An advantage of the algorithm is that it is relatively easy to implement and so might be useful in smaller organizations which do not possess sophisticated mathematical or financial resources. It also might be used in an academic setting to demonstrate heuristic solution processes.

Keywords: Traveling salesman problem, Heuristic, Algorithm

1 Introduction

The Travelling Salesman Problem (TSP) is a well-known and important example of combinatorial sequencing problems that find wide practical application in many fields. The classic case is where a salesperson wants to travel from his home base and visit clients in a number of other cities. The problem is to define a least-costly tour from city-to-city in which each city is visited only once and the salesperson ends up back at the home base. Interest in and application of the TSP began with the seminal paper by Dantzig, Fulkerson and Johnson [1] which found the shortest-route tour of 49 cities in the U.S. Other important, practical and useful examples of combinatorial sequencing problems with added refinements and extensions are; assigning airliners to routes [2], routing delivery trucks [3], drilling holes in printed circuit boards [4], order picking in a warehouse [5], and sequencing jobs on a machine [6].

A characteristic of any TSP is that it is easy to describe but difficult to solve to an exact, provable optimum. An exact solution to a TSP can be found by branch-and-bound integer programming methods, described by Lawler, et. al. [7] and branch-and-cut methods, described by Junger, et al. [8]. An exact solution can also be found by explicit enumeration of all possible solutions; however, this is not a viable methodology for larger problems as the number of possible solutions is a factorial of the number of nodes (cities). As a result of the difficulty in finding exact solutions, practical and useful solutions to TSPs can be obtained by the use of heuristic algorithms. A heuristic algorithm is a solution procedure that can lead to a good solution, but one that is not necessarily optimal. There are three general types of TSP heuristics; (1) construction methods, (2) improvement methods and (3) metaheuristic methods. Construction methods start at an arbitrary node and then select succeeding nodes according to a criterion such as cheapest or shortest distance. Well-known examples of construction methods are variations of the Nearest Neighbor Greedy (NNG) algorithm [9] which will be used for comparison purposes in this paper. Improvement methods start with a feasible tour and then make changes in an effort to find a shorter tour. The 2-Opt, 3-Opt and Lin-Kernighan algorithm [10] are examples of improvement methods. Metaheuristics such as simulated annealing, tabu search, genetic algorithms and artificial neural networks search neighborhoods for local optima and then use that information to search for better solutions without getting trapped in any one local neighborhood. A good description of basic metaheuristic methods can be found in [11]. A disadvantage of many of these methods is that they usually require specialized software that may be difficult or expensive to acquire and implement, especially for small companies that may find less than exact solutions to be an acceptable trade-off for a simpler solution methodology.

The Rotating Constellation Heuristic (RCH) algorithm described in this paper is a hybrid method. It starts with a feasible tour constructed by a simple node-to-node process, then systematically generates a subset of additional complete, feasible tours, ultimately selecting the best tour from the set of feasible tours. It has the advantage that the software may be developed and implemented by using the Excel sort function, and does not require extensive training in mathematics or expertise in a programming language. However, some basic skills in Excel Macros and VBA would be helpful to reduce the amount of time and effort required to find the best solution that the method is capable of delivering. The practical usefulness of any non-optimal heuristic solution, of course, would depend on its expected accuracy.
which in this paper is measured as the expected percent over an optimal or benchmark tour.

The next two sections of this paper will present the RCH process logic by example, and then the mathematics of the general RCH model will be developed. Its robustness and accuracy will then be benchmarked against actual TSP data for which a very good or optimum solution is known.

2 The Rotating Constellation Heuristic: Description by Example

As a simple example consider a set of ordered pair X-Y values for a ten node TSP, shown in Table 1, where the pairs are sorted in ascending X-value order.

<table>
<thead>
<tr>
<th>Node</th>
<th>X-value</th>
<th>Y-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>53</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>81</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
<td>67</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>39</td>
</tr>
<tr>
<td>6</td>
<td>54</td>
<td>57</td>
</tr>
<tr>
<td>7</td>
<td>61</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>68</td>
<td>37</td>
</tr>
<tr>
<td>9</td>
<td>87</td>
<td>49</td>
</tr>
<tr>
<td>10</td>
<td>93</td>
<td>72</td>
</tr>
</tbody>
</table>

The RCH algorithm is an eight step procedure, the first six of which are listed below. Steps 2, 4 and 5 are implemented by an Excel sort command. The algorithm is applied iteratively using Excel VBA in a search for a best solution tour distance value.

The RCH algorithm:
1. Assign a unique number to each node
2. Sort the complete set of nodes on the X-values, from low to high.
3. Based on the X-values, separate the nodes into two equal size sets.
   - Left-most set = nodes with the smallest X-values.
   - Right-most set = node with the largest X-values.
4. Sort the left-most set of nodes on the Y-values, from low to high.
5. Sort the right-most set of nodes on the Y-values, from high to low.
6. Connect the two sets to identify the complete tour and calculate the tour distance value.

Fig. 1 shows a graphic example of the first six steps of the algorithm applied to the data in Table 1. The vertical dashed line separates the 10 nodes into two equal size sets of five nodes each, designated as the left-most set and the right-most set (steps 2 and 3). The arrows show the tour path that the first six steps of the first iteration of the RCH algorithm would find from start to finish. Step 4 sorts the left-most set of nodes on the Y-values from low to high; step 5 sorts the right-most set on the Y-values, from high to low. The tour path by node number is 3-5-1-4-2-7-10-6-9-8-3. Its length is 314.20.

It can also be inferred from Fig. 1 that an advantage of the RCH algorithm is that there will be no crossing paths within either set. Crossing paths will add to the tour length, assuring that it is not optimal. The only time a crossing path could (but not necessarily) be generated is on the two paths that connect the sets together; the upper-most arrow and the lower-most arrow in Fig. 1.

The entire set of nodes is now treated as a constellation of ordered pair points that will be rotated about its geometric center in successive iterations of the RCH algorithm. The rotation does not change the distance between any pair of nodes. After each iterative rotation, steps 2 through 6 in the algorithm are repeated to generate a new solution. For example, the constellation of Fig. 1 is rotated 90 degrees clockwise and a new RCH solution is generated from the new, rotated X-Y coordinates shown in Table 2.

<table>
<thead>
<tr>
<th>Node</th>
<th>New X-value</th>
<th>New Y-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16.6</td>
<td>82.8</td>
</tr>
<tr>
<td>2</td>
<td>29.6</td>
<td>70.8</td>
</tr>
<tr>
<td>3</td>
<td>43.6</td>
<td>95.8</td>
</tr>
<tr>
<td>4</td>
<td>57.6</td>
<td>76.8</td>
</tr>
<tr>
<td>5</td>
<td>71.6</td>
<td>83.8</td>
</tr>
<tr>
<td>6</td>
<td>70.6</td>
<td>41.8</td>
</tr>
<tr>
<td>7</td>
<td>62.6</td>
<td>9.8</td>
</tr>
<tr>
<td>8</td>
<td>47.6</td>
<td>48.8</td>
</tr>
<tr>
<td>9</td>
<td>39.6</td>
<td>15.8</td>
</tr>
<tr>
<td>10</td>
<td>27.6</td>
<td>34.8</td>
</tr>
</tbody>
</table>
The new tour path by node number as shown in Fig. 2 is 9-10-2-1-3-5-4-8-6-7-9. The tour length is reduced to 262.83 which is a 31.69 percent improvement over the tour in Fig. 1. It is also optimal, proved by explicit enumeration.

**3 The Rotating Constellation Heuristic: General Model**

Assume the Cartesian coordinates, \(x_i, y_i\) (i = 1 to n) in a two-dimensional flat plane for a TSP with n nodes are known. The symmetric Euclidian distance, \(d_{ij}\) between any two nodes, i and j (i ≠ j) is calculated as:

\[
d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}
\]  

The ordered pair, \(x_c,y_c\) is defined as the geometric center of the constellation and is calculated as the mean of the X-values and the Y-values:

\[
x_c = \frac{\sum x_i}{n}
\]  
\[
y_c = \frac{\sum y_i}{n}
\]

Consider now a translated coordinate system centered on \(x_c,y_c\) in the original Euclidian plane which is divided into four quadrants (Q1 to Q4) as shown in Fig. 3. An arbitrary node, \(x_iy_i\) and its vector is shown in Quadrant 2 along with its angle, \(\theta_i\), relative to the positive X-axis which is defined as zero degrees. The Euclidian distance from \(x_c,y_c\) to \(x_i,y_i\) is \(h_i\) and calculated as:

\[
h_i = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2}
\]

Fig. 2. Two-set Solution Path Rotated 90 Degrees, for Table 2 Data

Even though 90 degrees is the best rotation angle there is no way to know, a priori, that this is the case. So the objective in the RCH algorithm is to rotate the constellation incrementally, attempting to generate a new solution with each incremental rotation until a best, but not necessarily optimal solution is found. The condition that creates a new solution path occurs when a node passes from the right-most set to the left-most set and vice-versa due to rotation. In effect the two sets evolve by exchanging a pair of nodes at each incremental rotation. Consequently, the largest number of different solutions that the rotation process is capable of generating is \(n\), the number of nodes. The set of \(n\) unique solutions, if they are all detected, is generated over a total rotation of only 180 degrees. Rotating the constellation beyond 180 and on to 360 degrees simply generates a repeat of the zero to 180 degree rotation solutions, except the tour path is in the opposite direction. But the tour length is obviously the same in either direction. There are a number of rules that could be used to determine the incremental angle of rotation, but the simplest rule, used here, is to rotate 180/n degrees in contiguous increments, starting from zero to (180 - 180/n) degrees. The zero degree position is the original configuration of the constellation. Two additional steps (7 and 8) are now added to complete the RCH algorithm.

7. Incrementally rotate the constellation 180/n degrees, then repeat steps 3 - 6 until the constellation has been rotated a total of (180 - 180/n) degrees.
8. Select the solution with the best tour length.

Another question to address is how to handle an instance where there is an odd number of nodes. Obviously the constellation cannot be divided into two sets, each with the same number of nodes. The easiest way to handle this is to arbitrarily assign one extra node to either set, which is done here. There are more sophisticated ways to solve the problem such as creating a “dummy” node as a duplicate of an existing node which will add nothing to any tour length because the sorting process should always connect the dummy node to its real node. However, this could perturb the geometric center of the constellation leading to unknown effects. The general RCH model will now be developed.
The calculation of any value of $\theta_i$ depends on which quadrant the node is in.

If: $x_i - x_c >= 0$ and $y_i - y_c >= 0$, the node is in Q1, and:
$$\theta_i = \sin^{-1}\left(\frac{(y_i - y_c)}{h_i}\right)$$
(5)

If: $x_i - x_c < 0$ and $y_i - y_c >= 0$, the node is in Q2, and:
$$\theta_i = 180 - \sin^{-1}\left(\frac{(y_i - y_c)}{h_i}\right)$$
(6)

If: $x_i - x_c < 0$ and $y_i - y_c < 0$, the node is in Q3, and:
$$\theta_i = 180 - \sin^{-1}\left(\frac{(y_i - y_c)}{h_i}\right)$$
(7)

If: $x_i - x_c >= 0$ and $y_i - y_c < 0$, the node is in Q4, and:
$$\theta_i = 360 + \sin^{-1}\left(\frac{(y_i - y_c)}{h_i}\right)$$
(8)

Once the set of $\theta_i$ have been calculated the entire constellation is incrementally rotated about the geometric center by angle, $\phi_k$ ($k = 0$ to $n-1$), where $k$ is the $k$th iteration out of $n$ total iterations.

$$\phi_k = \left(\frac{180}{n}\right)k$$
(9)

If $k = 0$, then $\phi_k = 0$, and the constellation is in its original, unrotated position. After each rotation, each node’s new vector angle relative to the zero degree position in Fig. 3 is ($\theta_i - \phi_k$), assuming clockwise rotation. New coordinates, $x,y_i$ in the original Euclidian plane can then be calculated for each rotated node:

$$x_i = h_i(\cos(\theta_i - \phi_k))x_c$$
(10)

$$y_i = h_i(\sin(\theta_i - \phi_k))y_c$$
(11)

Of course the distance, $d_{ij}$ between any two rotated nodes is the same as the original, unrotated distances:

$$d_{ij} = d_{ij}$$
(12)

And the tour distance value, $v_k$ for the $k$th contiguous rotational iteration is:

$$v_k = \sum d_{ijk}$$
(13)

where $i$ and $j$ are determined by the final sort sequence which becomes known after step 7 in the RCH algorithm. The best overall solution, $v^*$ is then selected in step 8 of the RCH as:

$$v^* = \min\{v_k\}$$
(14)

which occurs at a rotation angle of $\phi^*$.

4 Computational Test Results and Discussion

In this section results are presented on the performance of the RCH algorithm compared to the performance of the NNG algorithm using instances available in TSPLIB [12]. Instances of up to 76 nodes were selected from TSPLIB for which the X-Y coordinates and the optimal or best tour were known and available. All tour values are assumed to be correct as specified in TSPLIB. NNG and RCH solutions were generated for each instance and compared to the TSPLIB tour. Overall results are presented in Table 3.

It can be seen in Table 3 that the RCH algorithm at $\phi^*$ performed better than the NNG algorithm for instances of 38 nodes, or less. For these four instances, rotation of the constellation from $\phi = 0$ to $\phi = \phi^*$ improved the solution tour from an average of 35.87 percent over TSPLIB to 14.96 percent over TSPLIB. As might be expected, it can be seen in Table 3 that the error increases with the number of nodes. The data in the second and last columns in Table 3 can be used to estimate the expected accuracy of the RCH algorithm, relative to TSPLIB, as a function of the number of nodes which have been divided into two sets, each of size $n/2$.

Defining:

$$p = \% \text{ over TSPLIB, RCH at } \phi = \phi^*$$

$n = \text{ number of nodes}$

Simple linear regression generates the following equation with adjusted $R^2 = 0.89.$

$$p = 0.93(n) - 10.56$$
(15)

Obviously, the $p$ value in (15) cannot be less than zero. The $p$-value is equal to zero when the $n$-value is 11.35. Given that the $n$-value must be integer, the error predictor equation is modified accordingly and approximated as:

$$p = \begin{cases} 0 & \text{for } n \leq 11 \\ 0.93(n) - 10.56 & \text{for } n > 12 \end{cases}$$
(16)
5 Conclusions

A new heuristic algorithm for solving the TSP has been developed and tested. The algorithm can be implemented with only a basic knowledge of trigonometry, Excel and Excel macros or VBA programming. The algorithm performs well for small problems with the error increasing linearly with the problem size. This decrease in accuracy is caused by the sort solution methodology which naturally searches for the outside perimeter defined on the graph of the nodes for all values of φ which can lead to excessive back and forth, zigzag travel in larger problems. Accuracy could likely be increased by dividing the complete set of nodes into more than two sets, allowing the sort procedure to delve more deeply into the interior region of the node graph with less overall travel within each set. It is expected that the error predictor equation (16) would hold within each pair of sets, even as the overall number of nodes in the instance increases. Connecting the pairs of sets into a complete tour would likely contribute additional error. This is a topic for future research. Another avenue for future research could address more thoroughly the question of how to handle an odd number of nodes, especially when more than two sets are defined. Additional rules for determining the incremental rotation angles could also be investigated. The general solution methodology might also be modified to work for three-dimensional or spherical coordinate problems as long as the node coordinates are specified.

6 References

[12] TSPLIB. URL: <http://comopt.uni-heidelberg.de/software/TSPLIB95/>Last access; December 7, 2011.