Heuristic Methods for TSP

Chan-Ching Hsu
cchsu@iastate.edu

1 Problem

The traveling salesman problem (TSP) asks the following question: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the original city? It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.
Various heuristics and approximation algorithms, which quickly yield good solutions have been devised. Modern methods can find solutions for extremely large problems (millions of cities) within a reasonable time which are with a high probability just 2-3% away from the optimal solution [1].

2 Implementation

The nearest neighbor algorithm was one of the first algorithms used to determine a solution to the traveling salesman problem. In it, the salesman starts at some city and then visits the city nearest to the starting city. From there he visits the nearest city that was not visited so far, etc., until all cities are visited, and the salesman returns to the start. The algorithm is described as below.

- Nearest Neighbor Algorithm

```
1. $V = \{1, \dots, n-1\}$    * Vertices except for 0.
2. $U = \{0\}$                * Vertex 0.
3. while $V$ not empty
4.     $u$ = most recently added vertex to $U$
5.     Find vertex $v$ in $V$ closet to $u$
6.     Add $v$ to $U$ and remove $v$ from $V$.
7. endwhile
8. Ourput vertices in the order they were added to $U$
```

Though usually rather bad, nearest neighbor tours only contain a few mistakes, but at the same time contain long segments connecting nodes with short edges. Therefore, such tours can serve as good starting tours for improvement methods.
The tours computed by the various construction heuristics initially are only of moderate quality. In general, improvement heuristics are characterized by a certain type of pf basic move to alter the current tour.
A 2-opt move consists of eliminating two edges and reconnecting the two resulting paths in a different way to obtain a new tour. There is only one way to reconnect the paths that yield a

different tour. Among all pairs of edges whose 2-opt exchange decreases the length we choose the pair that gives the shortest tour. This procedure is then iterated until no such pair of edges is found. Two types of 2-opt heuristics are described as follows.

- 2-OPT Heuristic

```
 1. $T$ = some starting tour
 2. noChange=true
 3. **repeat**
 4.   **foe all** possible edge-pairs in $T$
 5.     $T'$ = tour by swapping end points in edge-pair
 6.     **if** $T' < T$
 7.       $T = T'$
 8.       noChange=false
 9.       **break**   * *Quit loop as soon as improvement is found*
10.     **endif**
11.   **endfor**
12. **until** noChange
13. **return** $T$
```

- An Alternative: Find Best Tour with all Possible Swaps

```
 1. $T$ = some starting tour
 2. noChange=true
 3. **repeat**
 4. $T_{best} = T$
 5.   **foe all** possible edge-pairs in $T$
 6.     $T'$ = tour by swapping end points in edge-pair
 7.     **if** $T' < T_{best}$
 8.       $T_{best} = T'$
 9.       noChange=false
10.     **endif**
11.   **endfor**
12.   $T = T_{best}$
13. **until** noChange
14. **return** $T$
```

Nearest neighbor algorithm is implemented to produce a initial tour and then the tour is improved by implementing the alternative 2-opt heuristic algorithm , which consider all possible swaps to find a better tour. Initial tours from my work were actually selected before performing 2-switches methods. All the possible initial tours were generated by choosing various starting vertex (first node added into U). The total length of each initial tour was computed and then compared; the shortest tour would be selected to conduct the local heuristic.
All the algorithm were developed in C Programming Language [4] and run on a ISU Engineering Linux server [3]. To plot tours in the solutions, I used gnuplot [2], which is a free, command-drive, interactive and data plotting program. As it is quick to implement the idea of the nearest neighbour algorithm, the 2-opt algorithm took me a long time to develop codes and verify the correctness. The most tricky part, I would say, is that one has to avoid accidently yield cycles after swapping; in addition, with the action of swapping, if the direction of the tour matters, the directions between effected vertices may need to be reversed accordingly.

3 Solutions

This section shows fundamentally the two results for each given instance, initial solutions and improved solutions.

3.1 Instance I1

With the solution from initial algorithm, I obtained the best initial tour and this tour doesn't have edges to swap in order to decrease the length. The best solution I have is illustrated in Fig. 1.

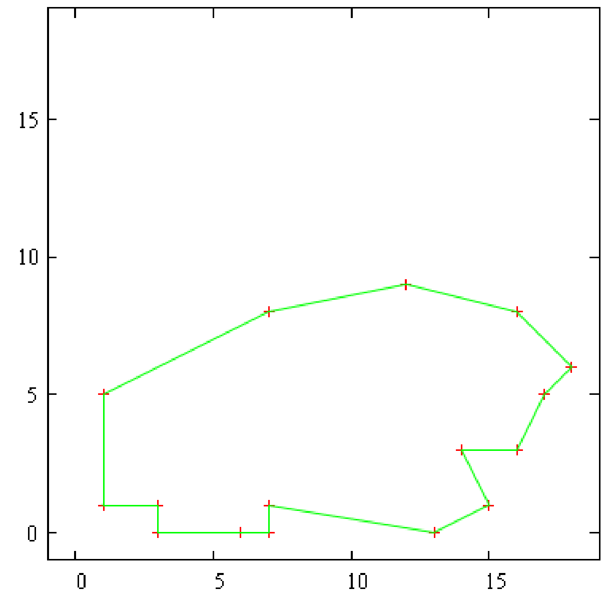
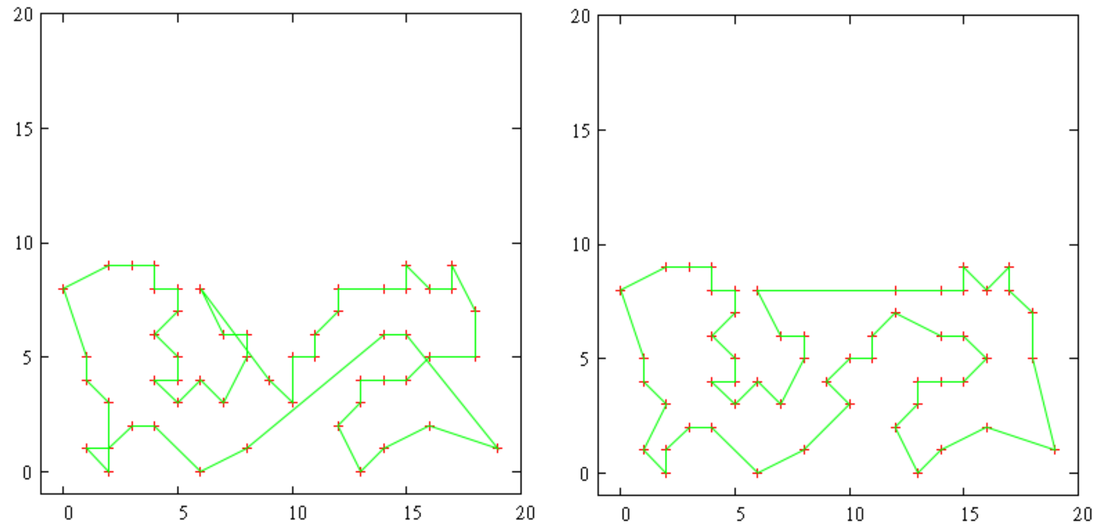


Figure 1: Initial Result no Improvement to made

3.2 Instance I2

Fig. 2(a) gives the best initial tour and Fig. 2(b) shows the best improved tour by 2-opt moves.

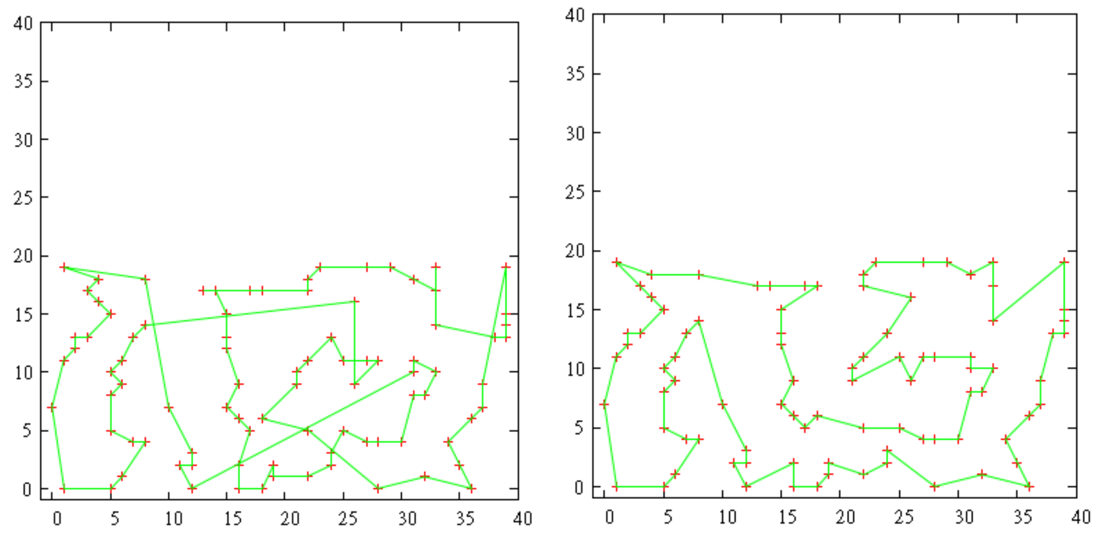


(a) Initial Result (b) Local Optimization

Figure 2: The initial output and improved solution to I2

3.3 Instance I3

Fig. 3(a) gives the best initial tour and Fig. 3(b) shows the best improved tour by 2-opt moves.

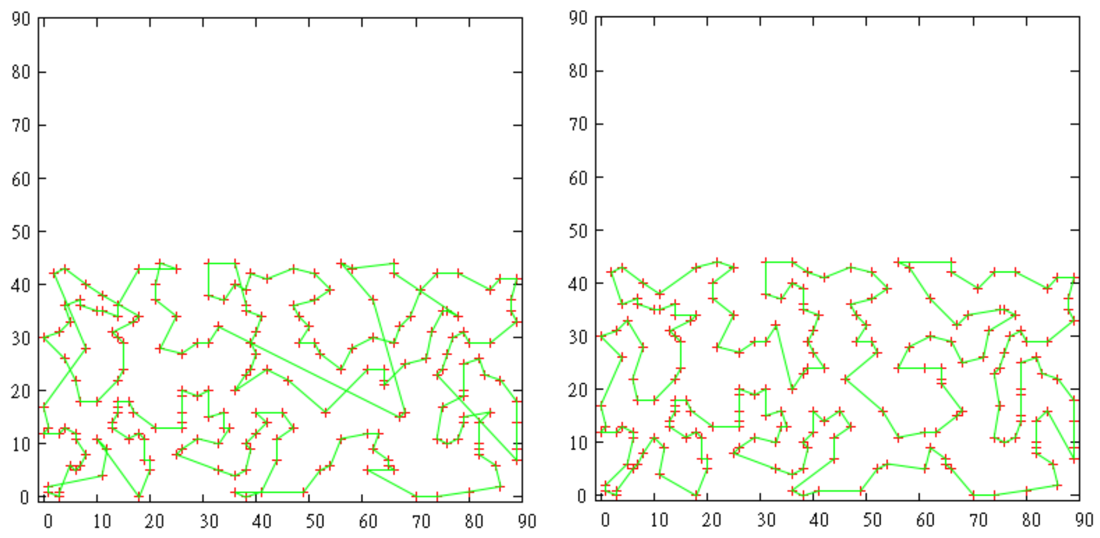


(a) Initial Result (b) Local Optimization

Figure 3: The initial output and improved solution to I3

3.4 Instance I4

Fig. 4(a) gives the best initial tour and Fig. 4(b) shows the best improved tour by 2-opt moves.



(a) Initial Result (b) Local Optimization

Figure 4: The initial output and improved solution to I4

References

[1] "Traveling Salesman problem heuristics: leading methods, implementations and latest advances", `http://www.sciencedirect.com/science/article/pii/S0377221710006065`

[2] gnuplot, `http://www.gnuplot.info/`

[3] Linux Servers in College of Engineering, Iowa State University, `http://it.engineering.iastate.edu/remote/`

[4] C Programming Language, *Wikipedia*, `http://en.wikipedia.org/wiki/C_(programming_language)`