ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/313898510

An Empirical Study of the Multi-fragment Tour Construction Algorithm for the Travelling Salesman Problem

Conference Paper *in* Advances in Intelligent Systems and Computing · November 2016

DOI: 10.1007/978-3-319-52941-7_28

CITATION
1

READS
1,891

3 authors:

Mehdi EL KRARI
University of Nottingham
11 PUBLICATIONS 32 CITATIONS

SEE PROFILE

Belaïd Ahiod
Mohammed V University of Rabat
43 PUBLICATIONS 956 CITATIONS

SEE PROFILE

Youssef B. el Benani
Mohammed V University of Rabat
23 PUBLICATIONS 184 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Developing and Adapting metaheuristics for Combinatorial Optimization Problems related to logistics, transportation, and other routing problems. View project

Project Nature-inspired metaheuristic algorithms for the Cell Formation Problem View project

All content following this page was uploaded by Mehdi EL KRARI on 26 April 2020.

The user has requested enhancement of the downloaded file.

An Empirical Study of The Multi-Fragment Tour Construction Algorithm for The Travelling Salesman Problem

Mehdi El Krari[1,2], Belaïd Ahiod[1,3], and Bouazza El Benani[1,2]

[1] Faculty of Science, Mohammed V University in Rabat
Rabat, Morocco
[2] Computer Science Laboratory
mehdi@elkrari.com, elbenani@hotmail.com
[3] LRIT, associated unit to CNRST (URAC 29)
ahiod@fsr.ac.ma

Abstract. This paper proposes a detailed study of the Multi-Fragment (MF) tour construction (TC) algorithm for the Travelling Salesman Problem (TSP). This TC heuristic is based on an edge selection strategy which favours edges with the smallest cost under the constraint to have a feasible tour. Extensive computational experiments have been performed on benchmark instances from the literature. The results show that the studied algorithm generally outperforms other constructive heuristics for the TSP.

Keywords: Travelling Salesman Problem, Edge Selection Strategy, Tour Construction, Heuristic.

1 Introduction

The Travelling Salesman Problem (TSP) is one of the most studied problems in combinatorial optimisation [15, 19]. It was first introduced by William Rowan Hamilton (1859). Formally, the problem can be defined as follows. Given a finite number of cities and the cost of travel between each pair of them, find the cheapest way of visiting all of the cities and returning to the starting point. It is well-known that the TSP is NP-hard [7]. Since only relatively small instances of the TSP can be solved to optimality within a reasonable amount of time. The problem has been mainly tackled using heuristic approaches, including constructive heuristics, local search heuristics, and metaheuristics [16, 11, 6].

There are several practical contexts where the TSP is a subproblem as in logistics, transportation of goods, shipping, and many other scheduling problems. The manufacturing of VLSI chips and X-ray crystallography are just a few examples of several issues in the industry that are modelled as a travelling salesman problem. In addition to its practical importance, the TSP is also a standard testbed for new algorithmic ideas.

As mentioned earlier, several heuristics have been developed to solve the TSP. These include, among others, k-opt [6], genetic algorithms [8], tabu search

[13], simulated annealing [10] and Lin-Kernighan [11]. These heuristics need a starting tour to work on and improve it. They are called tour improvement (TI) heuristics. This tour can be constructed either randomly, by choosing a city arbitrarily at each iteration, or by carrying out so-called tour construction (TC) heuristics, which builds tours according to specific criteria. This paper gives a detailed study of the greedy tour construction algorithm[11], also called the Multi-Fragment (MF) heuristic [3, 2], which is based on an edge selection strategy. The purpose is to build a tour with the smallest edges which are already sorted, under the constraint that the tour must be feasible, which makes the MF heuristic different from most known TC methods.

The remainder of this paper is organised as follows: Section 2 gives a brief overview of some tour construction heuristics proposed in the literature. Section 3 describes The MF heuristic. The computational results on a large set of standard instances from TSPLIB are presented and discussed in Section 4. Finally, we draw some conclusions in Section 5.

2 Tour construction: state of the art

A TC heuristic [12] has a well-defined rule(s) to construct a feasible tour but makes no improvement once the tour is constructed. In other words, the tour is iteratively built without making changes on parts already built. In what follows, we cite some TC methods frequently used in the literature.

2.1 Nearest Neighbor heuristic (NN)

The nearest neighbor heuristic (NN) [2] is the simplest heuristic dedicated to the TSP. The salesman starts from a city chosen arbitrarily, and goes to the nearest one and so on so forth. Making sure that none of the cities already visited is revisited. When all the n (which is the instance size) cities are visited, the salesman returns to the starting city. The procedure is summarized in Algorithm 1 below.

Algorithm 1 Nearest Neighbor heuristic

```
Select randomly a city (c)
while visited cities are less than n do
    Find nearest city (c_n) to (c) from unvisited cities
    c ← c_n
end while
```

NN's complexity is $O(n^2)$. For each starting city, it provides exactly one tour. Therefore, at most n possible tours can be constructed for an instance of size n, as two tours with two different starting cities may be similar.

2.2 Insertion heuristic (Ins)

The insertion heuristic [2] is based on the insertion of a city in a partial tour (V). There are several rules and criteria for insertion, the simplest one is to select a city among those unvisited (U) and insert it next to the nearest (resp. farthest) city from those existing in V (**Nearest** (resp. **Farthest**) **Insertion**). Once inserted, the selected city is removed from U. Another approach is to insert the city in V so as to minimize the cost of the new sub-tour (**Smallest Sum Insertion**). The process is repeated until U is empty.

Algorithm 2 Insertion heuristic

Build a partial tour (V) from two cities
while U is not null **do**
  Insert a city (c) from U in the tour V according to the chosen rule
  $U \leftarrow U \backslash \{c\}$
end while

There are $(n \times (n-1)/2)$ pairs of cities (edges) to start the construction of the tour. The number of tours provided by the insertion heuristic depends on the selection rule used. For example, if the nearest insertion criterion is used, the insertion heuristic gives 2^{n-2} tours for each couple chosen. While it gives n tours for each starting edge if the Smallest Sum insertion is used. Whatever the rule is, the insertion heuristic's complexity is $O(n^2)$.

2.3 Tour construction with spanning trees

The TSP can be interpreted as a weighted graph $G = (V, E, M)$, where

- V is a set of vertices
- E is a set of edges $\{v, w\}$ where $v, w \in V$
- M is a map from edges to values, which are positive numbers (distances) in the case of TSP.

This TC heuristic is based on the minimum spanning tree (MST) (a subtree $S_G = (V, E')$ (acyclic subgraph) of G, where $E' \subseteq E$) of an instance of TSP [9]. Once the tree is found, the salesman goes around its outside, starting at any node and not yet visited cities along the tree, until he comes back to the starting city. Figure 1 gives an example of visiting cities through a spanning tree: Considering the node (1) as a starting city, the salesman goes around the tree and visits the node (2) then (3). By continuing the tour, the salesman will not visit the node (2) since it has been visited previously and will visit node (4) then (5), and so on.

There are several algorithms to find the MST of a graph (and therefore an instance of TSP):

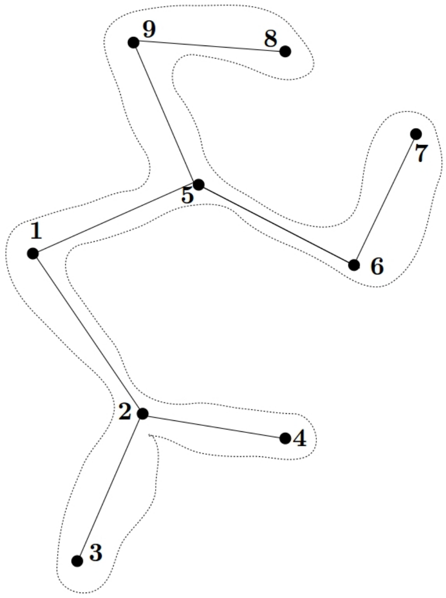

Fig. 1. Going around the Minimum Spanning Tree

- The first algorithm was proposed by Borůvka [4] (1926) . It is based on merging disjoint trees. At the beginning of the procedure, each vertex is considered as a separate tree. In each step, the algorithm merges every component with some other using strictly the cheapest outgoing edge of the given component.
- Kruskal's algorithm [14] (1956) creates a forest of trees which consists of n single node trees without edges. At each step, we add the cheapest edge so that it joins two trees together. If it were to form a cycle, it would simply link two nodes that were already part of the same tree, so that this edge would not be needed.
- Prim's algorithm [17] (1956) starts with one node. Then, it sequentially adds a node that is unconnected to the new graph, each time selecting the node whose connecting edge has the smallest weight out of the available nodes' connecting edges.

3 Multi-Fragment heuristic

The MF heuristic [3, 2] is an interesting approach that considers the edges as the main parameter. The idea is to select edges accordingly to their respective costs. Since the TSP's objective is to find a tour whose cost is minimal, this heuristic aims to select the edges as minimal as possible. Then, the edges are sorted. It is almost sure that the n smaller edges will not give a Hamiltonian feasible tour, and through all the cities of the problem. This heuristic builds a tour by connecting edges while avoiding to:

- Close a tour while unvisited cities exist (see Fig 2).
- Add an edge of which one of the two cities is already linked to two on the under construction tour. For example, in the scenario in Figure 3, the algorithm will not choose the edge {b,e} because the city 'b' cannot be linked to more than two cities in a tour.

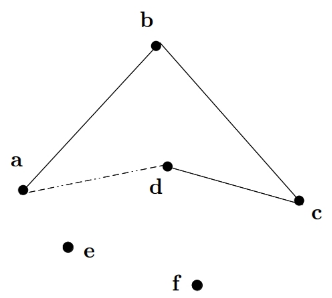

Fig. 2. Closing a tour smaller than n

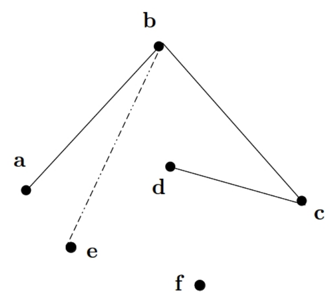

Fig. 3. Linking a city to an already connected city

MF is a deterministic heuristic. For any instance of the TSP, the tour built by this TC heuristic is unique since we choose the first edge found if many exist with the same cost. The results given in the next section show that MF heuristic often goes through a large number of edges to build the appropriate tour. Browsing all edges of the problem makes the complexity of the MF heuristic equal to $O(n^2)$. A pseudo-code of MF heuristic is given in Algorithm 3, while Figure 4 shows an example of constructing a solution (tour) by MF heuristic, for a TSP instance of 5 cities described by the complete graph in the right. Each row is an iteration of MF heuristic; the first column displays the selected edge. The second column shows the steps of the tour construction: the cities separated by a hyphen are part of the same section, while the pipe separates two sections not yet connected. The last column shows why the edge was rejected.

edges	tour	
1,3	1-3	
2,5	1-3 | 2-5	
2,4	1-3 | 4-2-5	
4,5	1-3 | 4-2-5	(4,5) is closing a tour: rejected
1,2	1-3 | 4-2-5	(2) connected to (4) & (5) : (1,2) rejected
1,4	3-1-4-2-5	

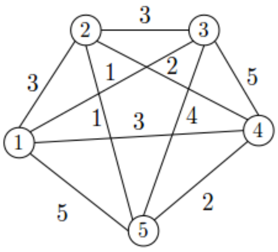

Fig. 4. Example of a tour construction using MF heuristic for an instance of 5 cities

Algorithm 3 MF heuristic

Require: Sorted set of all edges of the problem E.
Ensure: A tour T.

for each e in E **do**
  if (e is closing T and $size(T) < n$) **or** (e has a city already connected to two others) **then**
    go to the next edge
  end if
  if e is closing T and $size(T) = n$ **then**
    add e to T
    return T
  end if
  add e to T
end for
return T

4 Experimental results

We tested MF heuristic on 33 instances of the TSPLIB [18, 20], whose size ranges between 51 and 5934. First, we present a detailed MF heuristic performances. Then, we compare these results to those obtained with other TC methods.

MF heuristic is coded in Java 1.7, All tests were run on an Intel(R) Core(TM) i3-4150 CPU @ 3.50GHz and 4 GB of memory.

4.1 MF heuristic's performances

Since MF heuristic gives the same permutation for each run, each instance was solved only once. Table 1 below gives some statistics about MF heuristic results on different instances from the TSPLIB benchmark: each line gives the "Instance" name; its "Size"; the fitness of the Best Known Solution (BKS) (taken from [20]); the fitness value of the tour constructed — *Cost*; the gap of obtained tour from the BKS with a value from 0% (the best solution is found in this case)

— δ (equation 1), CPU time needed to build each tour in seconds, and finally the ratio of edges browsed – from all possible edges – before closing the tour with a value up to 100% (meaning that all edges have been checked)— BE (equation 2).

$$\delta = 100 \times (C - BKS)/BKS[\%] \tag{1}$$

$$BE = 100 \times BrowsedEdges/(n \times (n-1)/2)[\%] \tag{2}$$

Table 1: MF heuristic's performances

Instance	Size	BKS	*Cost*	δ (%)	CPU (s)	BE (%)
eil51	51	426	531	24.65	0.14	95
berlin52	52	7542	9951	31.94	0.14	100
st70	70	675	750	11.11	0.14	62
eil76	76	538	585	8.74	0.15	68
pr76	76	108159	147496	36.37	0.14	99
rat99	99	1211	1440	18.91	0.15	76
kroA100	100	21282	24287	14.12	0.15	54
kroB100	100	22141	25813	16.58	0.15	83
kroC100	100	20749	23295	12.27	0.14	59
kroD100	100	21294	24448	14.81	0.15	56
kroE100	100	22068	24846	12.59	0.15	45
rd100	100	7910	9253	16.98	0.14	95
eil101	101	629	783	24.48	0.15	99
lin105	105	14379	16766	16.60	0.15	56
pr107	107	44303	47545	7.32	0.15	59
pr124	124	59030	64998	10.11	0.15	50
bier127	127	118282	141339	19.49	0.16	99
ch130	130	6110	7223	18.22	0.15	93
pr136	136	96772	116048	19.92	0.15	58
pr144	144	58537	65844	12.48	0.15	89
ch150	150	6528	7809	19.62	0.15	90
kroA150	150	26524	31892	20.24	0.15	88
kroB150	150	26130	31427	20.27	0.15	62
pr152	152	73682	85420	15.93	0.15	99
u159	159	42080	49589	17.84	0.15	74
rat195	195	2323	2648	13.99	0.16	91
d198	198	15780	19147	21.34	0.17	99
kroA200	200	29368	34554	17.66	0.15	65
kroB200	200	29437	35975	22.21	0.17	97
ts225	225	126643	133460	5.38	0.15	18
gil262	262	2378	2739	15.18	0.17	68
a280	280	2579	3089	19.78	0.16	99
lin318	318	42029	49898	18.72	0.16	82

Table 1: MF heuristic's performances

Instance	Size	BKS	*Cost*	δ (%)	CPU (s)	BE (%)
rd400	400	15281	17272	13.03	0.17	83
fl417	417	11861	12931	9.02	0.16	46
pcb442	442	50778	61076	20.28	0.17	99
d493	493	35002	41362	18.17	0.18	100
u574	574	36905	45043	22.05	0.18	99
rat575	575	6773	7858	16.02	0.17	62
d657	657	48912	56612	15.74	0.20	99
u724	724	41910	49142	17.26	0.20	91
rat783	783	8806	10288	16.83	0.21	86
u1060	1060	224094	258947	15.55	0.23	66
vm1084	1084	239297	293277	22.56	0.29	99
pcb1173	1173	56892	70060	23.15	0.31	99
d1291	1291	50801	61253	20.57	0.30	100
rl1304	1304	252948	285123	12.72	0.25	49
rl1323	1323	270199	307420	13.78	0.33	90
nrw1379	1379	56638	66106	16.72	0.32	77
fl1400	1400	20127	24709	22.77	0.29	64
u1432	1432	152970	182229	19.13	0.29	99
fl1577	1577	22249	25221	13.36	0.31	97
d1655	1655	62128	72498	16.69	0.36	99
vm1748	1748	336556	397399	18.08	0.34	93
u1817	1817	57201	65361	14.27	0.32	72
rl1889	1889	316536	379953	20.03	0.40	96
d2103	2103	80450	95124	18.24	0.37	100
u2152	2152	64253	73785	14.84	0.34	61
u2319	2319	234256	260659	11.27	0.34	84
pr2392	2392	378032	453323	19.92	0.46	99
pcb3038	3038	137694	161399	17.22	0.56	94
fl3795	3795	28772	32475	12.87	0.63	96
fnl4461	4461	182566	210768	15.45	0.88	94
rl5915	5915	565530	638931	12.98	1.42	98
rl5934	5934	556045	634314	14.08	1.38	93

The results in Table 1 indicate the good performance of the MF heuristic, the average gap over all of them is only 17.08%. As mentioned in the previous section, MF heuristic browses majority of edges to build the tour, 81% (as an average) of edges are browsed to get a feasible tour. Instances with a low BE are of better quality because the tour is composed of edges with the smallest cost. The figure 5 below gives a correlation between the gap from the BKS and browsed edges for the listed instances above. It shows that the more edges MF heuristic browses, the larger the value of the gap is.

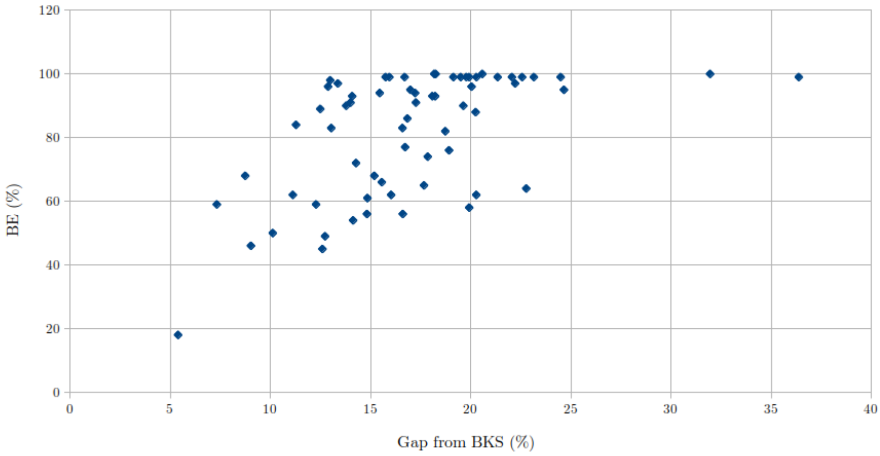

Fig. 5. Correlation between the gap from the BKS and browsed edges of each instance

4.2 Comparing MF heuristic with other TC methods

Table 2 below compares MF heuristic and the following TC methods, which has been described in Section 2: a) the Farthest Insertion (FI); b) the Nearest Neighbour (NN); c) Borůvka's algorithm (Bor); d) the Quick-Borůvka algorithm (Q-Bor). Since these methods can provide different solutions, each of them has been run 20 times with different starting cities/edges. We report the average gap (δ), as defined previously, average over the 20 runs, and the average CPU time in seconds ($CPU(s)$). Recall that, as mentioned previously, when using the MF heuristic, the instances were solved only once. Note also that while MF heuristic is coded in Java, the other methods used in the comparison are from the Concorde framework [1] and thus written in C, lighter than Java [5]. Running CPU time is given only for information.

Table 2: Comparing MF heuristic to other TC methods

Instance	Size	BKS	FI ($\delta, CPU(s)$)		NN ($\delta, CPU(s)$)		Q-Bor ($\delta, CPU(s)$)		Bor ($\delta, CPU(s)$)		MF($\delta, CPU(s)$)	
eil51	51	426	26.12	0.06	27.76	0.05	**12.68**	0.06	27.56	0.06	24.65	0.07
berlin52	52	7542	27.60	0.06	**21.43**	0.05	26.35	0.06	34.14	0.06	31.94	0.05
st70	70	675	27.96	0.06	22.83	0.05	23.35	0.06	12.15	0.06	**11.11**	0.05
eil76	76	538	26.44	0.06	22.68	0.05	12.08	0.06	11.05	0.06	**8.74**	0.05
pr76	76	108159	31.03	0.06	35.12	0.05	**17.55**	0.06	29.97	0.06	36.37	0.06
rat99	99	1211	33.49	0.06	25.59	0.05	20.71	0.06	20.70	0.06	**18.91**	0.05
kroA100	100	21282	26.85	0.06	26.17	0.05	28.47	0.06	19.57	0.06	**14.12**	0.06
kroB100	100	22141	26.01	0.06	26.28	0.05	20.98	0.06	17.12	0.06	**16.58**	0.05
kroC100	100	20749	30.53	0.06	24.77	0.05	**10.15**	0.06	19.41	0.06	12.27	0.05
kroD100	100	21294	27.93	0.06	29.38	0.05	24.75	0.06	28.17	0.06	**14.81**	0.05
kroE100	100	22068	25.51	0.06	24.26	0.05	**9.73**	0.06	11.27	0.06	12.59	0.05
rd100	100	7910	27.23	0.06	26.96	0.05	**14.69**	0.06	20.33	0.06	16.98	0.05
eil101	101	629	28.39	0.06	30.17	0.06	19.31	0.06	**15.76**	0.06	24.48	0.05

Table 2: Comparing MF heuristic to other TC methods

Instance	Size	BKS	FI ($\delta, CPU(s)$)		NN ($\delta, CPU(s)$)		Q-Bor ($\delta, CPU(s)$)		Bor ($\delta, CPU(s)$)		MF($\delta, CPU(s)$)	
lin105	105	14379	25.85	0.06	29.18	0.05	22.28	0.06	**14.40**	0.06	16.60	0.06
pr107	107	44303	**5.18**	0.06	14.63	0.06	20.46	0.06	7.15	0.06	7.32	0.05
pr124	124	59030	22.82	0.07	18.90	0.06	36.44	0.07	19.31	0.06	**10.11**	0.05
bier127	127	118282	26.58	0.07	23.28	0.06	**8.99**	0.07	17.69	0.07	19.49	0.06
ch130	130	6110	26.61	0.07	26.75	0.06	19.20	0.07	**12.29**	0.07	18.22	0.06
pr136	136	96772	22.18	0.07	24.14	0.06	21.99	0.07	**19.92**	0.07	**19.92**	0.06
pr144	144	58537	10.17	0.07	**9.68**	0.06	12.22	0.06	14.07	0.07	12.48	0.06
ch150	150	6528	24.86	0.07	20.13	0.06	25.87	0.06	22.43	0.07	**19.62**	0.06
kroA150	150	26524	27.20	0.06	26.30	0.06	26.94	0.06	21.65	0.06	**20.24**	0.06
kroB150	150	26130	29.14	0.07	32.28	0.06	**14.99**	0.07	15.78	0.07	20.27	0.05
pr152	152	73682	15.09	0.06	17.75	0.06	26.26	0.06	**10.11**	0.06	15.93	0.06
u159	159	42080	36.15	0.06	30.85	0.06	20.27	0.06	**14.42**	0.06	17.84	0.06
rat195	195	2323	35.25	0.06	19.92	0.06	17.32	0.06	19.40	0.06	**13.99**	0.06
d198	198	15780	24.01	0.06	21.54	0.06	**16.27**	0.06	16.43	0.06	21.34	0.06
kroA200	200	29368	35.48	0.06	27.72	0.06	20.89	0.06	18.53	0.06	**17.66**	0.06
%textbfkroB200	200	29437	27.54	0.06	26.30	0.06	18.64	0.06	**14.68**	0.06	22.21	0.06
ts225	225	126643	37.16	0.07	15.78	0.06	19.37	0.07	11.14	0.07	**5.38**	0.06
gil262	262	2378	31.96	0.06	23.57	0.06	20.15	0.06	15.33	0.06	**15.18**	0.06
a280	280	2579	36.85	0.06	25.64	0.06	20.94	0.06	22.93	0.06	**19.78**	0.06
lin318	318	42029	33.40	0.06	25.48	0.06	28.70	0.06	**17.34**	0.06	18.72	0.06
rd400	400	15281	34.90	0.07	25.98	0.06	18.05	0.06	22.48	0.06	**13.03**	0.07
fl417	417	11861	31.72	0.06	30.79	0.06	37.13	0.06	23.85	0.06	**9.02**	0.07
pcb442	442	50778	39.32	0.06	24.75	0.06	21.52	0.06	20.39	0.06	**20.28**	0.07
d493	493	35002	32.15	0.06	24.49	0.06	**14.64**	0.06	17.53	0.06	18.17	0.08
u574	574	36905	37.45	0.06	30.87	0.06	**20.95**	0.06	24.32	0.06	22.05	0.07
rat575	575	6773	36.57	0.06	26.22	0.06	18.22	0.06	17.26	0.06	**16.02**	0.07
d657	657	48912	38.55	0.06	30.84	0.06	17.30	0.06	19.57	0.06	**15.74**	0.08
u724	724	41910	37.12	0.06	26.07	0.06	17.78	0.06	18.91	0.06	**17.26**	0.09
rat783	783	8806	37.50	0.06	26.50	0.06	19.11	0.06	18.05	0.06	**16.83**	0.10
u1060	1060	224094	38.86	0.06	32.34	0.06	17.82	0.06	17.41	0.06	**15.55**	0.11
vm1084	1084	239297	37.44	0.06	24.87	0.06	22.41	0.06	**22.33**	0.06	22.56	0.12
pcb1173	1173	56892	43.57	0.06	28.67	0.06	18.53	0.06	**18.33**	0.06	23.15	0.12
d1291	1291	50801	49.79	0.06	22.23	0.06	17.76	0.06	**17.69**	0.06	20.57	0.15
rl1304	1304	252948	46.85	0.06	27.68	0.06	19.00	0.06	14.69	0.06	**12.72**	0.11
rl1323	1323	270199	45.87	0.06	22.70	0.06	21.32	0.06	16.64	0.06	**13.78**	0.12
nrw1379	1379	56638	39.09	0.06	24.44	0.06	16.67	0.06	**15.22**	0.06	16.72	0.14
fl1400	1400	20127	28.35	0.06	35.54	0.06	31.98	0.06	24.03	0.06	**22.77**	0.12
u1432	1432	152970	42.42	0.06	27.04	0.06	18.57	0.06	**18.05**	0.06	19.13	0.15
fl1577	1577	22249	38.17	0.06	24.15	0.06	30.16	0.06	17.95	0.06	**13.36**	0.16
d1655	1655	62128	45.10	0.06	22.36	0.06	19.15	0.06	**15.19**	0.06	16.69	0.14
vm1748	1748	336556	40.13	0.06	27.39	0.07	**17.83**	0.06	19.55	0.06	18.08	0.14
u1817	1817	57201	47.00	0.06	24.01	0.07	16.95	0.06	15.43	0.06	**14.27**	0.17
rl1889	1889	316536	46.14	0.06	24.95	0.07	19.23	0.06	**15.95**	0.06	20.03	0.17
d2103	2103	80450	52.56	0.06	13.68	0.07	**8.52**	0.06	12.97	0.06	18.24	0.18
u2152	2152	64253	47.66	0.06	22.98	0.07	18.61	0.06	17.39	0.06	**14.84**	0.15
u2319	2319	234256	35.85	0.06	20.49	0.07	12.98	0.06	15.05	0.06	**11.27**	0.18
pr2392	2392	378032	44.64	0.06	25.69	0.07	19.38	0.06	**19.02**	0.06	19.92	0.17

Table 2: Comparing MF heuristic to other TC methods

Instance	Size	BKS	FI ($\delta, CPU(s)$)		NN ($\delta, CPU(s)$)		Q-Bor ($\delta, CPU(s)$)		Bor ($\delta, CPU(s)$)		MF($\delta, CPU(s)$)	
pcb3038	3038	137694	43.69	0.07	25.87	0.09	18.36	0.07	17.85	0.07	**17.22**	0.20
fl3795	3795	28772	45.02	0.07	29.34	0.10	27.62	0.07	19.02	0.07	**12.87**	0.44
fnl4461	4461	182566	40.79	0.07	24.39	0.11	**15.11**	0.07	15.54	0.07	15.45	0.44
rl5915	5915	565530	49.25	0.09	23.09	0.17	14.24	0.08	**12.52**	0.08	12.98	0.74
rl5934	5934	556045	51.61	0.07	22.33	0.13	15.11	0.07	**13.68**	0.07	14.08	0.66
Average			34.27	0.15	25.31	0.16	19.80	0.15	17.87	0.15	**17.08**	0.27

Results in Table 2 show that the MF heuristic is very competitive. For the 65 instances considered in this paper, MF heuristic gives the best tour in 31 instances, as opposed to Borůvka and Quick-Borůvka, which provide the best solutions only in 18 and 14 instances, respectively.

The TC heuristics based on edges are more effective than others which select nodes since the choice of each edge/distance affects directly the quality of the tour.

5 Conclusion

This paper proposed a detailed and empirical study of the MF heuristic designed to build a tour for the travelling salesman problem. This heuristic is based on the selection of edges in an ascending order to build the tour with smaller edges. The results of the MF heuristic on multiple instances of different size of TSP instances demonstrated its effectiveness.

Future research will be devoted to adapt this heuristic to solve other variants of the TSP such as the generalised and asymmetric TSP.

References

[1] Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Concorde: A code for solving traveling salesman problems (1999), `http://www.math.uwaterloo.ca/tsp/concorde.html`

[2] Bentley, J.J.: Fast algorithms for geometric traveling salesman problems. ORSA Journal on computing 4(4), 387–411 (1992)

[3] Bentley, J.L.: Experiments on traveling salesman heuristics. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 91–99. SODA '90, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1990), `http://dl.acm.org/citation.cfm?id=320176.320186`

[4] Borůvka, O.: O jistém problému minimálním (about a certain minimal problem) (in czech, german summary) (1926)

[5] Bull, J.M., Smith, L.A., Pottage, L., Freeman, R.: Benchmarking java against c and fortran for scientific applications. In: Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande. pp. 97–105. ACM (2001)

[6] Croes, G.A.: A method for solving traveling-salesman problems. Operations Research 6(6), 791–812 (dec 1958), `http://dx.doi.org/10.1287/opre.6.6.791`

[7] Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified np-complete problems. In: Proceedings of the Sixth Annual ACM Symposium on Theory of Computing. pp. 47–63. STOC '74, ACM, New York, NY, USA (1974), `http://doi.acm.org/10.1145/800119.803884`

[8] Grefenstette, J., Gopal, R., Rosmaita, B., Van Gucht, D.: Genetic algorithms for the traveling salesman problem. In: Proceedings of the first International Conference on Genetic Algorithms and their Applications. pp. 160–168. Lawrence Erlbaum, New Jersey (160-168) (1985)

[9] Held, M., Karp, R.M.: The traveling-salesman problem and minimum spanning trees. Operations Research 18(6), 1138–1162 (1970)

[10] Helsgaun, K.: An effective implementation of the lin–kernighan traveling salesman heuristic. European Journal of Operational Research 126(1), 106–130 (2000)

[11] Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: A case study in local optimization. Local search in combinatorial optimization 1, 215–310 (1997)

[12] Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the stsp. In: The traveling salesman problem and its variations, pp. 369–443. Springer (2007)

[13] Knox, J.: Tabu search performance on the symmetric traveling salesman problem. Computers & Operations Research 21(8), 867–876 (1994)

[14] Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical society 7(1), 48–50 (1956)

[15] Lawler, E.L., Lenstra, J.K., Kan, A.R., Shmoys, D.B.: The traveling salesman problem: a guided tour of combinatorial optimization, vol. 3. Wiley New York (1985)

[16] Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & Operations Research 24(11), 1097–1100 (1997)

[17] Prim, R.C.: Shortest connection networks and some generalizations. Bell system technical journal 36(6), 1389–1401 (1957)

[18] Reinelt, G.: TSPLIB—a traveling salesman problem library. ORSA Journal on Computing 3(4), 376–384 (nov 1991), `http://dx.doi.org/10.1287/ijoc.3.4.376`

[19] Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications. Springer-Verlag, Berlin, Heidelberg (1994)

[20] Reinhelt, G.: {TSPLIB}: a library of sample instances for the tsp (and related problems) from various sources and of various types. URL: http://comopt. ifi. uniheidelberg. de/software/TSPLIB95 (2014)

View publication stats