Experimental Analysis of New Heuristics for the TSP

Fidan Nuriyeva[1], Gozde Kizilates[2], Murat Ersen Berberler[3]
[1, 2]Ege University, Izmir, Turkey
[3]Dokuz Eylul University, Izmir, Turkey
[1]*nuriyevafidan@gmail.com*, [2]*gozde.kizilates@gmail.com*, [3]*murat.berberler@deu.edu.tr*

***Abstract*— In this study, the three new heuristic algorithms which are proposed in [1] for the solution of traveling salesman problem is developed. In addition, the new versions of 2-opt and 3-opt algorithms are proposed. These algorithms are tested and their performances are compared with the well-known heuristic algorithms such as Nearest Neighbor, and Greedy algorithms.**

Keywords— traveling salesman problem; heuristic algorithms; local search

I. INTRODUCTION

Traveling salesman problem (TSP) is one of the most attractive problems of combinatorial optimization due to its theoretical difficulty and the richness of its applications. Since it is an NP-hard class problem, many heuristic and meta-heuristic algorithms have been developed to solve it [2-7].

TSP aims to find a route to a salesman who starts from a home location, visits prescribed set of cities and returns to the original location in such a way that the total traveling distance will be minimum and each city will have been visited exactly once. There are many variations of TSP: Symmetric TSP, Asymmetric TSP, The MAX TSP, The MIN TSP, TSP with multiple visits (TSPM), TSP with a closed tour, TSP with an open tour [4].

Many algorithms have been developed for TSP. The algorithms for TSP can be classified as:

- Heuristic Algorithms (NN, Greedy and others);
- Metaheuristic Algorithms (Tabu Search, Ant Colony Optimization, Genetic Algorithms and others);
- Approximate Algorithms (Christofides and others);
- Exact Algorithms (Branch & Bound and others);

II. TOUR CONSTRUCTION HEURISTICS

Every TSP heuristic can be evaluated in terms of two key parameters: its running time and the quality of the tours that it produces. Because of space limitations, we shall concentrate here on the most important undominated heuristics, where a heuristic is undominated if there is no competing heuristic which find better tours and runs more faster.

In this study, we focus on only two of these algorithms: Nearest Neighbor, and Greedy Algorithms. It is because we compare three new algorithms that we propose with these two algorithms [2].

A. Nearest Neighbor

This is perhaps the simplest and most straightforward TSP heuristic. The key to this algorithm is to always visit the nearest city.

Nearest Neighbor,

1. Select a random city.
2. Find the nearest unvisited city and go there.
3. Are there any unvisited cities left? If yes, go to step 2.
4. Return to the first city.

We can obtain the best result out of this algorithm by starting the algorithm over again for each vertex and repeat it for *n* times.

B. Greedy Algorithm

The Greedy heuristic gradually constructs a tour by repeatedly selecting the shortest edge and adding it to the tour as long as it doesn't create a cycle with less than *N edges, or increase the degree of any node by* more than 2. We must not add the same edge twice of course.

Greedy,

1. Sort all edges.
2. Select the shortest edge and add it to our tour if it doesn't violate any of the above constraints.
3. Do we have *n edges in our tour? If no, go to* step 2

III. 2-OPT AND 3-OPT ALGORITHMS

Once a tour has been generated by some tour construction heuristic, we might wish to improve that solution. There are several ways to do this, but the most common ones are the 2-opt and 3-opt local searches. The 2-opt algorithm basically removes two edges from the tour, and reconnects these two paths which are formed by removing these two edges. There is only one way to reconnect the two paths so that we still have a valid tour (Figure 1, 2). We do this only if the new tour will be shorter. This process of removing and reconnecting the tour continues until no 2-opt improvement is found. The tour we obtain at the end of this process is now 2-optimal. The 3-opt algorithm works in a similar fashion, but instead of removing two edges we remove three. This means that we have two ways of reconnecting the three paths into a valid tour (Figure 3). A 3-opt move can actually be seen as two or three 2-opt moves. We finish our search when no more 3-opt moves can improve the tour. If a tour is 3-optimal it is also 2-optimal. If we look at the tour as a permutation of all the cities, a 2-opt move will result in reversing a segment of the permutation. A 3-opt move can be seen as two or three segment reversals [3].

A. 2-opt and Shifting

The 2-opt code in is as following:

```
for(i=1;i<=n-3;i++)
        for(j=i+2;j<=n-1;j++)
                if(d[a[i]][a[i+1]]+d[a[j]][a[j+1]]>
                   d[a[i]][a[j]]+d[a[i+1]][a[j+1]])

                {

                        c=a[i+1];
                        a[i+1]=a[j];
                        a[j]=c;
                }
```

The code above improves the solution by shifting the vertexes when their indexes in solution vector are proper, that is when their position are between the 1st and the $(n\text{-}1)^{th}$ vertex. If, however, the shifting vertex j is in the n^{th} position in the solution vector, then the *if* part of the algorithm does not run and the algorithm does improve anything. In order to prevent this handicap, we shift the elements of the solution vector 1 unit to the left. Therefore, since the n^{th} vertex is now $(n\text{-}1)^{th}$, the *if* part of the algorithm runs properly and the algorithm improves its solution by shifting these two vertexes. This whole process is explained in an example below:

For example, let us say that the optimum result of the traveling salesman problem is 6 and the solution vector is [v1v2v3v4v5v6](6). If we apply the 2-opt algorithm to a vector like [v1v2v3v4v6v5](10) whose tour cost is 10, then the algorithm will not make any shifts between the vertexes. Yet, if the elements of solution vector are switched 1 unit to the left, then the 2-opt algorithm will find the optimum result: [v3v4v5v6v1v2](6)

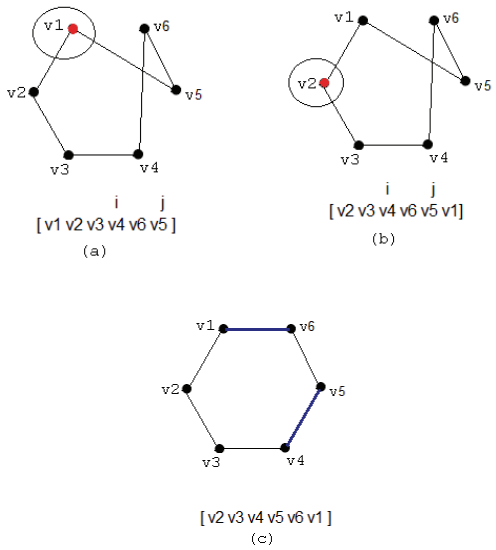

Figure 1. The demonstration of the shifting operation in a 2-opt application on an example

In the same way, if we apply the 2-opt algorithm to the vector, [v1v4v5v2v3v6](18), the algorithm improves this vector three times and the solution vector obtained at the end of the process is [v1v3v2v4v5v6](10). Even if the elements of this solution vector are shifted 1 unit to the left, there will be no improve. Yet if we apply the 2-opt algorithm after shifting the elements of this solution vector 2 units to the left, then we can obtain the optimum result for the problem [v1v2v3v4v5v6](6).

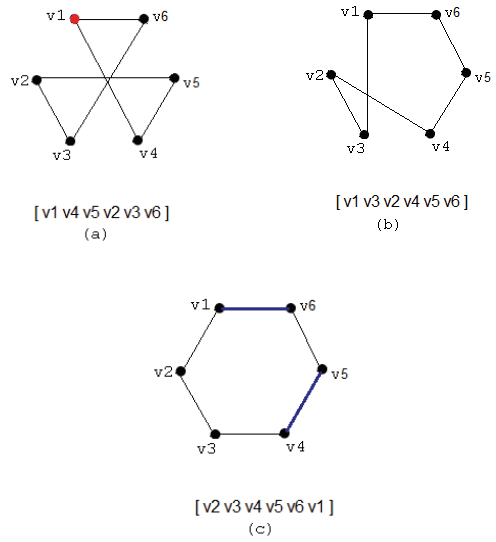

Figure 2. The demonstration of the double shifting operation in a 2-opt application on an example

B. 3-opt and Shifting

The 3-opt code is as following:

```
for(i=1;i<=n-5;i++)
  for(j=i+2;j<=n-3;j++)
     for(k=j+2;k<=n-1;k++)

if(d[a[i]][a[i+1]]+d[a[j]][a[j+1]]
 +d[a[k]][a[k+1]]>d[a[i]][a[j+1]]
 +d[a[j]][a[k+1]]+d[a[k]][a[i+1]]))
   {

        c=a[i+1];
        a[i+1]=a[j+1];
        a[j+1]=c;


        c=a[j];
        a[j]=a[k];
        a[k]=c;
  }
```

As in 2-opt algorithm, if the position of shifting vertex is n in the solution vector, then the algorithm will skip the *if* part and will not improve the solution. However, when the elements of the solution vector are shifted to the left, then we again obtain the optimum result. Also in this method, there are examples in which shifting the elements of the solution vector

1 unit to the left is not enough so there should be shifting more than 1 unit to the left.

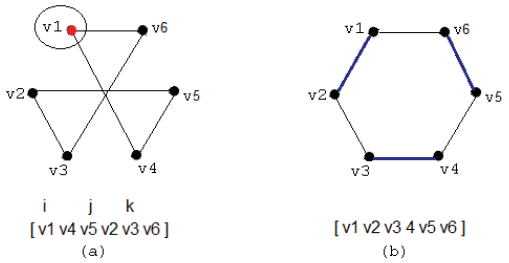

Figure 3. The demonstration of the shifting operation in a 3-opt application on an example

In this Table 1, the second column shows the results of 2-opt and 3-opt applications with shifting operation, and the third column shows the results of 2-opt and 3-opt applications without shifting operation. As it is seen in the table, the results of the applications with shifting operation give better results than the applications without shifting.

TABLE I. The comparison of the 2-opt and 3-opt applications with and without shifting operation

G	3-opt + 2-opt (S+)	3-opt + 2-opt (S-)
bayg29	9074.148	9324.444
att48	33607.682	34228.800
eil51	429.484	456.265
berlin52	7544.365	7993.064
gr96	516.870	529.997
kroA100	21285.443	22146.600
rd100	8101.042	8206.614
lin105	14382.995	14670.207
ch130	6250.213	6409.888
u159	43786.312	45147.919

IV. NEW HEURISTIC ALGORITHMS

In study [1], three new heuristic algorithms are proposed, their computer programs are developed and tested.

Algorithms are based on comparison of the maximum and minimum values of the distances between each city.

In these algorithms, the precedence was given to the worst vertexes in order to prevent the worst-case scenarios.

A. Algorithm 1 (Feinting)

This algorithm is about finding the maximum element for each row in the adjacent matrix. The algorithm continues to add to the tour the minimum distance of the row in which the maximum element exists. This process is applied to each row. The aim of the algorithm is to prevent the worst situations. The steps of this algorithm are as following:

1. Find the maximum distance for each row in adjacency matrix, and add it to MAX column.
2. Select the maximum distance in the MAX column.
3. In the same row in which this maximum distance exists, select the minimum distance which does not contain a sub tour and add it to the tour.
4. Increase the number of selected edges by 1.
5. If the number of selected edges is less than n then go to step 2.

We can demonstrate how the algorithm works in the following chart.

TABLE II. Table which shows how Algorithm 1 works

	c_1	c_2	c_3	...	c_l	...	c_n	MAX
c_1	0	a_{12}	a_{13}	...	...	...	a_{1n}	m_1
c_2	a_{21}	0	a_{23}	...	...	...	a_{2n}	m_2
c_3	a_{31}	a_{32}	0	...	...	...	a_{3n}	m_3
...	...	...	...	0	...	...	...	...
c_k	a_{k1}	a_{k2}	a_{k3}	...	a_{kl}	...	a_{kn}	m_k
...	...	...	...	...	...	0	...	...
c_n	a_{n1}	a_{n2}	a_{n3}	...	...	...	0	m_n

Here, $a_{ij} = \text{distance}(c_i, c_j)$, $m_s = \max_j \{a_{sj}\}$,

$m_k = \max_i \{m_i\}$, $a_{kl} = \min_j \{a_{kj}\}$, $s = \overline{1,n}$, $i, j = \overline{1,n}$

B. Algorithm 2 (The Most Advantageous Vertex)

This algorithm is about finding the maximum and minimum distances for each row in the adjacent matrix. The algorithm continues to find the difference between the maximum distance and the distances of the correspondent minimum column, and to add this difference to distance column. The steps of this algorithm are as following:

1. Find the maximum and minimum distances for each row in the adjacency matrix and add them to MAX and MIN columns.
2. Subtract the distances in MIN column from the correspondent distances in MAX column, and then add the result to DIFFERENCE column.
3. Find the maximum distance in DIFFERENCE column.
4. In the same row in which this maximum distance exists, select the minimum distance which does not contain a sub tour and add it to the tour.
5. Increase the number of selected edges by 1.
6. If the number of selected edges is less than n then go to step 3.

We can demonstrate how the algorithm works in the following chart.

TABLE III. Table which shows how Algorithm 2 works

	c_1	c_2	c_3	...	c_l	...	c_n	MAX	MIN	DIFFERENCE
c_1	0	a_{12}	a_{13}	...	...	...	a_{1n}	m_1	n_1	d_1
c_2	a_{21}	0	a_{23}	...	...	...	a_{2n}	m_2	n_2	d_2
c_3	a_{31}	a_{32}	0	...	...	...	a_{3n}	m_3	n_3	d_3
...	...	...	...	0	...	...	...	...	...	...
c_k	a_{k1}	a_{k2}	a_{k3}	...	a_{kl}	...	a_{kn}	m_k	n_k	d_k
...	...	...	...	...	...	0	...	...	...	...
c_n	a_{n1}	a_{n2}	a_{n3}	...	...	...	0	m_n	n_n	d_n

$$\text{Here, } a_{ij} = \text{distance}(c_i, c_j), \quad m_s = \max_j \{a_{sj}\},$$

$$n_s = \min_j \{a_{sj}\}, \quad d_s = m_s - n_s, \quad s = \overline{1,n}, \quad i, j = \overline{1,n},$$

$$d_k = \max_i \{d_i\}, \quad a_{kl} = \min_j \{a_{kj}\}$$

C. *Algorithm 3 (The Farthest Vertex)*

This algorithm is about finding the sums of each row in the adjacent matrix. The algorithm continues to add to the tour the minimum two distances of each row which includes the maximum distance. This process is applied to each row. The steps of this algorithm are as following:

1. Find the sums for each row in the adjacency matrix and add them to SUM column.
2. Find the maximum sum in SUM column.
3. In the same row in which this maximum sum exists, select the two minimum distances which do not contain a sub tour and add them to the tour.
4. Delete the row and column which correspondance to the maximum sum.
5. Increase the number of selected vertex by 1.
6. If the number of selected vertex is less than *n* then go to step 2.

We can demonstrate how the algorithm works in the following chart.

TABLE IV. Table which shows how Algorithm 3 works

	c_1	c_2	c_3	...	c_l	...	c_n	SUM
c_1	0	a_{12}	a_{13}	...	...	...	a_{1n}	s_1
c_2	a_{21}	0	a_{23}	...	...	...	a_{2n}	s_2
c_3	a_{31}	a_{32}	0	...	...	...	a_{3n}	s_3
...	...	...	...	0	...	...	...	...
c_k	a_{k1}	a_{k2}	a_{k3}	...	a_{kl}	...	a_{kn}	s_k
...	...	...	...	...	...	0	...	...
c_n	a_{n1}	a_{n2}	a_{n3}	...	...	...	0	s_n

$$\text{Here, } a_{ij} = \text{distance}(c_i, c_j), \quad s_i = \sum_{j=1}^{n} a_{ij}, \quad i = \overline{1,n},$$

$$s_k = \max_i \{s_i\}, \quad a_{kl} = \min_j \{a_{kj}\}$$

V. EXPERIMENTAL RESULTS

The sample problems used in these experiments are taken from the [7]. And the optimum solutions for each of these problems are taken from the [8] (Table V).

In Table V, * sign shows the optimum result, and ** sign shows the found result which is better than the optimum result.

In other words, in the problems which have ** sign, it is seen that the known optimum results are not actually the optimum ones.

TABLE V. Table which shows the experimental results

G	Optimal	Algorithm 1 Time(s)	Algorithm 2 Time(s)	Algorithm 3 Time(s)	NN Time(s)	Greedy Time(s)	3-opt+2-opt Time(s)
ulysses16	74.108	75.342 0.000	74.198 0.000	75.138 0.000	78.127 0.000	88.923 0.000	77.999 0.010
ulysses22	75.665	77.915 0.000	77.711 0.000	78.675 0.000	86.906 0.000	89.436 0.01	75.309** 0.042
bayg29	9074.148	9448.860 0.015	9768.448 0	9196.496 0.001	9964.781 0	9886.208 0.015	9074.148* 0.061
dantzig29	699	744.155 0.016	770.139 0.015	784.908 0.002	822.095 0	843.737 0.062	682.323** 0.352
att48	33523.708	37500.424 0.015	37057.581 0.016	36325.328 0.003	39236.885 0	38849.621 0.125	33607.682 0.516
eil51	429.983	443.070 0.031	495.628 0.031	440.746 0	505.774 0.016	481.518 0.125	429.983* 0.974
berlin52	7544.365	9047.211 0.031	9413.732 0.047	8618.198 0.002	8182.192 0	9954.062 0.281	7544.365* 0.300
st70	678.597	785.284 0.094	811.974 0.094	727.778 0.011	761.689 0	746.044 0.485	688.280 7.145
eil76	545.387	588.074 0.140	606.117 0.140	581.407 0.010	612.656 0.016	617.131 0.672	562.331 5.211
pr76	108159.438	118032.973 0.190	118235.042 0.190	117569.531 0.011	130921.005 0.010	127897.984 0.907	109266.770 0.744
gr96	512.309	581.877 0.235	540.357 0.235	550.495 0.029	603.302 0.015	580.101 1.609	516.870 3.488
rat99	1211	1311.904 0.266	1273.747 0.282	1316.432 0.030	1369.535 0.016	1528.308 1.875	1247.094 3.406
kroA100	21236.951	26135.302 0.360	24697.677 0.391	24093.242 0.020	24698.497 0.016	24197.285 1.937	21285.443 10.264
kroB100	22141	24700.544 0.406	23651.697 0.406	23419.490 0.031	25882.973 0.016	25815.214 2.469	22585.399 5.857

TABLE VI. Continuation of TABLE V

G	Optimal	Algorithm 1 Time(s)	Algorithm 2 Time(s)	Algorithm 3 Time(s)	NN Time(s)	Greedy Time(s)	3-opt+2-opt Time(s)
kroC100	20750.762	23962.861 0.391	24879.757 0.391	23512.300 0.030	23566.403 0.015	25313.671 2.610	20786.896 2.264
kroD100	21294.290	24783.197 0.422	23201.380 0.390	24758.054 0.010	24855.799 0.016	24631.533 2.359	21733.785 10.015
kroE100	22068	26036.072 0.375	25499.724 0.406	24822.113 0.010	24907.022 0.016	24420.355 2.609	22331.660 19.028
rd100	7910.396	9866.781 0.406	8945.544 0.375	9384.955 0.030	9427.333 0.015	8702.605 2.922	8101.042 4.409
eil101	642.309	712.461 0.329	694.685 0.359	704.361 0.010	736.368 0.015	789.112 2.609	661.138 7.162
lin105	14382.995	19679.294 0.360	17744.411 0.344	18354.693 0.030	16939.441 0.015	16479.785 3.187	14382.995* 45.414
pr107	44303	56635.995 0.453	47060.739 0.438	54003.941 0.010	46678.154 0.016	48261.816 2.109	44576.123 47.065
gr120	1666.508	1823.521 0.640	1737.452 0.703	1703.981 0.049	1850.263 0.032	1915.918 4.282	1675.255 6.100
pr124	59030	79142.752 0.797	64254.490 0.797	70395.718 0.056	67056.601 0.031	73952.492 5.281	59091.835 77.232
bier127	118282	134485.140 0.656	134398.465 0.687	129348.929 0.057	133970.646 0.032	136924.859 9.453	120970.110 81.012
ch130	6110.860	6963.303 0.875	6636.392 0.953	6873.837 0.055	7198.741 0.016	7142.045 7.688	6250.213 7.572
pr136	96772	108776.600 0.937	104197.185 0.937	105557.453 0.081	114560.902 0.031	119553.703 6.312	99604.824 206.122
pr144	58537	79638.710 1.125	66682.412 1.109	91107.078 0.057	60963.265 0.031	74613.406 8.438	58586.536 93.415
kroA150	26524	31027.270 1.469	28444.581 1.453	30216.728 0.082	31482.020 0.047	31442.994 11.094	27229.789 188.898
kroB150	26130	31824.589 1.547	30815.938 1.454	29631.101 0.064	31320.340 0.047	31519.083 11.156	26802.108 264.646
pr152	73682	98730.000 1.297	84999.926 1.329	93118.625 0.070	79566.585 0.047	84881.429 10.531	74661.440 247.713
u159	42080	54194.759 1.641	50014.988 1.656	49710.097 0.102	48586.725 0.031	50238.179 10.735	43786.312 26.115
rat195	2323	2763.001 3.563	2637.019 3.532	2554.672 0.085	2628.561 0.109	2957.176 29.719	2473.668 221.204
d198	15780	18147.697 3.625	17472.707 3.188	17941.087 0.192	17809.725 0.109	18595.822 39.625	16105.824 263.942
kroA200	29368	35195.046 5.187	35792.822 5.172	33629.972 0.112	34547.691 0.125	37650.812 45	30876.078 19.547

VI. CONCLUSION

In this study, the new heuristic methods for TSP are developed, these new methods are compared with the well-known methods in literature, and the experimental results are presented. The experimental results demonstrated that the known optimum results in literature are not actually optimum.

REFERENCES

[1] F. Nuriyeva, "New heuristic algorithms for travelling salesman problem", 25th Conference of European Chapter on Combinatorial Optimization, (ECCO XXV), Antalya, Turkey, April 26 – 28, 2012.

[2] D.S. Johnson and L.A. McGeoch, "The Traveling Salesman Problem: A Case Study", Local Search in Combinatorial Optimization, p. 215-310, John Wiley & Sons, 1997.

[3] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan, D. B. Shmoys, "The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization", John Wiley & Sons, 1986.

[4] G. Gutin and A. P. Punnen (Eds.), "The Travelling Salesman Problem and Its Variations", Kluwer Academic Publishers, 2002

[5] G. Reinelt, "The Traveling Salesman: Computational Solutions for TSP Applications", Springer-Verlag, Germany, 1994.

[6] http://www.tsp.gatech.edu/

[7] www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/

[8] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ST