# A Metaheuristic Algorithm with Hybrid Insertion Procedure for the Traveling Salesman Problem

**Takahiro Hoshino[1], Chuan Tian[1], Hisashi Kondo[2], Kazuhiro Tsuboi[2], Yoshio Hamamatsu[1]**

**[1] College of Science and Technology, Nihon University**
**Chiyoda-ku, Tokyo 101-8308, Japan**

**[2] College of Engineering, Ibaraki University**
**Hitachi, Ibaraki 316-8511, Japan**

**Abstract**

Finding the optimal solution to combinatorial optimization problems can be difficult, even when a partial structure (solution) of the optimal solution may be found easily. How to expand the partial solution into a full solution and how to insert other elements are important considerations for the construction of initial solutions. We propose a new procedure for constructing initial solutions of combinatorial optimization problems. This procedure employs hybrid insertion, which combines standard insertion with other heuristics. The proposed algorithm is applied to the traveling salesman problem (TSP), an NP-hard combinatorial optimization problem, to demonstrate the efficiency of the proposed algorithm. The accuracy of the obtained solutions is evaluated against the solutions of benchmark problems. On most problems, the proposed algorithm found shorter tours than other construction algorithms did.

***Keywords:*** *Traveling Salesman Problem, Threshold, CCA Algorithm, Minimum Spanning Tree*

## 1. Introduction

The main purpose of combinatorial optimization is to find the best choice from among a large but finite set of choices. It is often difficult to find the optimal solution to combinatorial optimization problems as the scale of the problem grows. However, a partial structure (solution) of the optimal solution can sometimes be found relatively easily. For example, parts of the solution to a scheduling problem can be assigned with certainty due to hard constraints. To generate initial solutions of higher accuracy, the partial solution needs to be more effectively expanded into a full solution via insertion of additional elements. Toward this end, we propose a new construction procedure of using hybrid insertion, which combines standard insertion with other heuristics. The traveling salesman problem (TSP) is an NP-hard combinatorial optimization problem. Application of the proposed algorithm to TSP is used here as a basis for discussing the effectiveness of the algorithm.

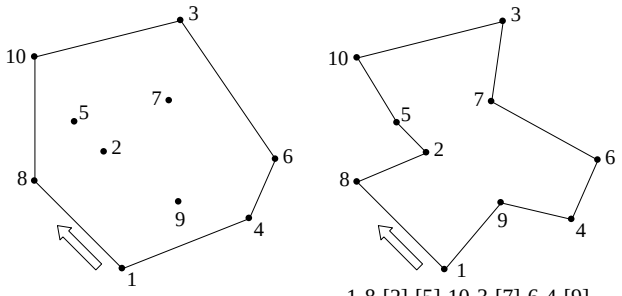
The aim of the TSP is to find the shortest possible Hamiltonian cycle for a number of cities given the distances between them. The TSP has several applications, such as vehicle routing problems[1]-[3] and circuit board drilling problems[4]-[6].

The optimal solution to the TSP can easily be obtained by finding all ordered tour combinations and selecting the shortest among them. However, the number of tour combinations increases exponentially with the number of cities, giving large computation times for even a small number of cities. Approximate metaheuristic algorithms that provide relatively high accuracy using short computation times are therefore required.

Genetic algorithms[7],[8], simulated annealing[9], ant colony optimization[10], and other techniques[11],[12] are well-known efficient metaheuristic algorithms for solving TSP. Many of these algorithms consist of a construction procedure that generates an initial set of tours and an improvement procedure that finds shorter tours from the initial tours. The nearest-neighbor algorithm, greedy algorithm, savings heuristic[13], and Christofides algorithm[14] are important construction algorithms. Another construction algorithm is the CCA algorithm[15], which combines convex hull, cheapest insertion, and angle selection techniques. It first selects cities located on the boundary of the convex hull as the initial subtour, and is able to find shorter tours from that set than the other construction algorithms mentioned can. Most of the construction algorithms select the locally optimal choice at each stage, and the insertion cost typically increases as the number of unlinked cities decreases.

The proposed algorithm chooses between using the standard insertion procedure from the CCA algorithm and an insertion algorithm employing the minimum spanning tree (MST). The choice depends on the insertion angle between the two edges that connect an isolated city to two cities on the current subtour. The solutions obtained by the construction procedure in the proposed algorithm are improved by using the 2-opt algorithm[16].

We compare the solutions given by our algorithm with results from the CCA plus 2-opt algorithm (CCAO algorithm) and the nearest neighbor plus 2-opt algorithm (NNO algorithm). To evaluate the accuracy of the obtained solutions, our algorithm and the other algorithms are applied to benchmark problems in TSPLIB 95[17].




(a) Cities on the boundary of the convex hull

[*i*]: City within the boundary

(b) Optimal tour

Fig.1 Convex hull and the optimal tour

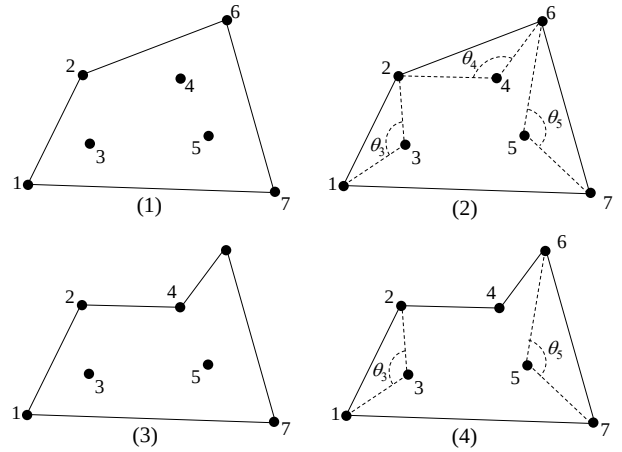


Fig.2 CCA algorithm

## 2. CCA Algorithm

Consider the optimal solution of a Euclidean TSP that visits the cities on the boundary of the convex hull in the same order as in Refs. [18] and [19]. Figure 1 shows an example that illustrates the relationship between the convex hull and the optimal tour. In the clockwise direction, the order of the cities on the boundary of the convex hull is 1-8-10-3-6-4 (Fig. 1(a)). The optimal tour maintains this order (Fig. 1(b)). The optimal solution can thus be obtained by inserting the remaining cities into the subtour given by the boundary of the convex hull. The CCA algorithm exploits this property by constructing solutions from the initial subtour given by the convex hull. The CCA algorithm as given in Ref. [15] is as follows.

1. Form an initial subtour by constructing the convex hull of the set of cities (Fig. 2(1)).
2. Let $c_{ij}$ denote the distance between cities $i$ and $j$. For each city $k$ not yet contained in the subtour, find the two cities $i$ and $j$ in the subtour that minimize $c_{ik} + c_{kj} - c_{ij}$ (Fig. 2(2)).

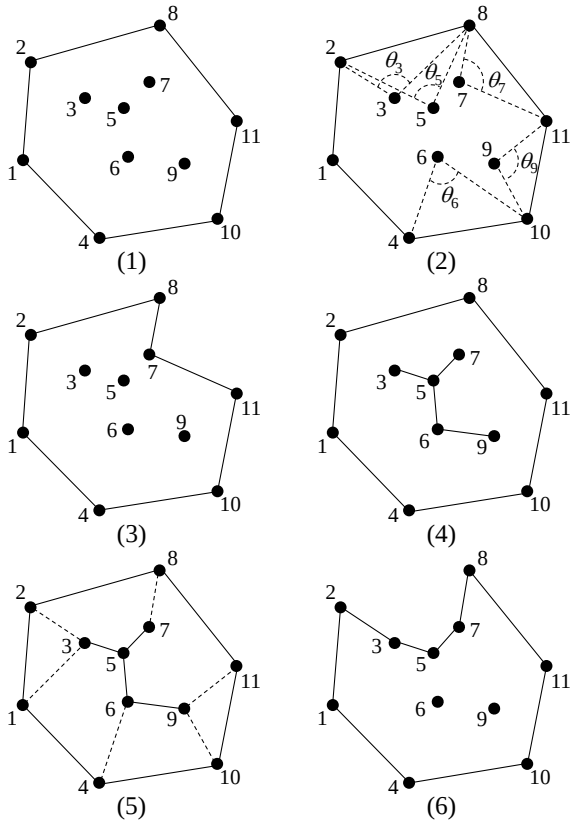


Fig.3 Construction procedure in the proposed algorithm

3. From all the combinations of ($i$, $k$, $j$) found in step 2, select ($i^*, k^*, j^*$) that gives the maximum insertion angle $\theta_{k^*}$ where $\theta_{k^*}$ is given by

$$\theta_{k^*} = \cos^{-1}\left(\frac{c_{i^*k^*}^2 + c_{j^*k^*}^2 - c_{i^*j^*}^2}{2c_{i^*k^*}c_{j^*k^*}}\right). \quad (1)$$

Insert city $k^*$ between $i^*$ and $j^*$ (Fig. 2(3)).

4. Repeat steps 2 and 3 until a Hamiltonian cycle is obtained (Fig. 2(4)).

Note that step 2 finds the two cities in the subtour that give the cheapest insertion cost and step 3 finds the unconnected city that gives the greatest insertion angle.

## 3. Proposed Algorithm

The proposed algorithm employs a threshold value that can be varied to generate several different solutions.

### 3.1 Construction Procedure

In the CCA algorithm, a long route may get inserted into the subtour during the process of reducing the number of unconnected cities since this algorithm does not reconstruct the tour or any subtours. We therefore improve the insertion procedure by implementing a threshold value as the tour cost increases. The threshold value is applied to

the cosine of the insertion angle because the insertion cost tends to grow as the insertion angle decreases, that is, as the cosine increases. If one or more cosines are less than the threshold value, then the city corresponding to the angle that has the minimum cosine value is inserted. However, if all cosines are greater than the threshold, then cities are inserted by the following procedure. First, to decrease the insertion cost for unconnected cities, the MST of the unconnected cities is constructed. The proposed algorithm then inserts the cities that give the minimum value of the ratio of route cost to number of cities in the MST. The entire algorithm as shown in Fig. 3 is as follows.

1. Generate an initial subtour by constructing the convex hull of the set of cities (Fig. 3(1)).
2. Perform step 2 of the standard CCA algorithm. That is, find cities *i* and *j* for each *k* not in the subtour (Fig. 3(2)), and then choose $k^*$ that has the minimum value of $\cos\theta_{k^*}$ from among all the combinations of (*i*, *j*, *k*). If $\cos\theta_{k^*}$ is less than the given threshold, go to step 3, otherwise go to step 4.
3. Insert city $k^*$ between $i^*$ and $j^*$ (Fig. 3(3)). If a Hamiltonian cycle has not been formed, go to step 2, otherwise go to step 7.
4. Generate the MST of the unconnected cities (Fig. 3(4)).
5. For each pair of linked cities *i* and *j* on the current subtour, search the MST for the nearest neighbors *l* and *m* of *i* and *j*, respectively (Fig. 3(5)).
6. Find ( $i^*$, $j^*$, $l^*$, $m^*$ ) from among all the combinations of (*i*, *j*, *l*, *m*) found in step 5. ($i^*$, $j^*$, $l^*$, $m^*$) gives the minimum value of
$$\frac{d_{l^*m^*} + c_{i^*l^*} + c_{j^*m^*} - c_{i^*j^*}}{\text{Number of cities from } l^* \text{ to } m^*}$$
where $d_{l^*m^*}$ is the total distance along the path linking $l^*$ to $m^*$ on the MST. Insert all of the cities contained in the path from $l^*$ to $m^*$ inclusive in between $i^*$ and $j^*$ (Fig. 3(6)). If a Hamiltonian cycle has not been formed, go to step 2, otherwise go to step 7.
7. Improve the tour by using the 2-opt algorithm.

The above algorithm thus gives a tour for a given threshold value, and the tour cost depends on that threshold value. The next section therefore presents a method for generating a threshold value that gives better solutions.

### 3.2 Method for Selecting the Threshold Value

We have also devised a procedure for searching for good solutions by varying the threshold value. Let $T$ be the threshold value, $L(T)$ be the tour length for the threshold value $T$, and $d$ be a "depth" parameter that is used for generating new threshold values.

First, at depth $d = -1$ we find the solutions for $T = -1$ and $T = 1$ by applying the construction procedure. For

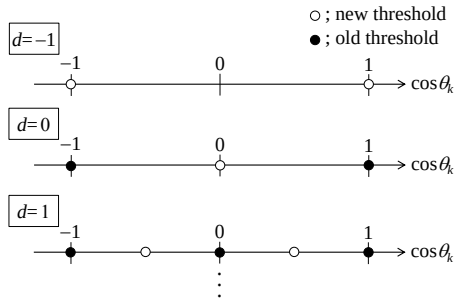


Fig.4 Process of generating new threshold values

depths $d \geq 0$, new threshold values are then obtained from the midpoint values between each pair of neighboring threshold values at the previous depth, such as shown in Fig. 4, and tours are constructed for each of these threshold values. The depth $d$ is increased by one after generating all of the midpoint values. This process is repeated until $d$ reaches some preset "max depth" parameter.

The number of new thresholds and solutions increases exponentially as the max depth increases, and so we impose the following conditions on generating new threshold values:

[1] The range ranking, defined as the ranking of the values of $\left(L(T_i) + L(T_j)\right)/2$ for the range between two neighboring threshold values $T_i$ and $T_j$, does not exceed some preset value.

[2] $L(T_i)$ does not equal $L(T_j)$.

Condition [1] means that threshold values likely to give good solutions are generated, and threshold values that give bad solutions are not generated. Since good solutions are expected to have similar structures in parts of the solutions, threshold values that give good solutions often exist near other threshold values that also give good solutions. The preset parameter employed in condition [1] is called the worst-range ranking. Condition [2] is applied to avoid generating duplicate solutions.

The proposed algorithm including the method for generating new threshold values operates as follows.

1. Take initial parameters $T = -1$ and $d = -1$ and construct a tour by using the proposed algorithm.
2. Update the present threshold value with a new threshold value. If the present threshold is $T = -1$, the new threshold value is set to $T \rightarrow T + 2^{-d}$, otherwise it is set to $T \rightarrow T + 2^{-d+1}$.
3. Check the value of the new threshold $T$. If $T$ satisfies $T \leq 1$, go to step 4, otherwise go to step 5.
4. Determine whether $T$ satisfies the conditions for generating new threshold values. If $T$ satisfies them, construct a tour by using the proposed algorithm, otherwise go to step 2.
5. Set the parameters to $T = -1$ and $d \rightarrow d + 1$. If $d$ is less than or equal to the preset max depth parameter, go to step 2, otherwise stop.

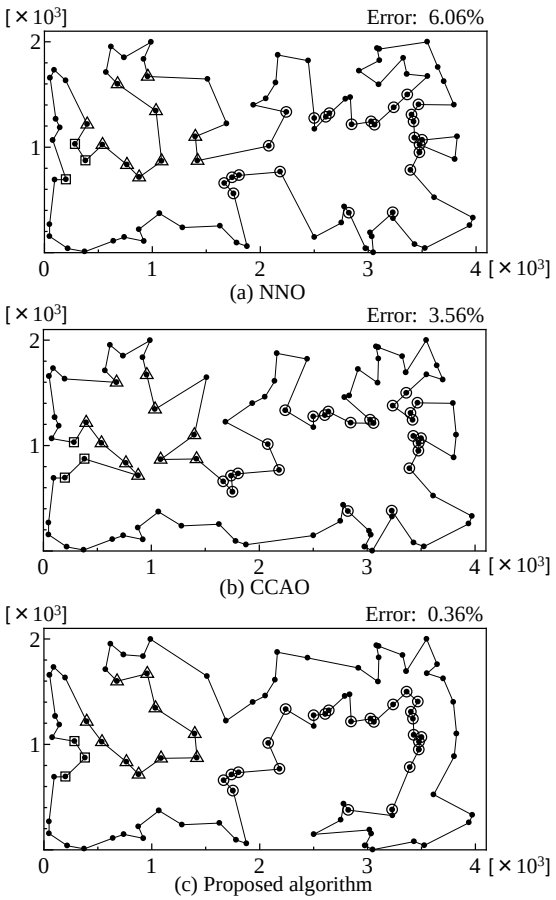



Fig.5 Computational results for kroE100

This algorithm is likely to find better solutions for larger values of the max depth and worst-range ranking parameters. However, more time is required for the computation as these values increase. To decrease the computation time, we made the following improvement to the above algorithm. When a city is inserted into a subtour during the construction procedure, the insertion procedure that was used (standard insertion or proposed insertion) and the insertion order of the city are recorded. This information is used in the construction procedure to allow generating only those parts that differ from previously obtained solutions.

### 3.3 Method for Setting Parameters

To obtain better solutions using short computation times, the max depth and worst-range ranking need to be set appropriately. The parameters are set through the following procedure.

The standard heuristic algorithm (i.e., CCA) is applied to a set of problems of about the same scale in which cities are randomly arranged and the tour length is recorded. The proposed algorithm generates solutions for these problems by using all settable combinations of worst-range ranking and max depth. We define improvement accuracy as the reduction in tour length by the proposed algorithm over the CCA algorithm, and create a table of the improvement accuracies corresponding to the worst-range ranking and max depth. The improvement accuracies of the proposed algorithm tend to saturate with respect to increases in max depth and worst-range ranking. When the worst-range ranking is set to $r$ th ($r = 1,2,\cdots$), if the reduction in the improvement accuracy of $d$ from the previous depth $d-1$ is less than a prescribed value, the depth $d$ is recorded as $D_r$. This procedure sets the max depth for each $r$ to a fixed value. Similarly, if reduction in the improvement accuracy from the parameters $(r-1, D_{r-1})$ to the parameters $(r, D_r)$ is less than a prescribed value, the worst-range ranking $r$ is recorded and these parameters $(r, D_r)$ are used in the algorithm. Setting the parameters by the above procedure makes it possible to cut down on wasted computing time that offers little chance of improvement.

## 4. Computational Study

The solutions obtained by the CCAO algorithm, the NNO algorithm, and the proposed algorithm were compared to investigate the influence of tour construction.

### 4.1 Computation Conditions

The solution accuracies of the algorithms were evaluated by using 70 benchmark problems taken from TSPLIB 95[17]. To set the worst-range ranking and max depth parameters, the CCA algorithm and the proposed algorithm were applied to 5 test problems consisting of 1000 randomly arranged cities. By method introduced in Section 3.3, the max depth was set to 8 and the worst-range ranking was set to 5 as appropriate parameters for the proposed algorithm. The best solution tours were obtained for several different threshold values using the proposed algorithm. All computational experiments were performed on a personal computer with a Core i7 processor and 4 GB of RAM.

### 4.2 Computational results

We now examine the difference in accuracy between the algorithms by comparing the solutions. Figure 5 shows the solutions given by each algorithm for a problem consisting of 100 cities (kroE100). Vertices represent cities and solid lines linking the cities represent the salesman's route. The cities indicated by circle, square, and triangle markings in Fig. 5(c) represent the cities that were inserted from the MST constructed in step 4 in Section 3.1. For comparison, these same cities are also indicated in Figs. 5 (a) and (b).

Comparing the results of the NNO and CCAO algorithms shows that CCAO improves the accuracy by setting the initial subtour to the boundary of the convex hull.

Table 1 Comparison of accuracies and computation times of the algorithms

| Instance | Accuracy [%] | | | Time [s] | | | Instance | Accuracy [%] | | | Time [s] | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NNO | CCAO | PA | NNO | CCAO | PA | | NNO | CCAO | PA | NNO | CCAO | PA |
| eil51 | 7.61 | 4.87 | 3.04 | 0.01 | 0.03 | 0.11 | **pr299** | 8.43 | 3.26 | 3.26 | 0.08 | 0.22 | 1.12 |
| berlin52 | 10.26 | 1.02 | 0.03 | 0.02 | 0.03 | 0.10 | lin318 | 8.06 | 6.65 | 4.28 | 0.09 | 0.27 | 1.97 |
| st70 | 10.90 | 4.90 | 3.56 | 0.02 | 0.05 | 0.16 | rd400 | 6.01 | 5.90 | 3.85 | 0.09 | 0.44 | 2.37 |
| eil76 | 6.38 | 7.53 | 2.03 | 0.03 | 0.05 | 0.17 | fl417 | 5.46 | 2.91 | 2.68 | 0.14 | 0.47 | 2.09 |
| pr76 | 4.91 | 1.87 | 1.17 | 0.02 | 0.05 | 0.18 | pr439 | 7.95 | 5.56 | 4.03 | 0.13 | 0.56 | 2.90 |
| rat99 | 5.26 | 4.94 | 2.63 | 0.03 | 0.06 | 0.31 | pcb442 | 5.97 | 9.01 | 1.24 | 0.13 | 0.56 | 2.09 |
| **kroA100** | 4.08 | 0.94 | 0.94 | 0.03 | 0.06 | 0.27 | d493 | 7.20 | 4.47 | 3.42 | 0.13 | 0.72 | 5.80 |
| kroB100 | 2.97 | 1.03 | 0.53 | 0.03 | 0.06 | 0.27 | u574 | 7.51 | 6.19 | 3.67 | 0.19 | 1.05 | 7.88 |
| **kroC100** | 6.62 | 1.13 | 1.13 | 0.03 | 0.06 | 0.28 | rat575 | 8.22 | 7.62 | 3.73 | 0.19 | 1.03 | 5.83 |
| kroD100 | 8.91 | 2.72 | 2.08 | 0.02 | 0.06 | 0.27 | p654 | 4.72 | 4.95 | 2.01 | 0.30 | 1.47 | 13.41 |
| kroE100 | 6.06 | 3.56 | 0.36 | 0.03 | 0.08 | 0.26 | d657 | 7.85 | 5.90 | 4.53 | 0.17 | 1.51 | 9.84 |
| **rd100** | 11.05 | 3.07 | 3.07 | 0.03 | 0.05 | 0.25 | u724 | 9.06 | 6.71 | 3.88 | 0.20 | 1.95 | 14.12 |
| eil101 | 7.91 | 5.43 | 2.58 | 0.02 | 0.08 | 0.23 | rat783 | 8.74 | 7.07 | 4.72 | 0.31 | 2.40 | 16.52 |
| lin105 | 14.92 | 1.21 | 1.02 | 0.03 | 0.08 | 0.21 | pr1002 | 9.23 | 6.76 | 5.15 | 0.44 | 5.12 | 41.40 |
| pr107 | 2.46 | 1.60 | 0.05 | 0.03 | 0.08 | 0.31 | u1060 | 9.28 | 5.68 | 3.26 | 0.52 | 6.05 | 55.35 |
| pr124 | 3.96 | 2.30 | 1.13 | 0.03 | 0.08 | 0.26 | vm1084 | 6.79 | 5.30 | 3.45 | 0.52 | 6.63 | 43.13 |
| bier127 | 8.06 | 4.34 | 2.96 | 0.05 | 0.09 | 0.34 | pcb1173 | 9.71 | 8.51 | 4.58 | 0.64 | 9.05 | 32.73 |
| ch130 | 8.67 | 5.63 | 2.87 | 0.05 | 0.09 | 0.39 | d1291 | 11.10 | 7.65 | 5.08 | 0.62 | 12.31 | 60.43 |
| **pr136** | 13.16 | 2.84 | 2.84 | 0.05 | 0.09 | 0.30 | rl1304 | 7.89 | 6.60 | 2.20 | 0.78 | 12.75 | 59.45 |
| pr144 | 3.79 | 4.27 | 2.27 | 0.03 | 0.09 | 0.31 | rl1323 | 8.16 | 5.10 | 3.15 | 0.62 | 13.46 | 68.80 |
| ch150 | 5.17 | 4.40 | 1.17 | 0.05 | 0.11 | 0.34 | nrw1379 | 7.19 | 6.10 | 3.16 | 1.00 | 15.38 | 117.13 |
| kroA150 | 7.64 | 1.82 | 1.06 | 0.05 | 0.11 | 0.43 | fl1400 | 4.69 | 3.79 | 2.57 | 1.37 | 15.94 | 121.37 |
| kroB150 | 9.22 | 2.93 | 1.36 | 0.05 | 0.11 | 0.34 | u1432 | 7.80 | 8.88 | 3.36 | 1.03 | 17.35 | 55.08 |
| pr152 | 5.18 | 4.38 | 2.17 | 0.05 | 0.09 | 0.50 | fl1577 | 5.81 | 6.14 | 3.02 | 1.73 | 23.51 | 164.77 |
| u159 | 10.01 | 5.11 | 2.94 | 0.05 | 0.09 | 0.34 | d1655 | 7.77 | 7.25 | 5.59 | 1.58 | 27.77 | 164.41 |
| rat195 | 5.88 | 6.82 | 3.00 | 0.05 | 0.12 | 0.44 | vm1748 | 9.99 | 6.19 | 4.92 | 1.58 | 33.10 | 183.80 |
| d198 | 2.88 | 2.01 | 1.00 | 0.05 | 0.13 | 0.71 | u1817 | 9.83 | 8.90 | 7.68 | 1.16 | 37.60 | 238.28 |
| kroA200 | 7.21 | 2.97 | 1.64 | 0.06 | 0.14 | 0.56 | rl1889 | 9.80 | 7.71 | 4.82 | 2.04 | 42.96 | 169.88 |
| **kroB200** | 7.16 | 1.70 | 1.70 | 0.05 | 0.12 | 0.59 | d2103 | 4.08 | 6.84 | 2.81 | 1.30 | 61.28 | 201.79 |
| ts225 | 10.76 | 8.40 | 0.00 | 0.06 | 0.16 | 0.30 | u2152 | 8.26 | 8.36 | 6.95 | 2.46 | 64.29 | 418.11 |
| tsp225 | 4.75 | 4.74 | 1.88 | 0.06 | 0.14 | 0.63 | *u2319* | 2.24 | 8.47 | 2.87 | 1.78 | 80.58 | 148.20 |
| pr226 | 9.39 | 3.72 | 1.38 | 0.06 | 0.17 | 0.65 | pr2392 | 8.14 | 7.92 | 6.05 | 3.40 | 90.06 | 684.20 |
| gil262 | 5.61 | 5.49 | 3.74 | 0.06 | 0.19 | 1.15 | pcb3038 | 9.15 | 7.71 | 4.96 | 4.26 | 193.93 | 891.99 |
| pr264 | 6.72 | 7.81 | 2.69 | 0.06 | 0.20 | 1.34 | fl3795 | 6.42 | 2.91 | 1.62 | 10.42 | 392.12 | 1815.80 |
| a280 | 6.65 | 4.01 | 1.01 | 0.06 | 0.19 | 0.92 | fnl4461 | 7.57 | 7.24 | 4.50 | 6.85 | 616.12 | 4227.81 |

However, the CCAO algorithm sometimes generates long paths due to the cheapest cost selection and greatest angle insertion procedure. By inserting cities from the MST instead of using this procedure, the tour in the proposed algorithm gives the route improvement shown in Fig. 5(c). In particular, the route through the cities indicated by the triangles is inserted more effectively in comparison with the CCAO algorithm. The proposed algorithm thus accomplishes a reduction in the total tour cost.

We next discuss the accuracies and computation times for the 70 problems listed in Table 1. The proposed algorithm gives the best accuracy in all of the problems except for the "u2319" problem indicated by italics. Compared to the CCAO algorithm, the proposed algorithm finds shorter tours in 64 of the problems. The solutions to the other 6 problems, which are highlighted in boldface, have accuracy equal to that given by the CCAO algorithm. Since the solution for $T$=1 corresponds to the solution of the CCAO algorithm because the MST is not constructed, the proposed algorithm can guarantee to match the accuracy of the CCAO algorithm. The average error is 7.40% for NNO, 5.13% for CCAO, and 2.86% for the proposed algorithm. Hence, our algorithm clearly increases the accuracy of the solutions.

We next discuss the scale of the problems and the accuracies. Figure 6 shows the accuracies of the proposed

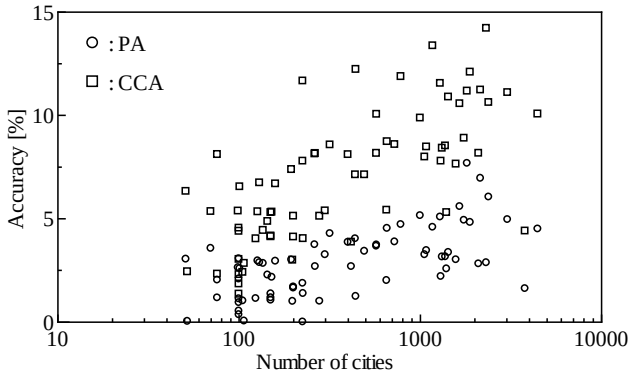



Fig.6 Accuracy versus number of cities

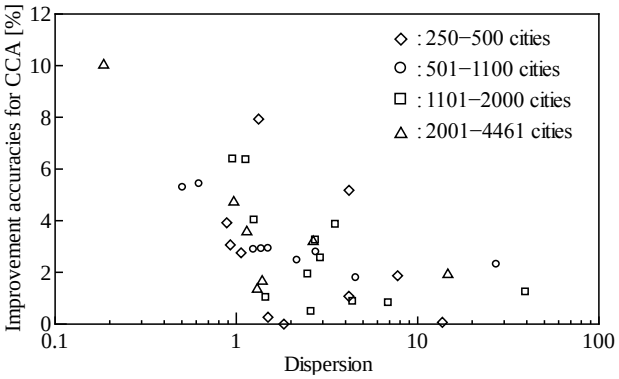



Fig.7 Accuracy improvement versus dispersion

algorithm and CCA algorithm versus the number of cities. The accuracies of both algorithms have a tendency to decrease as the number of cities increases. The CCA algorithm is able to efficiently insert cities on and near the boundary of the convex hull because of the characteristics of the algorithm. There is no guarantee of other cities being inserted efficiently because of greedy insertion. Thus, the accuracies of both algorithms worsen as the scale of the problem increases. By using hybrid insertion, however, the proposed algorithm can better prevent the decrease in accuracy than the CCA algorithm can.

Figure 6 also shows that the improvement accuracies of the proposed algorithm and CCA algorithm differ for the same scale of problem. Improvement accuracy tends to depend on the locations of the cities. We discuss the relationship between the accuracies and city arrangements in terms of the dispersion of the number of cities per unit area. The area for each benchmark problem is partitioned into a square lattice such that the average number of cities per unit area is approximately one and the dispersion of the number of cities per unit area can be obtained. Problems with large dispersions have many areas of dense cities, and problems with small dispersions have less variation between areas than problems with large dispersions. Figure 7 shows improvement accuracies of the proposed algorithm before 2-opt for the CCA algorithm versus dispersion value. This figure shows the results for problems containing 250 or more cities because few differences are found between the accuracies of the proposed algorithm and the CCA algorithm for problems with less than 250 cities. As can be seen in Fig. 7, improvement accuracies tend to decrease as dispersion increases. The reason for this trend is the similar structure of the routes generated by steps 2 and 3 in the CCA algorithm and steps 4 to 6 in the proposed algorithm, since distances between isolated cities are often similar in problems with large dispersions. However, the accuracies of the problems with small dispersions are often improved by using hybrid insertion due to maintaining more than a certain distance between isolated cities.

Finally, we examine the difference between the calculation times of the algorithms. From the results for the CCAO and the NNO algorithm, the computation time of the former is longer than the latter. The reason is that the computation time required by the construction procedure in the CCA algorithm is $O(N^3)$ compared to $O(N^2)$ in the nearest neighbor algorithm. A similar trend can be seen when the results for the proposed algorithm are compared with the NNO algorithm. The computation time of the proposed algorithm is on the order of 2 to 9 times larger than that of the CCA algorithm. The increase depends on the number of tours, that is, the number of thresholds, and this number varies between problems due to the generating conditions described in Section 3.2. The proposed algorithm thus requires different computation time even when the number of cities is the same. The computation time also depends on the frequency with which the insertion procedure employing the MST is used. The proposed procedure requires longer running times when insertion using the MST occurs more frequently.

## 5. Conclusion

We proposed a new construction procedure that employs hybrid insertion and discussed the effectiveness of the algorithm in terms of application to the TSP. In the proposed algorithm, an insertion criterion is employed that depends on the cosine of the insertion angle exceeding a given threshold value. When the threshold is exceeded, the algorithm inserts cities from an MST over the cities that have not yet been inserted, otherwise the CCA algorithm is employed. We also proposed a method for generating new thresholds by using preset max depth and worst-range ranking parameters to obtain better solutions.

We applied our algorithm to benchmark problems in TSPLIB 95, and compared the solutions with those obtained by the CCAO and NNO algorithms. The results clearly showed that our algorithm gives higher accuracy in most of the benchmark problems. Through a combination

of insertion by the standard CCA algorithm and insertion from an MST, our algorithm is guaranteed to give at least the accuracy of the CCA algorithm while simultaneously offering the ability to find shorter tours. We expect that errors could be further reduced by utilizing the tour constructed by our algorithm as the initial tour of other metaheuristic algorithms, such as GA. The idea of selecting between the standard insertion procedure and the proposed insertion procedure depending on the situation could also be incorporated into other construction algorithms. Examples of such algorithms are the nearest insertion method, the farthest insertion method, and the arbitrary insertion method. The TSP is used in this paper as an example of the application of this idea. How to apply the idea to other combinatorial optimization problems is left as an issue for the future.

The proposed algorithm requires longer computation time than the CCAO and NNO algorithms do, which is commensurate with the increase in accuracy. The computation time of the proposed algorithm could be improved by replacing the CCA algorithm which is the base of the proposed algorithm with a more efficient algorithm. Furthermore, we used Prim's algorithm[20] for the construction of the MST which has a computation time of $O(N^2)$ for finding the MST. The computation time for building the MST could be reduced to $O(N\log N)$ by implementing a faster MST algorithm such as that given in Refs. [21] and [22]. This issue will be addressed in the future.

## References

[1] G. Laprote: "The vehicle routing problem: An overview of exact and approximate algorithms", European Journal of Operations Reseach, Vol. 59, No. 3, pp. 345-358, (1992)

[2] Mir Mohammad Alipour: “A Learning Automata Based Algorithm For Solving Capacitated Vehicle Routing Problem”, International Journal of Computer Science Issues, Vol. 9, Issue 2, No.1 pp. 138-145 (2012)

[3] S. A. Kallel1 and Y. Boujelbene: “Heterogeneous Vehicle Routing Problem with profits Dynamic solving by Clustering Genetic Algorithm”, International Journal of Computer Science Issues, Vol. 10, Issue 4, No.1 pp. 247-253 (2013)

[4] S. Danusaputro, C. Y. Lee and L. A. Martin-Vega: "An efficient algorithm for drilling printed circuit boards", Computers & Industrial Engineering, Vol. 18, No. 2, pp. 145-151, (1990)

[5] E. Aoyama, T. Hirogaki, T. Katayama and N. Hashimoto: "Optimizing drilling conditions in printed circuit board by considering hole quality optimization from viewpoint of drill-movement time", Journal of Materials Processing Technology, Vol. 155-156, No. 30, pp. 1544-1550, (2004)

[6] M. Ancau: "The optimization of printed circuit board manufacturing by improving the drilling process productivity", Computers & Industrial Engineering, Vol. 55, No. 2, pp. 279-294, (2008)

[7] M. Yamamura, T. Ono and S. Kobayashi: "Character-preserving genetic algorithms for traveling salesman problem", Journal of the Japan Society for Artificial Intelligence, Vol. 7, No. 6, pp. 1049-1059, (1992)

[8] Hassan Ismkhan, Kamran Zamanifar: “Developing Improved Greedy Crossover to Solve Symmetric Traveling Salesman Problem”, International Journal of Computer Science Issues, Vol. 9, Issue 4, No.3 pp. 121-126 (2012)

[9] S. C. Lin and J. H. C. Hsueh: "A new methodology of simulated annealing for the optimization problems", Physica A: Statistical Mechanics and its Applications, Vol. 205, No. 1-3, pp. 367-374, (1994)

[10] H. M. Naimi and N. Taherinejad: "New robust and efficient ant colony algorithms: Using new interpretation of local updating process", Expert Systems with Applications, Vol. 36, No. 1, pp. 481-488, (2009)

[11] X. Yan, C. Zhang, W. Luo, W. Li, W. Chen and H. Liu: “Solve Traveling Salesman Problem Using Particle Swarm Optimization Algorithm”, International Journal of Computer Science Issues, Vol. 9, Issue 6, No.2 pp. 264-271 (2012)

[12] J. Knox: "Tabu search performance on the symmetric traveling salesman problem", Computers & Operations Research, Vol. 21, No. 8, pp. 867-876, (1994)

[13] G. Clarke and J. W. Wright: "Scheduling of vehicles from a central depot to a number of delivery points", Operations Research, Vol. 12, No. 4, pp. 568-581 (1964)

[14] N. Christofides: "Worst-case analysis of a new heuristic for the travelling salesman problem", Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh (1976)

[15] E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy and D.B.Shmoys, "The Traveling Salesman Problem", John-Wiley and Sons (1985)

[16] E. Aarts and L. K. Lenstra: "Local search in combinatorial optimization", John wiley and sons, pp. 215-310 (1997)

[17] G. Reinelt, "TSPLIB 95", Universität Heidelberg, http://www2.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ (1995)

[18] Merrill M. Flood, “The traveling salesman problem”, Operations Research, Vol. 4, pp. 61-75 (1956)

[19] Y. Takenaka, N. Funabiki, “An application of convex hull to genetic algorithm for traveling salesman problem, Technical report of IEICE, SS97-48, pp. 17-24 (1998)

[20] R. C. Prim: “Shortest connection networks and some generalizations”, Bell Syst. Tech. J., Vol. 36, pp. 1389-1401 (1957)

[21] J. L. Bentley and J. H. Friedman: "Fast Algorithms for constructing minimal spanning trees in coordinate spaces", IEEE Trans. on computers, Vol. C-27, No. 2, pp. 97-105 (1978)

[22] Nevalaninen, J. Ernvall and J. Katajainen: "Finding minimal spanning trees in a Euclidean coordinate space", BIT, Vol. 21, pp.46-54 (1981)

**Takahiro Hoshino** received his B.E., M.E., and Ph.D. degrees in systems engineering from Ibaraki University in 2002, 2007, and 2010 and is now an assistant professor at the College of Science and Technology, Nihon University. He is a member of IEEJ and the Japan Society for Industrial and Applied Mathematics.

**Chuan Tian** graduated from the department of Computer-aided Mechanical Design, Qingdao Technical College, in 2005, and

received his B.E. degree in department of Computer Science and Technology from Southwest University of Science and Technology of China in 2009. After working as an engineer at software development in Qingdao Haier Software Corporation, he received his M.E. degree in Electrical Engineering, Nihon University of Japan in 2014. He is now a Ph.D. candidate at the College of Science and Technology, Nihon University.

**Hisashi Kondo** received his Dr.Eng. degree in information engineering from Hokkaido University in 1994. He is now a lecturer at the College of Engineering, Ibaraki University. His current research interests include cellular automata, swarm intelligence, and metaheuristics. He is a member of IEEE-CS, EATCS, IPSJ, JSAI, and the Japan Society for Software Science and Technology.

**Kazuhiro Tsuboi** received his M.Eng. and D.Eng. degrees in aeronautical engineering from the School of Engineering of the University of Tokyo in 1985 and 1993. After working as a researcher at the Institute of Computational Fluid Dynamics and at the Institute of Mathematical Science in Kao Corporation, he joined the faculty of Ibaraki University in 1996. He is now a professor in the Department of Intelligent Systems Engineering of Ibaraki University. He is a member of the Physical Society of Japan, the Japan Society for Industrial and Applied Mathematics, the Japan Society of Fluid Mechanics, the Japan Society of Mechanical Engineers, and also a senior member of the American Institute of Aeronautics and Astronautics.

**Yoshio Hamamatsu** graduated from the Department of Electrical Engineering, Nihon University, in 1974 and received his M.Eng. degree in 1976. In 1984, he received his Ph.D. degree from Hokkaido University. He became a research associate at Tamagawa University in 1976, an assistant professor in 1982, and an associate professor in 1988; an associate professor at Ibaraki University in 1992 and a professor in 1998; and a professor in the College of Science and Technology, Nihon University, in 2008, and a professor emeritus at Ibaraki University in 2008. He was a visiting research professor at the University of Delaware (USA) from 1985 to 1986, and also in 1988. His main area of research deals with traffic flow and queuing theory. He is a member of IEEJ.