Solving Traveling Salesman Problems by Combining Global and Local Search Mechanisms

Huai-Kuang Tsai¹, Jinn-Moon Yang², and Cheng-Yan Kao¹

Dept. of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan¹

Dept. of Biological Science and Technology & Institute of Bioinformatics, National Chiao Tung University, Hsinchu, Taiwan²

d7526010@csie.ntu.edu.tw

Abstract - In this paper an evolutionary algorithm for the traveling salesman problem is proposed. The key idea is to enhance the ability of exploration and exploitation by incorporating global search with local search. A new local search, called neighbor-join (NJ) operator, is proposed to improve the solution quality of the edge assembly crossover (EAX) considered as a global search mechanism in this paper. Our method is applied to 15 well-known traveling salesman problems whose numbers of cities are from 101 to 3038 cities. The experimental results indicate that the neighbor-join operator is very competitive with the related operators surveyed in this paper. Incorporating the NJ into the EAX significantly outperforms the method incorporating 2-opt into the EAX for some hard problems. For each test instance the average value of solution quality stays within 0.03% from the optimum. For the notorious hard problem att532, it is able to find the optimum solution 23 times in 30 independent runs.

I. INTRODUCTION

The traveling salesman problem (TSP) is a well-known NP-hard optimization problem, in which we require to determine the shortest closed route passing through a set of n cities under the condition that each city is visited exactly once. Many problems in science, engineering, and bioinformatics fields, such as flexible manufacturing systems, routing as well as scheduling problems, physical mapping problems [2], genome rearrangement [3], and phylogenetic tree construction [4] can be formulated as a TSP.

A large number of approaches have been devoted to solve the TSP. These approaches can be roughly divided into local (heuristic) search and global search approaches. In general, the local search approaches, such as 2-opt, 3-opt [5] and Lin-Kernigan [6] are efficient and fast convergence, but they may get stuck at local minima. On the other hand, the global search approaches, such as simulated annealing [7], Hopfield neural networks [8], and evolutionary algorithms [1], [9]-[13] have been proposed to reduce the ill effect of these local search methods, but they often converge more slowly than local search approaches. Therefore, incorporating heuristics into global search methods has been used to tradeoff between the convergence speed and the solution quality.

An evolutionary algorithm is based on the ideas borrowed from genetics and natural selection. It is a generally adaptable concept for problem solving, especially well suited for solving difficult optimization problems, where traditional optimization methods are less efficient. There are about three main independently developed but

strongly related implementations of evolutionary algorithms: genetic algorithms [14], evolution strategies [15], and evolutionary programming [16]. These three types of standard evolutionary algorithms are not very efficient for some specific application domains. Thus, some modifications [17]-[20] have been proposed to improve solution quality and to speed up convergence. A popular approach is to combine evolutionary algorithms with local search approaches [18], [21]. Such a hybrid approach may possess both the global optimality of the genetic algorithms and the convergence of the local searches.

In this paper, an evolutionary algorithm combining the edge assembly crossover (EAX) [1] and a new operator, called the neighbor-join (NJ) operator, is proposed for solving TSPs. The NJ is regarded as a local search strategy, while the EAX is treated as a global search strategy. The basic idea of the proposed approach is to enhance the ability of exploration and exploitation of the search space by the cooperation of global and local searches. The EAX proposed by Nagata [1] was considered a powerful crossover operator [22]. We design the NJ operator that is able to improve the solution quality of EAX. According to the experimental results, the proposed approach seems to be able to balance exploration and exploitation.

The NJ operator reconnects cities by considering neighbor information and referring to the knowledge taken from other individuals in the population. This operator can be viewed as a mixture of local search, mutation and recombination. The NJ operator mutates an individual itself (mutation), but the selection of reconnecting cities depends on both the geometric neighborhood information (local search) and the edges from other individuals in the population (recombination). Experimental results indicate that the NJ operator is competitive with other genetic operators proposed for the TSPs. The average value of solution quality stays within only 0.1% from the optimum when the numbers of cities are less than 318. At the same time, incorporating the NJ into the EAX outperforms the method incorporating 2-opt into the EAX.

The rest of this paper is organized as follows. Section 2 introduces the evolutionary nature of the proposed approach. Section 3 provides the experimental results on some benchmark problems and comparisons of our approach with several methods. Some characteristics of our approach are also discussed. Concluding comments are drawn in Section 4.

Fig. 1. The outline of the proposed evolutionary algorithm

II. APPROACH

In this section the details of the proposed genetic algorithm for TSP are presented. Our algorithm consists of the edge assembly crossover (EAX) for global search and a new genetic operator for local search, called neighbor-join (NJ). The EAX and the NJ are used to balance exploration and exploitation in the search space.

Fig. 1 shows the main steps of the proposed evolutionary algorithm. N solutions are generated as the initial population. Each solution is represented as a permutation from 1 to n , where n is the number of cities of a TSP. After evaluating the fitness, each individual (S_i) in the population sequentially applied the following steps to produce a child: S_i randomly selects another individual S_j and these two individuals produce an intermediate offspring (I_i) by conducting the EAX. Then the NJ is executed L times (L is the local search length) to generate a child C_i by refining from the offspring I_i . Each solution in the population executes above steps to generate its child and these N children become the new population of the next generation.

Our algorithm terminated when one of following criteria is satisfied: 1) all individuals of a population are the same, or 2) all of the children generated in continuous ten generations are worse than their parents. In the following subsections, the EAX and the NJ are described.

A. EAX

The EAX proposed by Nagata [1] was considered a powerful crossover operator. The EAX has two important features: preserving parents' edges using a novel approach and adding new edges by applying a greedy method, analogous to a minimal spanning tree. The EAX is considered as the global search strategy in our proposed algorithm.

The EAX is briefly described here and the detail

information can be referred [1]. Two individuals, denoted as A and B , were selected as parents. EAX first merges A and B into a single graph denoted G (Fig. 2). EAX travels G to generate many AB -cycles by alternately picking edges from parents A and B (Fig. 3). According to the heuristic and random selection rules, some of AB -cycles are selected to generate a quasi solution. Since edges in these selected AB -cycles are either from A or B , those edges exist in A are removed from A and those edges that exist in B are added to A .

The modified A then contains some disjoint subtours (Fig. 4). The EAX uses a greedy method to merge these distinct subtours into a valid solution (Fig. 5). This solution is returned if the fitness of this solution is better than its parents. Otherwise this procedure is repeated until a solution is better than A and B or L children are produced.

Fig. 2. A , B , and the graph G . G is obtained by overlapping A and B .

Fig. 3. An example of the division of undirected edges into AB -cycles on G . (a), (b), and (c) are effective AB -cycles, while AB -cycles in (d) are ineffective.

Fig. 4. Four examples of EAX individually apply some AB -cycles on A . (a), (b), and (c) are the results of applying AB -cycle shown in Fig. 3(a), 3(b), and 3(c), respectively, and (d) is the result of applying AB -cycles shown in Fig. 4(a) and 4(b) on A .

Fig. 5. After Modification, all examples in Fig. 5 are individually modified to valid solutions.

NJ (I_i, L): Given an intermediate individual I_i and local search length L

```

Let  $I_i' = I_i$ 
repeat  $L$  times
{
  randomly select a city  $c$  from  $I_i'$ 
  if ( $\text{rand}() > 0.5$ )
    Randomly select another individual from current
    population. Select the city  $c'$  connecting to  $c$  in the
    selected individual.
  else randomly select  $c'$  from the geometric nearest
  three cities of  $c$ .
  if the city  $c'$  doesn't connect to  $c$  in  $I_i'$ 
    a. To align  $c'$  and  $c$  together to generate four
    candidates (see Fig. 7).
    b. Select the one with lowest fitness from these
    four candidates and  $I_i'$  to replace  $I_i'$ .
}
return  $I_i'$ 

```

Fig. 6. The outline of the neighbor-join operator

Fig. 7. Four possible types are considered to connect city c and c' together. The results of Type I and Type II are obtained via the simple invert operation. The results of Type III and Type IV are constructed by applying a greedy method (see text) from two disjoint subtours.

B. NEIGHBOR-JOIN OPERATOR

The neighbor-join (NJ) operator constructs a new solution by stealing edges from other individuals in the population or by considering the geometric information. Although the NJ is applied only on the single solution, the offspring is generated considering both the neighborhood information and knowledge from other individuals. Thus, the NJ operator is a genetic operator combining with the characteristics of local search, mutation, and recombination.

The NJ is inspired by the inver-over mutation [11] and by analyzing the TSP search space [20]. The main difference between the inver-over mutation and the other mutations is that it inherits edges both from parent and from other individuals in the current population. According

to the analysis of the optimal tour of *att532*, we find that most of the links in the optimal tour of *att532* are the neighbor cities of each city.

Fig. 6 outlines the steps of the NJ. By given the input of an individual I_i and the search length L , the NJ applies L modifications from the start solution $I_i' = I_i$. In each modification, a city c is randomly selected from I_i' . With equal probability, a city c' is randomly selected either from the geometric nearest three neighbors of c or from an individual, which is randomly selected from the population. If the city c' does not connect to the city c in I_i' , to align c and c' together generates four possible types shown in Fig. 7. The NJ generates four candidates by sequentially executing each type once. The one with lowest fitness of these four candidates and I_i' are selected as the parent of next loop. Above steps are executed L times.

In the four candidates Types I and II are the simple invert operation to align c and c' together. For Types III and IV a greedy method is applied to merge two disjoint subtours into a valid solution. The greedy method works as follows: Let v_i represent a city, (v_i, v_j) , $i \neq j$, represents an edge, and $w(v_i, v_j)$ be the edge length of (v_i, v_j) . At the same time, let (v_r, v_{r+1}) and (v_s, v_{s+1}) be the edges of the subtour G_r and the subtour G_s , respectively. We find a pair of edges (v_r, v_{r+1}) and (v_s, v_{s+1}) to connect these two subtours, G_r and G_s , into a legal tour by maximizing the value of the following equation

$$w(v_r, v_{s+1}) + w(v_s, v_{r+1}) - w(v_r, v_{r+1}) - w(v_s, v_{s+1}) \quad \forall r, s; r \in G_r \text{ and } s \in G_s.$$

The new edges, (v_r, v_{s+1}) and (v_s, v_{r+1}) , are inserted to replace the original edges, (v_r, v_{r+1}) and (v_s, v_{s+1}) , to form the new solution.

C. AN EXAMPLE OF THE NEIGHBOR-JOIN

Let's illustrate a single iteration of the NJ on the following example. Assume that the current individual S_i is

$$I_i' = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$$

and the current city c is 3. If $\text{rand}() > 0.5$, another individual, $(0, 3, 7, 2, 9, 8, 4, 1, 5, 6)$, is randomly selected from the current population. Since the city 7 connect the city 3 in this individual and does not align next to the city 3 in individual I_i' , the cities 3 and 7 are decided to be connected. Four types of candidate are considered (see Fig. 7) and the one with lowest fitness function value is accepted. If type III is the lowest one and the gain of exchanging edge-pair $(v_2, v_8) - (v_4, v_5)$ by edge-pair $(v_2, v_4) - (v_5, v_8)$ is maximum, I_i' is updated as $(0, 1, 2, 4, 3, 7, 6, 5, 8, 9)$ if the fitness function value is better than the original I_i' .

Again, randomly assign the new city c as 5. If one, c , of the geometric three nearest cities is selected, assume, city 4 is determined to connect to city 5. Four candidates are considered:

- Type I : $(0, 1, 2, 4, 5, 6, 7, 3, 8, 9)$
- Type II : $(0, 1, 2, 6, 7, 3, 4, 5, 8, 9)$
- Type III : $(0, 1, 2, 8, 9, 0) (4, 3, 7, 6, 5, 4)$
- Type IV : $(0, 1, 2, 4, 5, 8, 9, 0) (3, 7, 6, 3)$

If type I is the one with lowest fitness function value, I_i' is updated as (0,1,2,4,5,6,7,3,8,9) if the fitness function value is better. Above steps are repeated for L times.

III. EXPERIMENTAL RESULTS

In this section, some well-known TSPs [23] are used to illustrate the robustness of our proposed approach. First, two important parameters, search length L and population size N , are discussed. Next, we compare the NJ with 2-opt and city swap to verify the effectiveness of the NJ. Finally, the proposed algorithm is compared with some other methods. The proposed algorithm was implemented in C++ and run on a Pentium III 500MHz personal computer with single processor.

A. PARAMETERS SETTINGS

The population size (N) and the search length (L) are main parameters in our algorithm. To decide the parameter values, we have tested various values of these two parameters on 15 TSP problems selected from TSPLIB[23].

Fig. 8 shows the relationship between the solution quality and the convergence time of our algorithm tested on problem *att532* with various population sizes (N). According to the experiments we set the population size roughly equal to the number of cities of a problem for the tradeoff between the solution quality and the convergence time.

Fig. 8. Relationship among population size, error rate, and convergence time of our algorithm tested on the *att532* problem.

Fig. 9. Comparisons of the proposed algorithm on 15 TSP problems with different search lengths 5, 20, and 50 based on the number of finding the optimal solution in 30 trails. The performance was worst when L is set to 5, and it has similar performance when the values of L are 20 and 50.

TABLE I.

Comparisons among neighbor-join, 2-opt and city-swap genetic operator on TSP problems taken from TSPLIB [23] based on the error rate, the number of optimal solutions found vs the number of trails, average generation, and average CPU time in 50 trails. In all cases, the error rate of NJ is always best (only 0.1% above the optimal tour length) and NJ is quicker than other operators for larger problems. Here the error (%) is defined as $\frac{\text{average} - \text{optimum}}{\text{optimum}}$.

Problem	Method	Error rate (%)	Optimal times	Gen(CPU)
att48	NJ	0.012	47/50	2.14(0.12)
	2-opt	0.490	5/50	19.00(0.266)
	cityswap	1.673	1/50	12.00(0.180)
eil51	NJ	0.097	29/50	6.46(0.20)
	2-opt	1.409	1/50	19.76(0.166)
	cityswap	2.859	0/50	11.16(0.180)
eil76	NJ	0.294	16/50	11.64(0.66)
	2-opt	4.368	0/50	35.86(0.500)
	cityswap	11.66	0/50	20.00(0.520)
eil101	NJ	0.550	7/50	17.78(1.44)
	2-opt	7.043	0/50	55.20(1.033)
	cityswap	21.93	0/50	35.16(1.180)
kroA200	NJ	0.311	1/50	29.22(9.28)
	2-opt	12.85	0/50	204.0(8.500)
	cityswap	93.32	0/50	220.9(13.06)
pr264	NJ	0.285	5/50	35.80(15.40)
	2-opt	26.93	0/50	286.5(17.16)
	cityswap	159.4	0/50	405.8(30.86)
lin318	NJ	0.894	0/50	47.20(30.06)
	2-opt	29.06	0/50	358.9(30.50)
	cityswap	173.9	0/50	493.8(47.08)

To understand the influences of the search lengths of the EAX and the NJ operators, we observed the performance with various search lengths (L). Fig. 9 shows the performance of the purposed algorithm on 15 TSP problems when the values of L are set to 5, 20, and 50, respectively. Each problem was tested 30 distinct trails. For each problem the proposed approach has the worst performance when L is 5 and the improvement of the solution quality is not significant when L exceeds 20. In practice, the longer L is, more time will spend. Therefore, L is set to 20 in this paper.

B. THE PERFORMANCE OF NJ

Table I summarizes the results of the NJ operator, 2-opt, and city swap on some benchmark problems. Each problem was tested 50 independent runs. In the table, the "Error rate (%)" is the relative error of the average of best value to the optimum in percentage; The column, "Opt/trail" denotes the number of finding the optimal solution in 50 trails; and "Gen (CPU)" denotes the values of the average generation and average CPU time required for finding the best tours, respectively. By the observation, the solution qualities of the NJ operator outperform the 2-opt and city swap. For all test instances the average value of solution quality stays within only 0.1% from the optimum. For the problem *lin318*, although the NJ is not able to find the optimum solution, it can still find near-optimal tours in each run and the average tour length that was only 0.894% above the optimum tour length. By comparing the famous 2-opt, these results show that the NJ is more robust than 2-opt for these testing TSPs.

C. THE IMPROVEMENT OF APPLYING NJ

Here we analyzed the influences of our method by cooperating different heuristics, such as 2-opt and NJ mutation, into the EAX operators. Table II shows the results of our methods with applying different operators for 15 TSP problems. According to the results, the methods combining 2-opt into EAX is limited for improving the solution quality. On the other hand, the method combining the NJ into EAX significantly promotes the solution qualities for all test problems. Although the NJ cannot be proved to be complementary with EAX, incorporating the NJ with the EAX indeed improves the solution qualities. The key novelty of the present work is the seamless ability of integrating global and local search mechanism through incorporation of a number of genetic operators each having unique search mechanisms.

D. COMPARISONS WITH OTHER METHODS

Following the discussions of the proposed algorithm (denoted as HGA) in the previous subsection, we compare the results of the proposed approach with the best-known results of four methods based on our best knowledge. Five TSP benchmark problems, including *lin318*, *pcb442*, *att532*, *rat783*, and *pcb3038*, shown in Table III are used because they have been widely used to compare the performance among algorithms. Our proposed algorithm was executed 30 trials for each problem.

TABLE II

Comparisons of our method with applying different local search methods. These methods are tested on 15 TSP problems based on the average CPU time (Time), the number of trails that optimal solutions found (Opt times), and average solution qualities in 30 trails (Error). Here the error (%) is defined as $\frac{\text{average} - \text{optimum}}{\text{optimum}}$.

All of three methods use the same parameters, including search length (*L*), population size (*N*), EAX crossover and stop criteria. The only difference is the use of **local search**.

Local search	No local search			NJ			2-opt		
	Time (Sec.)	Opt times	Error (%)	Time (Sec.)	Opt times	Error (%)	Time (Sec.)	Opt times	Error (%)
problem									
eil101	1.03	30	0.0000	1.50	30	0.0000	1.22	30	0.0000
kroA200	3.24	30	0.0000	4.82	30	0.0000	3.97	30	0.0000
lin318	14.54	26	0.0291	19.1	30	0.0000	15.65	27	0.0191
pcb442	39.91	30	0.0000	52.26	30	0.0000	43.57	30	0.0000
att532	97.93	7	0.0373	128.2	24	0.0090	105.4	11	0.0312
u574	64.65	30	0.0000	89.47	30	0.0000	76.89	30	0.0000
rat575	128.73	21	0.0043	172.9	28	0.0009	147.5	23	0.0031
u724	257.60	18	0.0122	335.4	27	0.0019	279.3	16	0.0059
rat783	375.45	30	0.0000	487.3	30	0.0000	396.7	30	0.0000
vm1084	810.12	18	0.0277	1045	21	0.0227	854.9	20	0.0099
pcb1173	1089	24	0.0050	1401	29	0.0005	1171	27	0.0005
u1432	2034	16	0.0145	2664	26	0.0080	2160	24	0.0038
vm1748	4295	19	0.0189	5663	27	0.0074	4633	21	0.0125
pr2392	9989	21	0.0049	13172	30	0.0000	10750	28	0.0009
pcb3038	22341	24	0.0071	29813	28	0.0045	24296	25	0.0006

TABLE III

Comparisons of the proposed algorithm (denoted as HGA) with other methods, including NGA [24], ACS [10], DGA [9], and EGA[1], on five TSP problems based on the best tour length, average tour length, and the standard deviation. "N/A" represents not available in original papers.

Problems/ (optimum)	Methods	Best (Error %)	Average (Error %)	$\frac{\sigma}{n}$
Lin318 (42029)	ACS	N/A	N/A	N/A
	DGA	42029(0.000)	42033.44(0.011)	1.35
	NGA	42029(0.000)	42029.00(0.000)	0.00
	EGA	42029(0.000)	42041.23(0.011)	1.81
	HGA	42029(0.000)	42029.00(0.000)	0.00
pcb442 (50778)	ACS	N/A	N/A	N/A
	DGA	50778(0.000)	50778(0.000)	0.00
	NGA	50778(0.000)	50778(0.000)	0.00
	EGA	50778(0.000)	50778(0.000)	0.00
	HGA	50778(0.000)	50778(0.000)	0.00
att532 (27686)	ACS	27693(0.000)	27718.20(0.112)	0.72
	DGA	27686(0.000)	27697.58(0.042)	0.48
	NGA	27686(0.000)	27695.61(0.035)	0.71
	EGA	27686(0.000)	27696.33(0.037)	0.62
	HGA	27686(0.000)	27688.49(0.009)	0.00
rat783 (8806)	ACS	8818(0.136)	8837.90(0.362)	0.19
	DGA	8806(0.000)	8806.00(0.000)	0.00
	NGA	8806(0.000)	8806.00(0.000)	0.00
	EGA	8806(0.000)	8806.00(0.000)	0.00
	HGA	8806(0.000)	8806.00(0.000)	0.00
pcb3038 (137694)	ACS	N/A	N/A	N/A
	DGA	137705(0.008)	137760.55(0.048)	4.28
	NGA	137695(0.001)	137765.02(0.052)	4.55
	EGA	137694(0.000)	137703.77(0.007)	0.93
	HGA	137694(0.000)	137700.19(0.004)	0.00

Table III summarizes the results of our method and four other approaches, including nature crossover genetic algorithm (NGA) [24], ant colony system (ACS) [10], distance-preserving crossover genetic algorithm (DGA) [9], and EAX genetic algorithm (EGA) [1]. NGA integrated nature crossover and LK local search [6]; ACS is an ant colony system cooperated with 3-opt operator; DGA combined the distance-preserving crossover and 3-opt; and EGA used the EAX crossover. These four approaches perform well on these test problems according to our surveys. The results of first three methods were directly summarized from original papers [9][10][24]. The best tour length, average tour length, and standard deviation of trails are used to measure the performance of comparative methods. The values in parentheses of the best and the average tour length represent the percentages of error defined as $\frac{\text{sol} - \text{optimum}}{\text{optimum}}$, where *sol* is the experimental value and *optimum* is the optimum of a TSP problem. The column " $\frac{\sigma}{\sqrt{n}}$ " shows the group standard deviation of trails where σ is the standard derivation and *N* is the number of cities.

Table III shows that our algorithm performs better and more robustly than comparative methods for testing problems. For each problem the proposed algorithm can find the best tour in almost each trial and the error rate is only 0.01% away from the optimal.

IV. CONCLUSION

This study demonstrates that combining suitable global and local search is an encouraging approach for solving TSPs. Experimental results verify that the proposed approach is able to generate efficient solutions for TSPs. The proposed neighbor-join operator, combining with local search, mutation, and recombination together, outperforms other genetic operators surveyed in this paper. Moreover, our approach can find out the near-optimal solutions with only about 0.03% above the optimum. We believe that the flexibility and robustness of our approach make it an effective tool for TSPs.

References

- [1]. Nagata, Y., Kobayashi, S. (1997) "Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem." In T. Back, editor, *Proc. of the 7th Int'l. Conf. on GAs*, pp. 450-457. Morgan Kaufmann.
- [2]. Alizadeh, F., Karp, R.M., Newberg, L.A., and Weissner, D.K. (1993) "Physical mapping of chromosomes: a combinatorial problem in molecular biology." In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 371-381.
- [3]. D. Sankoff and M. Blanchette. (1997) "The median problem for breakpoints in comparative genomics." *Computing and Combinatorics, Proceedings of COCOON '97. Lecture Notes in Computer Science 1276*, Springer Verlag, New York, pp. 251-263.
- [4]. Korostensky, C. and Gonnet, G. H. (2000) "Using traveling salesman problem algorithms for evolutionary tree construction." *Bioinformatics* Vol. 16 no. 7, pp. 619-627.
- [5]. Lin, S. (1965) "Computer solutions of the traveling salesman problem." *Bell Syst., J.*, vol. 23, pp. 2245-2269.
- [6]. Lin, S., and Kernighan B. (1973) "An effective heuristic algorithms for the traveling salesman problem." *Operations Research* Vol.21, pp.498-516.
- [7]. Kirkpatrick, S., Gelatt C. D., and Vecchi M. P. (1983) "Optimization by simulated annealing." *Sci*, vol. 200, pp. 671-680.
- [8]. Hopfield, J. J. and Tank, D. W. (1985) "Neural computation of decisions in optimization problems." *Biological Cybernetics*, vol. 52, pp. 141-152.
- [9]. Freisleben, B. and Merz, P. (1996) "New genetic local search operators for the traveling salesman problem." In *Proc. PPSN IV-4th Int. Conf. Parallel Problem Solving from Nature*. Berlin, Germany: Springer-Verlag, pp. 890-899.
- [10]. Dorigo, M., and Gambardella, L. M. (1997) "Ant colony system: A cooperative learning approach to the traveling salesman problem." *IEEE. Transaction on Evolutionary Computation*, vol.1, no.1, pp. 53-66.
- [11]. Tao, G., and Michalewicz, Z. (1998) "Inver-over Operator for the TSP." *Proceedings of the 5th Parallel Problem Solving from Nature, September 27-30, 1998, Lecture Notes in Computer Science*, pp.803-812.
- [12]. Mulhem, M., and Maghrabi, T. (1998) "Efficient convex-elastic net algorithm to solve the Euclidean traveling salesman problem." *Systems, Man and Cybernetics, Part B, IEEE Transactions*, Vol.284, pp.618-620.
- [13]. Zhenya, H., Chengjian, W., Bingyao, J., Wenjiang, P., and Luxi, Y. (1999) "A new population-based incremental learning method for the traveling salesman problem." *Proceedings of the 1999 Congress on Evolutionary Computation-CEC 99*, pp.1150-1156
- [14]. Goldberg, D. E. (1989) *Genetic algorithms in search, optimization & machine learning*. Reading, MA: Addison-Wesley.
- [15]. Baeck, T. (1996) *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, USA.
- [16]. Fogel, D. B. (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligent*. NJ:IEEE Press, Piscataway.
- [17]. Xiao, J., Michalewicz, Z., Zhang, L., and Trojanowski, K. (1997) "Adaptive evolutionary planner/navigator for mobile robots." *IEEE Trans. On Evolutionary Computation*, 1(1): pp.18-28.
- [18]. Hart, W. E. (1994) *Adaptive global optimization with local search*. PhD thesis, University of California, San Diego.
- [19]. Saravanan, N. and Fogel, D. B. (1997) "Multi-operator evolutionary programming: A preliminary on function optimization." In P. J. Angeline et al. editor, *Proc. 6th Annu. Conf. On Evolutionary Programming (Lecture Notes in Computer Science, vol. 1213)*, pp.215-222.
- [20]. Back, T., Hammel, U., and Schwefel, H. -P. (1997) "Evolution computation: Comments on the history and current states" *IEEE. Transactions on Evolutionary Computation*. pp. 3-17.
- [21]. Yen, J., Yip, J. C. and Pao, Y. H. (1998) "Combinatorial optimization with use of guided evolutionary simulated annealing." *IEEE. Trans. On Systems, Man, and Cybernetics -Part B*, 28(2): pp.173-191.
- [22]. Watson, J., Ross, C., Eisele, V., Denton, J., Bins, J., Guerra, C., Whitely, D., and Howe, A. (1998) "The traveling Salesrep Problem, edge assembly crossover, and 2-opt." In *Parallel Problem Solving from Nature V, A.E. Eiben et al, eds. Springer-Verlag*. pp.823-832.
- [23]. Reinelt, G. (1991) "TSPLIB- A Traveling salesman problem library." *ORSA Journal on Computing*, Vol.3, No.4, pp.376-384.
- [24]. J. Soonchul and M. Byung-Ro (2000) "The nature crossover for the 2D Euclidean TSP." in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1003-1010.