Original paper

Soft Computing 8 (2004) 689–697 © Springer-Verlag 2003
DOI 10.1007/s00500-003-0317-8

Some issues of designing genetic algorithms for traveling salesman problems

H.-K. Tsai, J.-M. Yang, Y.-F. Tsai, C.-Y. Kao

Abstract This paper demonstrates that a robust genetic algorithm for the traveling salesman problem (TSP) should preserve and add good edges efficiently, and at the same time, maintain the population diversity well. We analyzed the strengths and limitations of several well-known genetic operators for TSPs by the experiments. To evaluate these factors, we propose a new genetic algorithm integrating two genetic operators and a heterogeneous pairing selection. The former can preserve and add good edges efficiently and the later will be able to keep the population diversity. The proposed approach was evaluated on 15 well-known TSPs whose numbers of cities range from 101 to 13509. Experimental results indicated that our approach, somewhat slower, performs very robustly and is very competitive with other approaches in our best surveys. We believe that a genetic algorithm can be a stable approach for TSPs if its operators can preserve and add edges efficiently and it maintains population diversity.

Keywords Edge assembly crossover, Heterogeneous pairing selection, Genetic algorithm, Neighbor-join mutation, Traveling salesman problem

1 Introduction

The traveling salesman problem (TSP) is a well-known NP-hard optimization problem which requires the determination of the shortest route passing through a set of M cities under the condition that each city is visited exactly once. TSPs raise important issues because many problems in science, engineering, and bioinformatics fields, such as routing, scheduling problems, flexible manufacturing systems, physical mapping problems [1], and phylogenetic tree construction [14] can be formulated as TSPs.

A large number of approaches have been developed for solving TSPs. A very promising direction is the genetic algorithm (GA) [8]. It is based on the ideas borrowed from genetics and natural selection. A genetic algorithm is a generally adaptable concept for problem solving. It is especially well suited for solving difficult optimization problems, where traditional optimization methods are less efficient. However, general problem-independent GAs are not very efficient in solving TSPs, especially for large problems.

To further improve the GAs for TSPs, many approaches have been proposed. Among these approaches, designing TSPs-specific operators, incorporating domain-specific local searches, and keeping population diversity are considered as promising strategies. Designing TSPs-specified crossovers, such as cycle crossover [20], edge recombination crossover [29], maximally preserving crossover [18], edge assembly crossover [19] and inver-over operators [27], could raise the performances of GAs for solving TSPs. Incorporating domain-specific local search techniques into GAs [12–13, 17] possess both the global optimality of the GAs as well as the convergence of the local search. Keeping population diversity is useful to avoid premature convergences [5, 23].

Several issues responsible for improving the solution quality of GA for solving TSPs were discussed. Here we addressed these issues by analyzing the behaviors of some crossover and mutation operators on some well-known TSPs. We found that genetic algorithms for TSPs should at least have three mechanisms: powerful genetic operators which can preserve and add "good edges" (i.e., the edges in the optimal tour) as well as a mechanism to keep the population diversity. Based on these factors we designed a new genetic algorithm for TSPs. It was evaluated on 15 well-known traveling salesman problems [22] whose numbers of cities range from 101 to 13509 cities. The experimental results indicated that the solution quality of the proposed GA stays within 0.00043 from the optima for testing problems. It is more robust than comparative approaches.

The rest of this paper is organized as follows. In Sect. 2, some issues of designing GA for TSP are discussed by analyzing crossover operators, mutation operators, and the mechanism of keeping population diversity. Section 3

Published online: 24 November 2003

H.-K. Tsai (✉)
Department of Computer Science and Information Engineering,
National Taiwan University,
Taipei 106, Taiwan
e-mail: d7526010@csie.ntu.edu.tw

J.-M. Yang
Department of Biological Science and Technology &
Institute of Bioinformatics,
National Chiao Tung University, Hsinchu 30050, Taiwan

Y.-F. Tsai
Department of Information Management,
Chung Yu Junior College of Business Administration,
Keelung 201, Taiwan

C.-Y. Kao
Bioinformatics Center,
National Taiwan University, Taipei 106, Taiwan

```
begin
    initialize population P
    repeat
       P'=empty set;
       for i=1 to population size do
          Select two solutions s_a, s_b from P by pairing selection;
          for k = 1 to brood size (L) do
                s'_k = crossover (s_a, s_b)
          endfor
          s_best = the one with shortest length from s'_1, ..., s'_L
          for k = 1 to brood size (L) do
                s_best = mutation (s_best)
          endfor
          add s_best to P'
       endfor;
       totally replace P with P' with elitism
    until termination = true;
end;
```

Fig. 1. The pseudo code of the tested simple GA

introduces a new genetic algorithm based on the analysis of these factors and compares it to other famous approaches on some TSP benchmarks. Concluding comments are drawn in Sect. 4.

2
Some issues of designing GAs for TSPs

This section discusses some issues of designing GAs for TSPs by experiments. Some crossover and mutation operators are analyzed based on the mechanisms of preserving and adding good edges. The experimental results indicated that edge-based operators, such as edge assembly crossover and neighbor-join mutation, were able to meet these features. In order to keep the population diversity, a heterogeneous pairing selection was developed.

2.1
Analysis of crossover operators

Crossover operators for TSPs can be roughly divided into three categories: interval-preserving, position-based, and edge-based crossover. In an interval-preserving crossover, such as partially mapped crossover (PMX) [10], order crossover (OX) [6], order-based crossover (OBX) [26], and maximal preservative crossover (MPX) [18], a sub-path between two selected cities were copied from one parent to the offspring. Other cities were added into offspring according to the relative order of another parent. It executes just like typical two-point crossovers and repairs the solution based on parents' information. A position-based crossover preserves the relative position of cities in parents and acts like traditional uniform crossover. It attempts to create an offspring from the parents where every position is occupied by a corresponding element from one of the parent. The most representative ones are cycle crossover (CX) [20] and position-based crossover (PBX) [26]. An edge-based crossover, such as heuristic crossover (HX) [11], edge recombination crossover (ERX) [29], distance preserving crossover (DPX) [7], and edge-assembly crossover (EAX) [19], generates offspring by preserving edges in parents and adding new edges heuristically. Different methods employ different preserving and adding mechanisms.

We used a simple GA to analyze the performances of these crossover operators. Fig. 1 shows the pseudo code of the simple GA, where the wheel selection [4] is used to select the pairing chromosomes and total generation replacement with elitism is adapted to select next population. The population size is set equally to *N*. Termination criteria are that the number of function evaluation exceeds 10000 *N*, where *N* is the number of cities.

Table 1 shows the experimental results for each tested crossover with the simple GA (without mutation) on five small TSP benchmarks, including *att48*, *eil76*, *eil101*, *kroa150*, and *kroa200*. In Table I, the values in parentheses of problems represent the optimal tour length. The column "Length" represents the brood size (L) defined in Fig. 1. brood size is the number of solutions a genetic operator generated. Each entry is the average error defined as $\left(\frac{\text{average}}{\text{optimum}} - 1\right) * 100$ where average is the average values of the best solutions obtained by testing approaches executed for 20 trials. As observed, when the brood size equals to 1, all the edge-based crossovers, except ERX, outperform the interval-based and position-based crossovers. When the brood size is 20, the solution qualities of all operators are improved except CX, which is a position-based crossover. Another position-based crosssover, PBX, is not shown here beacuse the testing results is similar to CX. At the same time, the edge-based crossovers, including EAX, HX, DPX, and ERX are significantly better than those none edge-based crossovers. According to these experiments, we suggested that a good crossover operator should maintain the spirit of edge-based crossovers and execute for a larger brood size. The following describes that to preserve and add good edges is the main spirits of an edge-based operator.

In these surveyed edge-based crossover operators, EAX is the best one. We further discussed the results of these operators on several large TSPs. Figure 2 shows the results of these edge-based crossovers on eight TSP benchmarks, where the brood size is 20. Each problem is executed 20 trials. The relative error is defined as $\left(\frac{\text{average}}{\text{optimum}} - 1\right) * 100$ where the "*average*" is the average values of the best solutions obtained by these methods. EAX still outper-

Table 1. Comparison of nine crossovers with different search lengths using the simple GA (Fig. 1) on five TSPs benchmarks. The values in parentheses of problems represent the optimal tour length. "Length" denotes the brood size. Each entry is the average error defined as $(\frac{\text{average}}{\text{optimum}} - 1) * 100$ where average is the average values of the best solutions obtained by testing approaches.for independent 20 trials

Type	Operator	Length	*att48*	*eil76*	*eil101*	*kroa150*	*kroa200*
Edge based crossover	EAX	1	0.0263	0.0186	0.0000	0.0008	0.0136
		20	0.0000	0.0000	0.0000	0.0000	0.0051
	HX	1	1.2006	3.4944	4.8490	5.3314	8.1745
		20	2.3758	2.5465	3.8474	6.0153	7.1700
	DPX	1	6.2495	4.3123	14.3402	11.8323	10.8428
		20	4.0581	2.4535	8.6963	5.8355	3.7602
	ERX	1	43.6187	88.3086	178.6010	460.6364	635.6923
		20	9.2416	11.2825	12.5278	23.2024	28.5654
Interval based crossover	MPX	1	82.9592	149.8885	208.1240	460.6858	602.9120
		20	12.0342	11.8773	17.6471	50.0475	173.2563
	PMX	1	87.2168	104.6097	124.1812	226.7577	275.0259
		20	63.3628	82.9740	105.5803	190.0468	237.2678
	OX	1	80.3218	152.8903	214.7059	474.0624	609.4436
		20	12.8265	11.8680	40.5087	222.6821	359.7756
	OBX	1	96.3361	111.3941	148.5692	168.8045	105.8833
		20	49.0102	63.2528	72.0668	73.2160	44.7327
Position based crossover	CX	1	163.8700	188.3829	216.0890	392.3567	460.2087
		20	160.1402	183.7732	224.2925	401.2091	479.9527

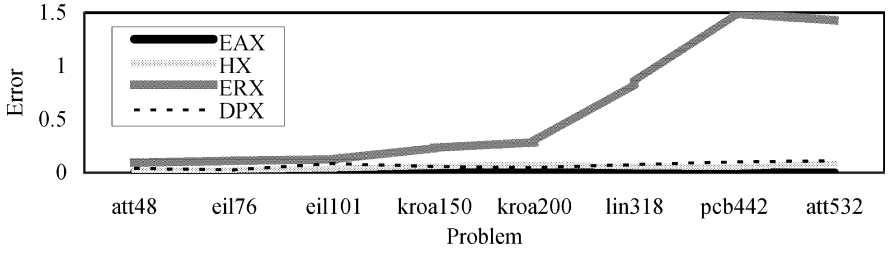

Fig. 2. Comparisons of four edge-based crossovers, including EAX, HX, ERX, and DPX using the simple GA (Fig. 1) on 8 TSP problems based on error defined as $(\frac{\text{average}}{\text{optimum}} - 1) * 100$ where the "*average*" is the average values of the best solutions obtained by these methods. EAX is the best one. The results of HX and DPX are similar, while the ERX is much worse when the number of cities increases

forms the others. HX and DPX get similar results, while ERX becomes worse when the number of cities increases.

In the following, we will analyze the characteristics of edge-based operators based on their abilities of preserving and adding edges to understand two issues: why edge-based operators are better than none edge-based operators and why EAX is the best one among these tested operators. As mentioned in the previous paragraph, edge-based crossovers generate offspring by preserving and adding edges heuristically.

Table 2 briefly summarizes the mechanisms of preserving and adding edges of these four edge-based crossovers. Observing the mechanism of preserving edges of these crossovers, we found DPX only preserves the common edges appeared in parents, HX preserves edges by concerning shorter edges from parents, and ERX inherits common edges from parents as much as possible. EAX inherits shorter edges from parents and considers the frequency of edges appeared in the current population. As to the aspect of adding edges, DPX modifies the intermediate solution by adding shorter edges, ERX and HX randomly add new edges to form feasible solutions, and EAX preserves edges heuristically and adds new edges with a greedy method which is analogous to a minimal spanning tree.

To further examine the mechanisms of preserving and adding edges, we measured the abilities of adding and preserving "good edges" (i.e., the edges in the optimal tour) of EAX, DPX, HX, and ERX operators. Figure 3 shows the results of these four operators tested on problem *att532.tsp* for adding and preserving "good edges" and average edge length. The average added "good edge" is defined as

$$\frac{\sum_i^n (\#\text{of edges added after the genetic operator applied on solution } i)}{n},$$

where n is the popluation size. The average preserved "good edge" is defined in the similar way. The average edge length is defined as

$$\frac{\sum_i^n f(s_i)}{nm},$$

where m is the number of cities, n is the popluation size, and $f(s_i)$ is tour length of individual s_i.

Table 2. The mechanisms of preserving and adding edges of edge-based crossovers, including EAX, DPX, HX, and ERX

Operator	Mechanisms	
	Preserving edge	Adding edge
EAX	Offspring inherits edges from parents according to the edge length and the frequencies appeared in the population	New edges are added into offspring to modify the intermediate solution by adding short edges through a spanning tree method
DPX	Only the common edges in both parents are passed into offspring	Intuitively adding short edges to make the intermediate solution feasible
HX	Offspring inherits edges from parents where the edge length is as short as possible	New edges are randomly added to become a feasible solution
ERX	Offspring inherits edges from parents as much as possible, but the maximum number of inherited edges is not guaranteed	New edges are randomly added to become a feasible solution

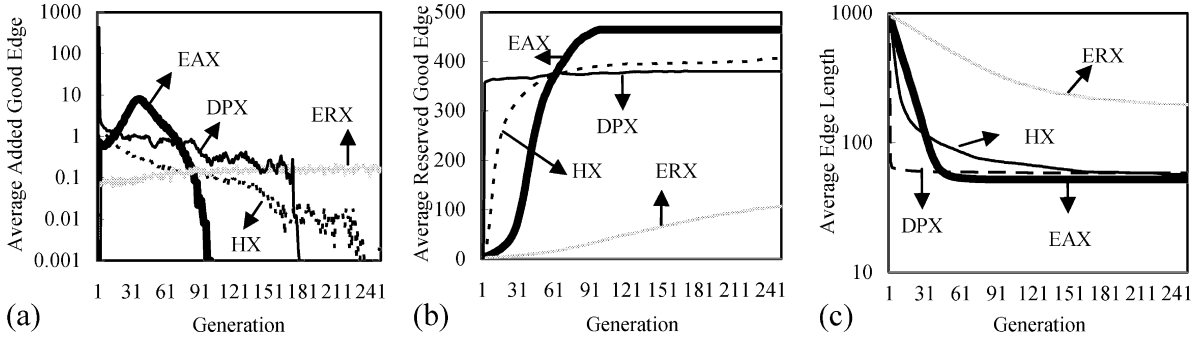

Fig. 3. Comparison of edge-based crossovers, EAX, DPX, HX, and ERX, on *att532.tsp* based on the average added "good edges" **a**, the average preserved "good edges" **b**, and the average edge length **c**

The population size and brood size are set to 500 and 20, respectively. Only the first 250 generations were observed. The similar results were obtained for the other problems. Figure 3(a) shows that EAX can continuously add more "good edges" than other operators in earlier stage (before the 70th generation). After the 70th generation, the values of the average preserving "good edges" (Fig. 3(b)) and average edge length (Fig. 3(c)) of EAX are better than other operators. HX and DPX have similar behaviors for these three factors and ERX has the worst values. These results are consistent with our previous discussions. In summary, a good crossover operator should be edge-based and possess good mechanism of adding and preserving "good edges". EAX is considered as a good crossover which meets these requirements.

2.2 Analysis of mutation operators

Some local search heuristics, such as 2-swap, 2-opt, 3-opt [15], neighbor-join [28] and Lin-Kernighan [16] have been used in GAs as mutation operations for TSPs. GAs and other evolutionary algorithms that use local search are often referred as Memetic algorithms. These mutations exchange some edges of parents to generate new children. Generally, the stronger the mutation (local search) used the better the performance of the GA.

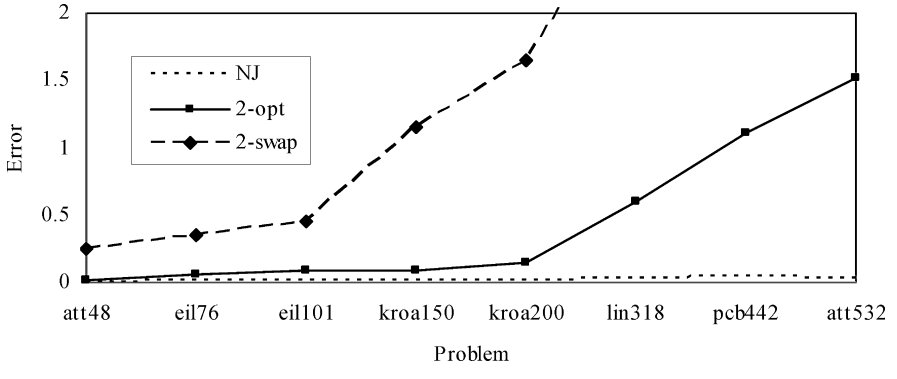

Fig. 4. Comparisons of three mutations, including 2-swap, 2-opt, and NJ using the simple GA (Fig. 1) on 8 TSP problems based on "error" defined as $(\frac{\text{average}}{\text{optimum}} - 1) * 100$ where *average* is the average values of the best solutions obtained by these methods. NJ is the best one. The results of 2-opt and 2-swap performed poorly as the number of cities increased

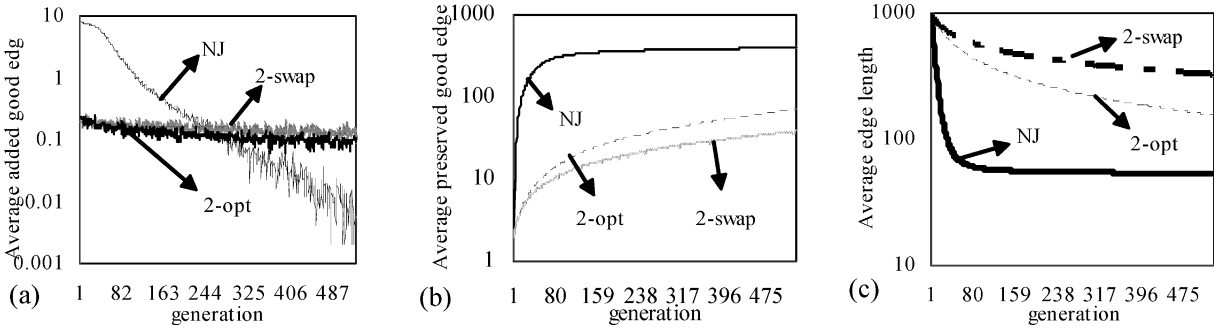

Fig. 5. Comparison of three mutations, 2-swap, 2-opt, and NJ, on *att532.tsp* based on the average added "good edges" a, the average preserved "good edges" b, and the average edge length c

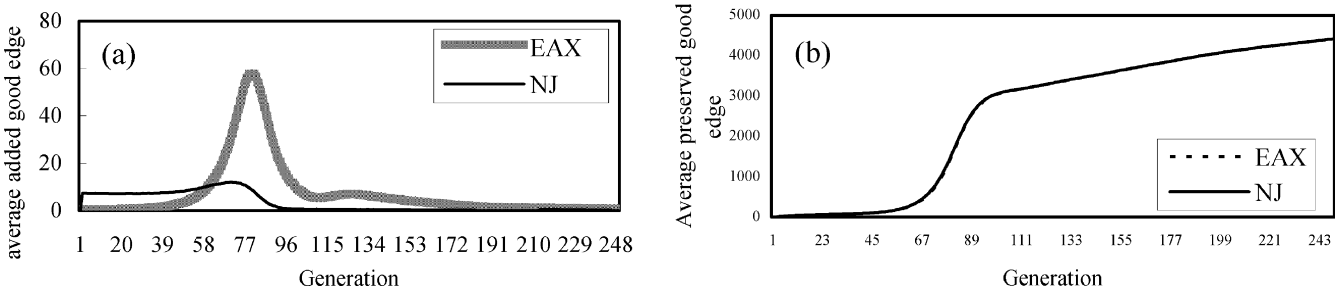

Fig. 6. The ability of adding and preserving good edges of EAX and NJ on the problem *fnl4461*. They seem be able to compensate each other on adding "good edge" and are similar on preserving "good edge"

Figure 4 shows the experimental results for three tested mutations, including NJ, 2-swap, and 2-opt, with the simple GA (Fig. 1) on eight small TSP benchmarks. These problems are including *att48*, *eil76*, *eil101*, *kroa150*, *kroa200*, *lin318*, *pcb442*, and *att532*. No crossover operators were adapted in this test set. The error is defined as $\left(\frac{\text{average}}{\text{optimum}} - 1\right) * 100$ where *average* was the average values of the best solutions obtained by these methods in 20 trials. NJ performed well. 2-opt became worse when the number of cities increased. 2-swap was the worst one.

The mechanisms of adding and preserving edges of these mutations vary. The 2-swap mutation arbitrarily changes two cities at a time and thus four edges are randomly removed and four edges are randomly added. For 2-opt and NJ, they exchange some edges if the solution is better. In each iteration 2-opt and NJ exchange 2 and 4 edges, respectively.

We used the same strategies for crossover operators to measure the abilities of adding and preserving "good edges" of NJ, 2-opt, and 2-swap mutations. Figure 5 shows the results of these operators tested on problem *att532.tsp* for adding and preserving "good edges" and average edge length. The similar results were obtained for the other problems. Figure 5(a) shows that NJ can add more "good edges" than other operators in earlier stage as well as the average preserving "good edges" (Fig. 5(b)) in later. 2-opt and 2-swap have similar behaviors in adding edges, however 2-swap is worse in preserving good edges. These results are consistent with our previous observations. In summary, a good mutation operator should possess good mechanisms of adding and preserving "good edges". NJ is considered as a good mutation operator which meets these requirements.

Experimental results show that by combining NJ with EAX, TSPs can be well solved [28]. To examine the complementary characteristics of EAX and NJ, we measure the abilities of adding and preserving "good edges" (i.e., the edges in the optimal tour) by EAX and NJ operators. Figure 6 shows the results of EAX and NJ for adding and preserving "good edges". Figure 6(a) shows that NJ can add more "good edges" than EAX in early stage, while EAX outperforms NJ in late stage. The abilities of preserving "good edges" of these two operators are similar (Fig. 6(b)). Although we cannot theoretically prove the seamless of combining EAX and NJ, they indeed compensate each other in adding and preserving "good edges" in our experiments.

2.3 Mechanism of keeping diversity

Keeping population diversity is another issue in GA for TSPs. Here we designed a new pairing selection, named heterogeneous pairing selection (HpS), to select two parents for crossover operators to reduce the premature convergence effect based on the edge similarity of a population.

The formulation and implementation of the HpS is described as follows: Let $\{s_1, s_2, \ldots s_N\}$ be the current population, $E(s_i)$ be the set of the edges of s_i, and $||E(s_i)||$ be the number of the edges of $E(s_i)$. The number of identical

Table 3. Comparisons of various pairing selections, including HpS, RpS, random, rank, wheel, and tournament selections, with the simple GA (Fig. 1) and EAX based on the "error" defined as $(\frac{\text{average}}{\text{optimum}} - 1) * 100$.where average is the average values of the best solutions obtained by testing approaches. These methods are tested on eight TSP problems based on the average solution qualities in 20 independent trials

	HPS	RPS	Random	Rank	Wheel	Tournament
att48	0.000000	0.000000	0.000000	0.000235	0.000000	0.000000
eil76	0.000000	0.000000	0.000000	0.000000	0.000000	0.000093
eil101	0.000000	0.000000	0.000130	0.000239	0.000000	0.000159
kroa150	0.000000	0.000002	0.000251	0.000004	0.000000	0.000008
kroa200	0.000000	0.000000	0.000342	0.000003	0.000510	0.000032
lin318	0.000000	0.000136	0.001006	0.000889	0.000834	0.001354
pcb442	0.000000	0.000000	0.000000	0.000000	0.000000	0.000007
Att532	0.000013	0.000224	0.000327	0.000354	0.000435	0.000502

edges $||T_{i,j}||$ of two individuals (s_i and s_j) is defined as $||T_{i,j}|| = ||E(s_i) \cap E(s_j)||$ For each individual s_i, let t_i be the average number of identical edges between s_i and the other individuals in the population $t_i = \frac{1}{N-1}\sum_{j=1,j\neq i}^{N} ||T_{i,j}||$, where N is the population size.

For the given individual s_i, the HpS selects s_i and another individual s_j with $||T_{i,j}|| \leq t_i$ for the EAX operator. This similarity-based mechanism is useful for keeping the population diversity. Our experimental results were consistent with this claim. In practical implementation, we used another method to calculate t_i due to the time complexity of getting all t_i through calculating $||T_{i,j}||$ is $O(N^2M^2)$. At the beginning of each generation, $F(e)$, the count of edge e appearances in the current population, is calculated in advance where $e \in \{E(s_1) \cup E(s_2) \cup \ldots \cup E(s_N)\}$. The sum of $||T_{i,j}||$ of s_i in the population, can be reformulated as

$$t_i = \frac{1}{N-1}\sum_{j=1,\, j\neq i}^{N} ||T_{i,j}|| = \frac{1}{N-1}\sum_{j=1}^{N} ||T_{i,j}|| - ||T_{i,i}||$$

$$= \frac{1}{N-1}\left(\sum_{e\in E(s_i)} F(e) - M\right)$$

$$= \frac{1}{N-1}\sum_{e\in E(s_i)} (F(e) - 1) \ .$$

Therefore, all t_i can be calculated in $O(NM)$ by looking up the pre-calculated table. Since the EAX crossover also uses the information $F(e)$, the extra effort of calculating t_i will be limited.

To show the performance of HpS, we compared HpS with five selections, including random pairing selection (RpS) [19], random selection [4], rank selection [3], roulette wheel selection [4], and tournament selection [9], on eight TSPs benchmarks. The simple GA (Fig. 1) and EAX were used and the search length was set to 20. Table 3 shows that HpS is better than the others in these eight tested problems with the same function evaluations. For *att532*, EAX with HpS can find the optimal solution for 19 times in 20 independent runs. For the other problems, EAX with HpS can always found the optima in each trial. In summary, we have demonstrated that the HpS is able to improve the solution quality for the EAX through maintaining of population diversity and provision of a good pairing scheme.

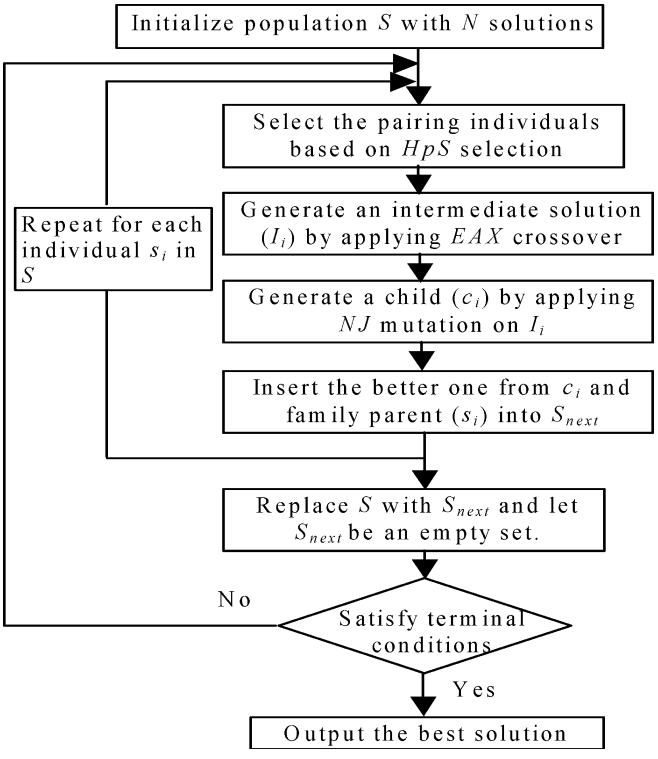

Fig. 7. Overview of our proposed genetic algorithm

3 System and results

Based on the discussion of section II, we investigated a new genetic algorithm combining the edge assembly crossover (EAX), neighbor-join mutation (NJ), and the heterogeneous pairing selection (HpS). Figure 7 shows the main steps of the proposed approach. N solutions are randomly generated as the initial population. After evaluating the fitness, each solution in the population sequentially uses the HpS to select itself (s_i) and another individual from the population based on the edge similarity. These two individuals become the parents of the EAX which generates only one intermediate offspring (I_i). The NJ mutation is then executed L times to generate a child (c_i) by refining the intermediate solution I_i. In each

Table 4. The experimental results of our method on 15 TSP problems based on the average time (sec), generation, tour length, and the optimal times our approach found the optima. The error is defined as where average is the average values of the best solutions obtained by testing approaches

Problem	Time	Generation	Average tour length (error)	Optimal times
eil101.tsp (629)	0.5	17.5	629 (0.000000)	20
Kroa200.tsp (29368)	4	35.9	29368 (0.000000)	20
Lin318.tsp (42029)	15	49.8	42029 (0.000000)	20
Pcb442.tsp (50778)	33	43.7	50778 (0.000000)	20
Att532.tsp (27686)	97	73.05	27686 (0.000000)	20
u574.tsp (36905)	106	70.75	36905 (0.000000)	20
rat783.tsp (8806)	376	83.25	8806 (0.000000)	20
vm1084.tsp (239297)	955	100.25	239304.8 (0.000033)	19
Pcb1173.tsp (56892)	1177	105.6	56892.5 (0.000009)	18
u1432.tsp (152970)	1875	82.8	152970 (0.000000)	20
pr2392.tsp (378032)	4968	147.5	378032 (0.000000)	20
Pcb3038.tsp (137694)	11617	178.2	137694 (0.00000)	20
Fnl4461.tsp (182566)	50033	255.4	182570.3 (0.000024)	16
rl5915.tsp (565530)	78527	195	565530.5 (0.000001)	19
Usa13509.tsp (19982859)	256186	452	19991528 (0.000434)	0

Table 5. Comparisons of our method (HSGA) with six methods, ant colony system (ACS) [25], the voronoi crossover genetic algorithm (VGA) [24], the compact genetic algorithm (CGA) [5], iterative Lin-Kernighan (ILK) [12], chained Lin-Kernighan (CLK) [2], and Tabu search with Lin-Kernighan (LK) [30] on eight larger TSP problems based on the average tour length. The length of the optimal tour of each problem is in brackets. Here the error is defined as where average is the average values of the best solutions obtained by testing approaches. N/A means the result is *not available*

Problem	HSGA	ACS	VGA	CGA	ILK	CLK	Tabu with LK
lin318 (42029)	42029 (0.000000)	42029 (0.000000)	42029 (0.000000)	42029 (0.000000)	N/A	N/A	N/A
att532 (27686)	27686 (0.000000)	27718.2 (0.001163)	27686.7 (0.000025)	27686 (0.000000)	N/A	N/A	N/A
rat783 (8806)	8806 (0.000000)	8837.9 (0.003622)	N/A	8806 (0.000000)	N/A	N/A	N/A
vm1084 (239297)	239304.8 (0.000033)	N/A	N/A	N/A	239349 (0.000217)	239301 (0.000017)	240238 (0.003932)
pcb1173 (56892)	56892.5 (0.000009)	N/A	N/A	N/A	56897 (0.000088)	56984 (0.001617)	57290
u1432 (152970)	152970 (0.000000)	N/A	N/A	N/A	153122 (0.000994)	153328 (0.002340)	153727 (0.004949)
pr2392 (378032)	378032 (0.000000)	N/A	N/A	N/A	378597 (0.001495)	379629 (0.004225)	380486 (0.006492)
pcb3038 (137694)	137694 (0.00000)	N/A	137706.8 (0.000093)	N/A	137861 (0.001213)	138055 (0.002622)	138893 (0.008708)
fnl4461 (182566)	182570.3 (0.000024)	N/A	182605.9 (0.000219)	182578.4 (0.000068)	182814 (0.001358)	182840 (0.001501)	184373 (0.009898)
frl5915 (565530)	565530.5 (0.000001)	N/A	N/A	565554.0 (0.000042)	565625 (0.000168)	568570 (0.005375)	570650 (0.009053)
usa13509 (19982859)	19991528 (0.000434)	N/A	N/A	19991585 (0.000437)	20015598 (0.001638)	20022550 (0.001986)	20160648 (0.008897)

pair of s_i and c_i, the one with the better solution survives where $1 \leq i \leq N$. These N solutions become the new population of the next generation. Note that the probabilities of both the crossover and the mutation operators are 1.0. Our algorithm is terminated when one of the following criteria is satisfied: 1) the maximum preset search time is exhausted, 2) all individuals of a population represent the same solution, or 3) all of the children generated in five continuous generations are worse than their respective family parents.

We tested the proposed approach on 15 TSP benchmark problems whose numbers of cities range from 101 to 13509 cities and executed on a Pentium IV 1 GHz personal computer with single processor. Each problem was tested for 20 independent runs. According to the experiments, the population size was set to the number of cities of a TSP whose number of cities is smaller than 1000 and set to the half of the number of cities of a TSP whose number of cities is larger than 1000 for the tradeoff between solution quality and convergence time. For *usa13509*, the population size was set to 2000 due to the memory constraint.

Table 4 shows the experimental results of our approach tested on 15 problems. The values in parentheses of problem represent the optimal tour length. The values in parentheses of the average tour length represent the percentage of error defined as $(\frac{\text{average}}{\text{optimum}} - 1) * 100$, where *average* is the experimental value and *optimum* is the optimum of a TSP problem. The "optimal times" means the times our approach found the optima in 20 trials for a TSP problem.

Table 4 shows the proposed approach which performs robustly for these tested problems. Except usa13509, our approach is able to find the optimum of the 14 tested benchmarks for at least 16 times in 20 trials. All the average tour length above the optima is within 0.00043. Our approach, somewhat slower, is able to find the optima and is stable for all tested problems.

To show the robustness of our proposed approach on large TSPs, we compared it with some methods, including the ant colony system (ACS) [25], the voronoi crossover genetic algorithm (VGA) [24], the compact genetic algorithm (CGA) [5], iterative Lin-Kernighan (ILK) [12], chained Lin-Kernighan (CLK) [2], and Tabu search with Lin-Kernighan [30], as shown in Table 5. These six approaches performed well on these test problems according to the original papers and the results on "8th DIMACS Implementation Challenge: The Traveling Salesman Problem".

Table 5 shows that our method outperforms these surveyed approaches in these testing problems. It is able to find the optimum and the average solution quality is within 0.00043 above the optima value for each testing problem although the proposed GA is somewhat slower than these approaches. ILK is the fastest approach among these approaches and the testing result is slightly better than other surveyed approaches. For the larger problem such as *usa13509*, ILK is about 30 times faster than the proposed method with population size equal to 2000. Fortunately, when we set the population size to 100, the running time for our method is about the same as ILK and the average tour length is 20014159 (0.001566) which is slightly better than ILK.

4 Conclusion

This study demonstrates that GA can be a stable approach for TSPs via designing their components. From our analysis and experiences, we suggest that a genetic algorithm for TSPs should consist of local search strategies and maintain population diversity as well as implement the mechanisms of preserving good edges and inserting new edges into offspring. In our proposed approach, the heterogeneous edge selection keeps the population diversity; the neighbor-join mutation is local search strategies. Our experiments indicated that the edge assembly crossover and the neighbor-join mutation are able to preserve good edges and add new edges. These strategies seem to be able to closely cooperate with each other to improve the overall search performance.

Experiments on 15 benchmark TSPs verify that the proposed approach is robust and is very competitive with algorithms of our best surveys. Our approach is able to find optimum and stable solutions for all testing TSPs; specifically, it finds out the optimum over 16 times in 20 independent runs for problems smaller than 6000 cities. For larger problem usa13509, the average solution is only 0.00043 above the optima. We believe that a GA with these claimed issues will become a robust tool for TSPs and potential applications.

References

1. **Alizadeh F, Karp RM, Newberg LA, Weisser DK** (1993) Physical mapping of chromosomes: a combinatorial problem in molecular biology. Proceeding the Fourth ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 52–76
2. **Applegate D, Bixby R, Chvatal V, Cook WJ** (1999) Finding Tours in the TSP. Tech. Rep. TR99-05, Dept. Comput. Appl. Math., Rice Univ., Houston, TX 77005
3. **Bäck T, Hoffmeister F** (1991) Extended selection mechanisms in genetic algorithms. In: Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA), pp. 92–99
4. **Baker JE** (1987) Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA), pp. 14–21
5. **Baraglia R, Hidalgo JI, Perego R** (2001) A hybrid heuristic for the traveling salesman problem. IEEE Transactions on Evolutionary Computation **5**: 613–22
6. **Davis L** (1985) Applying adaptive algorithms to epistatic domains. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 162–164
7. **Freisleben B, Merz P** (1996) New genetic local search operators for the traveling salesman problem. In: Parallel Problem Solving from Nature IV (PPSN IV), pp. 890–899
8. **Goldberg DE** (1989) Genetic Algorithms in Search, Optimization & Machine Learning. Reading, MA: Addison-Wesley
9. **Goldberg DE, Deb K** (1991) A comparative analysis of selection schemes used in genetic algorithms. Foundations of Genetic Algorithms, pp. 69–93
10. **Goldberg DE, Lingle R Jr** (1985) Alleles, Loci and the TSP. In: Proceedings of the First International Conference on Genetic Algorithms and Their Applications, pp. 154–159
11. **Grefenstette JJ** (1987) Incorporating problem specific knowledge into genetic algorithms. Genetic Algorithms and Simulated Annealing pp. 42–60
12. **Johnson DS, McGeoch LA** (1997) The Traveling Salesman Problem: A Case Study in Local Optimization. In: Aarts EHL, Lenstra JK (eds), Local Search in Combinatorial Optimization, John Wiley and Sons, Ltd., pp. 215–310
13. **Jung S, Moon BR** (2000) The nature crossover for the 2D Euclidean TSP. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 1003–1010
14. **Korostensky C, Gonnet GH** (2000) Using traveling salesman problem algorithms for evolutionary tree construction. Bioinformatics **16**: 619–627
15. **Lin S** (1965) Computer solutions of the traveling salesman problem. Bell Systems Technical Journal **23**: 2245–2269
16. **Lin S, Kernighan B** (1973) An effective heuristic algorithms for the traveling salesman problem. Operations Research **21**: 498–516
17. **Merz P** (2002) A comparison of memetic recombination operators for the traveling salesman problem. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 472–479
18. **Mühlenbein H, Gorges-Schleuter M, Krämer O** (1988) Evolution algorithms in combinatorial optimization. Parallel Computing **7**: 65–85
19. **Nagata Y, Kobayashi S** (1997) Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In: Proceeding of the Seventh International Conference on Genetic Algorithms (ICGA), pp. 450–457
20. **Oliver IM, Smith DJ, Holland JRC** (1987) A study of permutation crossovers on the TSP. In: Genetic Algorithm and Their Applications: Proceedings of the Second International Conference pp. 224–230
21. **Padberg M, Rinaldi G** (1987) Optimization of a 532-city symmetric traveling salesman problem by branch and cut. Operation Research Letters **6**: 1–7
22. **Reinelt G** (1991) TSPLIB- A Traveling salesman problem library. ORSA Journal on Computing **3**(4): 376–384

23. **Ronald S** (1995) Preventing Diversity Loss in a Routing Genetic Algorithm with Hash Tagging. Complexity International volume 2 ISSN1320–0683
24. **Seo D, Moon BR** (2002) Voronoi quantized crossover for traveling salesman 0problem. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 544–552
25. **Stützle T, Dorigo M** (1999) ACO algorithms for the traveling salesman problem. Evolutionary Algorithms in Engineering and Computer Science, pp. 163–183
26. **Syswerda G** (1991) Schedule optimization using genetic algorithm. Handbook of Genetic Algorithms pp. 332–349
27. **Tao G, Michalewicz Z** (1998) Inver-over operator for the TSP. Parallel Problem Solving from Nature V (PPSN V), pp. 803–812
28. **Tsai HK, Yang JM, Kao CY** (2001) Solving Traveling Salesman Problems by Combining Global and Local Search Mechanisms. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC), pp. 1290–1295
29. **Whitely D, Starkweather T, D'Ann Fuquay** (1989) Scheduling problems and traveling salesman: The genetic edge recombination operator. In: Proceeding of the Third International Conference on Genetic Algorithms (ICGA), pp. 133–140
30. **Zachariasen M, Dam M** (1995) Tabu Search on the Geometric Traveling Salesman Problem. In: Proceedings from Metaheuristics International Conference, pp. 571–587