Vol. 43 No. SIG 10(TOM 7) IPSJ Transactions on Mathematical Modeling and Its Applications Nov. 2002

Regular Paper

Greedy Genetic Algorithms for Symmetric and Asymmetric TSPs

Hung Dinh Nguyen,[†] Ikuo Yoshihara,[††] Kunihito Yamamori[††]
and Moritoshi Yasunaga[†††]

This paper presents new enhancements to a multi-population GENITOR-type Genetic Algorithm (GA) for solving symmetric and asymmetric Traveling Salesman Problems (TSPs). First, improvements to the greedy subtour crossover are proposed so that it works more effectively at the stage of highly fit individuals. Next, local search heuristics are combined with GA to compensate for its lack of local search ability. The powerful Lin-Kernighan heuristic is used for symmetric TSPs and the fast 3-Opt heuristic is used for asymmetric TSPs. Various symmetric and asymmetric TSP benchmarks taken from the TSPLIB are used to validate the method. Experimental results show that the proposed method can find optimal solutions for problems ranging in size up to 3795 cities in a reasonable computing time. From the viewpoint of quality of solution, these results are the best so far obtained by applying GA to the TSP.

1. Introduction

The Traveling Salesman Problem (TSP) is one of the most important and representative combinatorial optimization problems, because it is simple but fundamental and widely applicable. The TSP has applications in many fields such as vehicle routing, robot control, and crystallography etc. For example, the problems of collecting coins in automatic vending machines, scheduling jobs in a single machine, and ordering drill holes in a circuit board can all be formulated as TSPs.

The TSP is simple to state. In the TSP, locations of all the cities are given and the salesman's task is to find the minimum cost route connecting them all, with each city visited only once and return to the city of origin. The cost here can be distance, time, or money etc. If all the costs between any two cities are equal in both directions the problem is called a symmetric TSP (STSP), otherwise it is an asymmetric TSP (ATSP).

Since the TSP is an NP-complete problem, any method of finding the true optimal solution has a search space that expands no less than exponential order of the number of cities. Therefore, they are impractical for large problems. So far, many efforts have been concentrated on the development of practical algorithms that do not always aim at finding the best solution but at quickly finding a reasonably good solution. They can be roughly divided into two categories: local search algorithms that use problem-specific knowledge and global search algorithms that are problem independent. One of the global search algorithms is Genetic Algorithm (GA), which is an optimization and search method inspired by the evolutionary process of nature[1),2)].

There have been a lot of attempts to apply GA to solve the TSPs. One of the important parts of GA is the crossover (recombination) operator. Many crossover operators[3)] have been proposed for solving the TSP, for example, partially matched crossover (PMX)[4)], cycle crossover (CX)[5)], order crossover (OX)[6)], maximal preservative crossover (MPX)[7)], edge recombination (ER)[8)~14)], greedy subtour crossover (GSX)[15),16)], distance preserving crossover (DPX)[18)], edge assembly crossover (EAX)[21)], and natural crossover[22)].

Although GA is a robust search algorithm suitable for problems having huge search spaces such as the TSP, it is often outperformed by local search heuristics when applying to the TSP. This is because GA is lack of local search ability, or in other words, it does not utilize the problem-specific knowledge. It seems that the only way to develop a high performance GA-based method for the TSP is to incorporate problem-specific knowledge into GA. All of the best-so-far GA-based methods for the TSP used problem-specific knowledge[19)~22)].

Merz and Freisleben[19)] proposed the Genetic Local Search, which stresses on the use of the powerful Lin-Kernighan heuristic. The GA part of their algorithm, however, applied an ex-

† Graduate School, Miyazaki University
†† Faculty of Engineering, Miyazaki University
††† Institute of Information Sciences and Electronics, University of Tsukuba

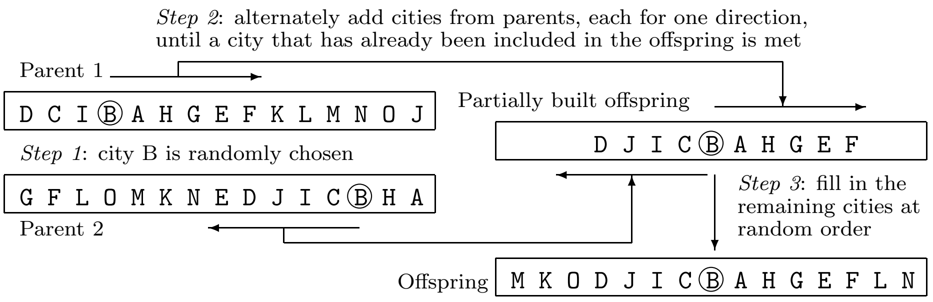
Step 2: alternately add cities from parents, each for one direction, until a city that has already been included in the offspring is met
Parent 1
D C I B A H G E F K L M N O J
Step 1: city B is randomly chosen
G F L O M K N E D J I C B H A
Parent 2
Partially built offspring
D J I C B A H G E F
Step 3: fill in the remaining cities at random order
Offspring
M K O D J I C B A H G E F L N

Fig. 1 The original GSX (GSX-0).

tremely small population size (20 individuals for the STSP). On the other hand, Gorges-Schleuter[20)] proposed a method that uses a powerful hierarchy population GA with a relatively weak local search heuristic. She compared her method with Merz and Freisleben's method and pointed out that for large problems, the strategy of using many times a weaker but faster local search can be as effective as using few times of a more powerful but slower local search. Unfortunately, both methods did not clearly demonstrate the potential of combining GA with local search as expected. Their results reported for large problems are still not convincible.

Nagata and Kobayashi[21)] incorporated the TSP-specific knowledge directly into the crossover operator, which they named edge assembly crossover. They did not use any other local search heuristics. In order to find high quality solutions for large problems, however, their method needs a big population size (1,200 individuals for a 2,392-city problem), which makes their method rather slow.

Recently, Jung and Moon[22)] proposed a GA-based method that uses the natural crossover and involves the Lin-Kernighan heuristic for the STSP. However, their crossover is only applicable for Euclidean problems and their method does not have impressive quality of solutions.

The goal of our research is to develop a more effective GA-based method for the TSPs. To achieve this goal, we first propose new improvements to the GSX operator to make our GA more effective at the stage of highly fit individuals. GA is then combined with heuristics to compensate for its lack of local search ability. Local search heuristics for both STSP and ATSP are investigated and a way of incorporating them into GA is proposed. Various problems from the TSPLIB[26)], which contains standard benchmarks for testing TSP algorithms, are used to validate the method.

The paper is organized as follows. Section 2 presents the improvements of GSX. Section 3 describes the hybrid GA. In Section 4, experiments and results for TSP benchmarks are presented. Section 5 concludes the paper and briefs works for future research.

2. Improvements of the Greedy Subtour Crossover

Desirable characteristics of the optimal solution of the TSP are embedded in subtours, which consist of connecting relations of edges. Therefore, an appropriate crossover for the TSP should transmit as much connecting relations of edges as possible from parents to offspring. The GSX is designed for such purpose. In this section, the conventional versions of GSX operator and its newly improvements are presented.

2.1 The Original Greedy Subtour Crossover (GSX-0)

The original GSX (GSX-0) was proposed by Sengoku and Yoshihara[15)]. It consists of three steps as follows.

Step 1: a city is randomly chosen as the *crossover city* and copied to the offspring.

Step 2: cities from parents are alternately copied to the offspring, with each parent being extended from one direction of the crossover city, until a city that has already been included in the offspring is met.

Step 3: all of the remaining cities are filled in the offspring at random order.

Figure 1 shows an example of the GSX-0 operator. First, city B is randomly chosen as the crossover city and copied to the offspring. Next, other cities are alternatively copied from parents, with parent 1 from the right and parent 2 from the left of city B. In this example, first city A from parent 1 and city C from parent 2 are added to the partially built offspring. The process is repeated until city F from par-

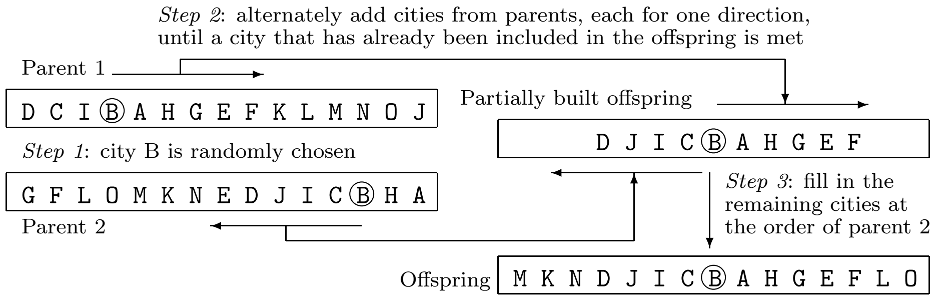
Step 2: alternately add cities from parents, each for one direction, until a city that has already been included in the offspring is met
Parent 1
D C I B A H G E F K L M N O J
Step 1: city B is randomly chosen
G F L O M K N E D J I C B H A
Parent 2
Partially built offspring
D J I C B A H G E F
Step 3: fill in the remaining cities at the order of parent 2
Offspring
M K N D J I C B A H G E F L O

Fig. 2 Takeda's improved GSX (GSX-1).

ent 1 is added to the partially built offspring and city E from parent 2 is attempted. However, since city E has already been included in the offspring, the above process is stopped without adding city E to the offspring. Finally, the remaining cities are filled to the offspring at random order as shown in Fig. 1. Note that there are three *foreign* edges (edges are inherited from neither parent) KO, OD, and LN in the offspring.

2.2 Takeda's Improved GSX (GSX-1)

Takeda, et al.[16] modified only the final step of the operator. They proposed to fill the remaining cities at the order that they appear in one of the parent, skipping all the cities that have been included in the offspring. In the above example, since before the stop of adding cities the algorithm attempted to add city E from parent 2, so the remaining cities will be filled to the offspring from the left according to the order that they appear in the parent 2, skipping all cities that have already been included in the offspring. That order is "N K M O L" and the offspring becomes as shown in **Fig. 2**. There is only one foreign edge ND in the offspring.

Despite of minor modification, the effectiveness of the operator is improved fairly well, especially when combining GA with local search heuristics. The reason why GSX-1 is superior to GSX-0 can be explained as follows. When combining with local search heuristics, the population rapidly converges to a stage of highly fit individuals. At this stage, the edges that compose the parents should involve in subtours with desirable characteristics. Therefore, even a single randomly created foreign edge may degrade the fitness of the offspring. The improved GSX-1 fills in the remaining cities in the order of one of the parent so it can reduce the number of foreign edges introduced into the offspring, and therefore it is able to produce better offspring at this stage.

2.3 Newly Improved GSX (GSX-2)

We observed that the GSX-1 operator has a drawback when being applied to the STSP. For the STSP, there are two ways of representing a tour (an inversion of a tour represents also that tour). For example, tour "A B C D E" and tour "A E D C B" are the same for the STSP. Therefore, the population may contain individuals that share a common subtour but in inverse order. Let consider the case that parent 1 partially contains the subtour "A B C" and parent 2 contains the subtour "C B A", and B is chosen as the crossover city. In this case GSX-1 will stop step 2 immediately after city C of parent 1 is added to the partially built offspring. The offspring is then filled in with the remaining cities at the order of parent 2, thus it is almost identical to parent 2. The consequence is that the operator's ability of creating diversity for the population is degraded.

Our solution to this problem is as follows. Before adding cities from both parents (step 2), the direction of adding cities from parent 2 is decided according to the crossover city and its four neighbors. Let assume that parent 1 and parent 2 are as follows.

Parent 1: ". . . x B y . . .",
Parent 2: ". . . p B q . . .".

Where B is the crossover city, x and y are its two neighbors in parent 1, and p and q are its two neighbors in parent 2. The following fragment of pseudo-code will decide the *direction_2* for adding cities of parent 2. Note that GSX-1 always adds cities of parent 2 from the left side of the crossover city.

```
if (x == q || y == p) then
  direction_2 = right;
else
  direction_2 = left;
```

As we know that random foreign edges introduced in the offspring may degrade the fitness of

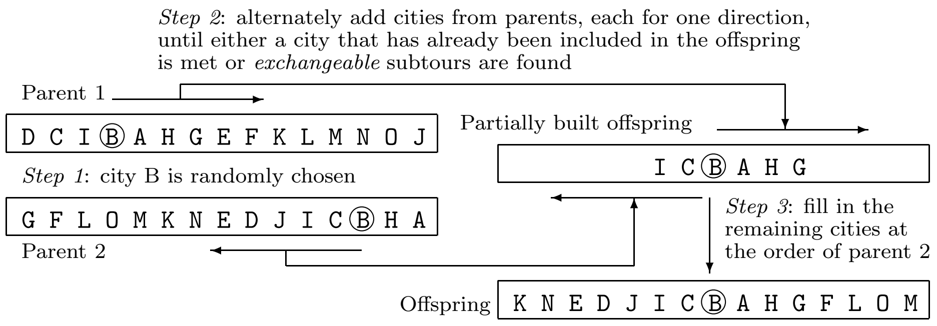
Step 2: alternately add cities from parents, each for one direction, until either a city that has already been included in the offspring is met or *exchangeable* subtours are found
Parent 1
D C I Ⓑ A H G E F K L M N O J
Step 1: city B is randomly chosen
G F L O M K N E D J I C Ⓑ H A
Parent 2
Partially built offspring
I C Ⓑ A H G
Step 3: fill in the remaining cities at the order of parent 2
Offspring
K N E D J I C Ⓑ A H G F L O M

Fig. 3 Newly improved GSX (GSX-2).

the offspring, we propose another modification in order to reduce further the number of foreign edges. This modification is applicable to both STSP and ATSP. Only the step 2 is modified. Cities are alternately added from the parents to both sides of the crossover city until either a city that has been included in the offspring is met or *exchangeable* subtours are found. Exchangeable subtours are defined as two subtours, one from each parent, that consist of the same set of cities but in different order and have the same two end cities.

In our example, the subtour "B A H G" of parent 1 and the subtour "B H A G" of parent 2 are exchangeable subtours. Therefore, the operator will stop adding cities from both parents after city G of parent 1 is added. Then the offspring is filled in with the remaining cities according to the order that they appear in parent 2, starting from J. The final offspring is as shown in **Fig. 3**. We can easily see that if exchangeable subtours are found, all the edges of the offspring are inherited from parents (i.e., there is no foreign edge in the offspring). When properly implemented, checking for exchangeable subtours can be done in a time order of $O(N)$. Therefore, the time complexity of the newly improved crossover remains $O(N)$, which is the same with GSX-0 and GSX-1.

3. The Hybrid GA

3.1 Hybridizing GA with Heuristics

In TSP applications, many studies have indicated that pure GA methods are outperformed by the conventional heuristics, particularly when the problem size increases. Therefore, many researchers have tried to combine local search heuristics with GAs to gain better performance[15)~22)]. The hybrid method combines the global search ability of GA with the local search ability of heuristics, thus it possibly becomes a more robust search algorithm. In our hybrid GA, a tour construction heuristic is used to generate the first population and a tour improvement heuristic is used after the crossover and mutation operators to improve the quality of individuals during the search. The pseudocode in **Fig. 4** illustrates our hybrid GA.

```
Procedure HybridGA
Begin
 For each subpopulation do
  Initialize subpopulation by tour
  construction heuristic;
 Repeat {
  For each subpopulation do {
   If (rand() < mutation_rate) {
    Select one parent p using linear ranking;
    Mutation(p, c);
   } Else {
    Select two parents p1, p2 using linear
    ranking;
    Crossover(p1, p2, c);
   }
   Tour_improvement_heuristic(c);
   Replace c to the worst parent if fitter;
  }
  At predefined migration_interval do
   Migration between subpopulations
 }
 Until converged;
End
```

Fig. 4 Pseudo-code of the hybrid GA.

The mutation operator performs a special kind of non-sequential 4-Opt move or the double-bridge move (**Fig. 5**)[27)]. This kind of move has been proven to be very effective when working with local search heuristics such as 2-Opt, 3-Opt or Lin-Kernighan since it is not easy to undo this kind of move by these heuristics.

3.2 Genetic Algorithm

A multi-population GENITOR-type GA is used in our algorithm. The GENITOR was originally proposed by Whitley and is available on the Internet[23)]. It has some distinguished characteristics, which are (i) using the steady-

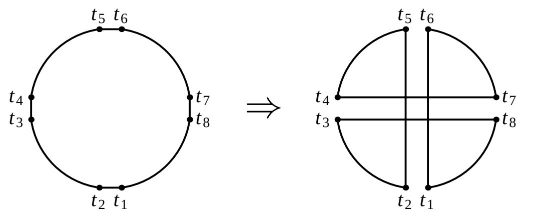

Fig. 5 A double-bridge move.

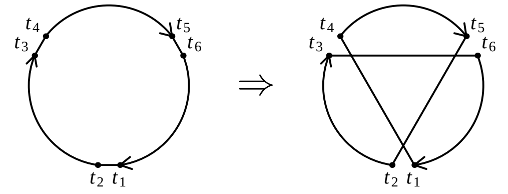

Fig. 6 A 3-Opt move (ATSP).

state update strategy, (ii) using linear ranking for selecting parents, and (iii) replacing the worst individual in the population. The reason for choosing the GENITOR-type GA is because Whitley has pointed out that the GENITOR algorithm is very effective. It needs less number of evaluations than canonical GAs (e.g., SGA, Genesis) in order to gain the same quality of solutions. We choose the multi-population model because it is very easy to implement in parallel, which we want to do in the future to speed up the method.

Our algorithm, however, has one point different from the GENITOR algorithm. It uses a replacement scheme of replacing the worst parent. This replacement scheme causes slower convergence but it can maintain a more diversity population. In every generation and for each subpopulation, only one offspring is created either from two parents by crossover or from one parent by mutation. If the offspring is better than the worst parent, it is immediately inserted into the subpopulation to replace the worst parent. However, duplicate individuals are not allowed in each subpopulation. In order to avoid premature convergence, the population is divided into a number of subpopulations and after a pre-defined number of generations, some best individuals are exchanged between these subpopulations. The algorithm halts either after a fixed number of generations or after a fixed number of generations that the best individual cannot be improved.

3.3 Heuristics

(a) Heuristics for Tour Construction

The random insertion heuristic is used to generate the initial population of GA for both types of TSPs. It can be described as follows. At first, three cities are randomly picked to form a triangle tour. Then the remaining cities are randomly chosen one by one and sequentially inserted into the tour at the position that causes the minimum increase in the length of the tour. Using this heuristic, a population with fairly good quality can be obtained and at the same time a diversity of the population needed for GA can be produced.

(b) Heuristics for Tour Improvement

The Lin-Kernighan heuristic is used as tour improvement heuristic for the STSP. For the asymmetric case, we use a version of fast 3-Opt heuristic.

Lin-Kernighan is a very effective heuristic for the STSP[24),25)]. It repeatedly checks for a sequential r-opt move (r is variable) that can produce a shorter tour and performs the move if such move exists. In our implementation, we limit the search to 12-quadrant nearest neighbors (3 nearest neighbors for each quadrant) for Euclidean distance problems or 12 nearest neighbors for others. The maximum value of r is set to 25. The *don't look* bit idea[28)] is used to speed up the algorithm. Each city has a don't look bit and it is investigated as the first city for an improving move only if its don't look bit is turned off. Initially, every city has its don't look bit turned on and after the crossover or mutation operators, only cities at the two ends of foreign edges have their don't look bits turned off. (For the mutation operator, the foreign edges are the four bridge edges.) Using the don't look bit, though the quality of tour each time the Lin-Kernighan heuristic is applied may be a little worse, the heuristic runs much faster and allows our method to process more generations and ends up with a better final results. To make the heuristic more robust, our implementation also performs a limited check for non-sequential 4-Opt moves.

Since the Lin-Kernighan heuristic cannot be applied to the ATSP, a version of fast 3-Opt is used for this case[25)]. A 3-Opt move (**Fig. 6**) causes a 3-edge change at a time, a minimum change for the ATSP. It first removes three edges of a tour to form three separate subtours, and then reconnects them in the other order, such that the direction of each subtour is not reversed. A 3-Opt heuristic checks if there is any 3-Opt moves that can reduce the length of the tour. If such move is found, it performs the move and repeats the procedure. Fast 3-Opt heuristic limits the cities investigated to K nearest neighbors and uses the don't look bits to reduce the computational time. In our implementation, the value K is set to 12.

4. Experiments and Discussions

All the experiments were run on a computer running Windows 98 with a Pentium-III 700 MHz processor and 128 MB of memory. The program is written in C language and compiled using Microsoft Visual C++ 6.0.

4.1 Comparing GSX Operators

The first set of experiments was performed to measure the effectiveness of the newly improved operator. We ran the program with three versions of the GSX, namely, GSX-0, GSX-1, and GSX-2. Three benchmarks, `gr96`, `d198`, and `pcb442` were chosen from the TSBLIB. For each benchmark, we set the population size to 1,000, which is equally divided into 10 subpopulations with each subpopulation having 100 individuals. The selection bias was set to 1.0 (i.e., random selection) and the mutation rate to zero (i.e., no mutation).

To see the differences between the crossovers more clearly, the tour improvement heuristic was not used. The tour construction heuristic, however, was used to generate the initial population because we want to know how each crossover works at the stage of highly fit individuals. The migration interval was set to 1,000 generations and 5 best individuals of each subpopulation are exchanged at each migration interval. The program stopped after 50,000 generations. The results are averages of 20 runs and are given in **Fig. 7**, **Fig. 8**, and **Fig. 9**.

For all three problems, GSX-2 always converges fastest and gives the best final results. GSX-0 converges slowest and has the worst final results. For the `pcb442` benchmark, the operator could not improve the best individual of the population after 50,000 generations. This shows that the number of foreign edges of an operator has very strong effect at the stage when the quality of individuals is high. Three benchmark experiments lead us to the conclusion that the newly improved GSX-2 is greedier than conventional ones, especially when the problem size is large.

Table 1 shows the CPU times of the method for each of the GSX operators. We can notice that the CPU times of all three operators are proportional to the number of cities. GSX-2 is slightly faster than GSX-0 and slightly slower than GSX-1. GSX-2 is slower than GSX-1 because it has to spend time for checking exchangeable subtours. However, since the running time of our hybrid GA is largely occupied by local search heuristics, the extra running time of GSX-2 over GSX-1 has almost no effect to the overall running time of our hybrid algorithm.

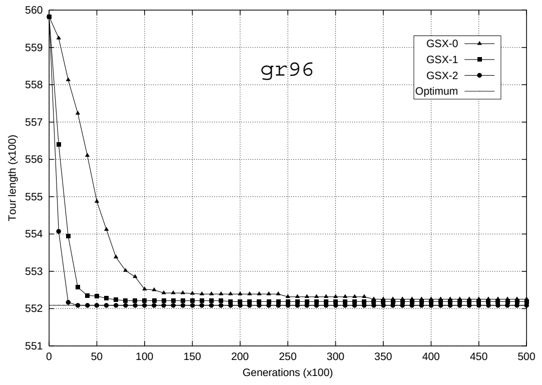

Fig. 7 Comparative convergence for gr96.

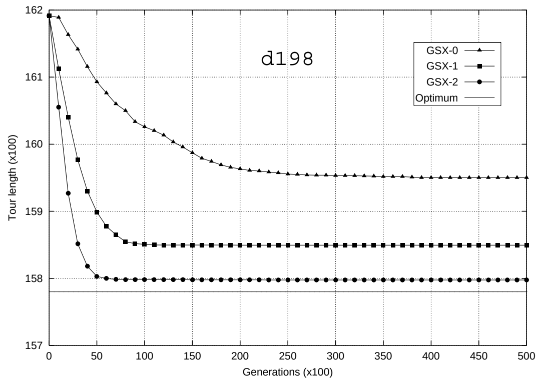

Fig. 8 Comparative convergence for d198.

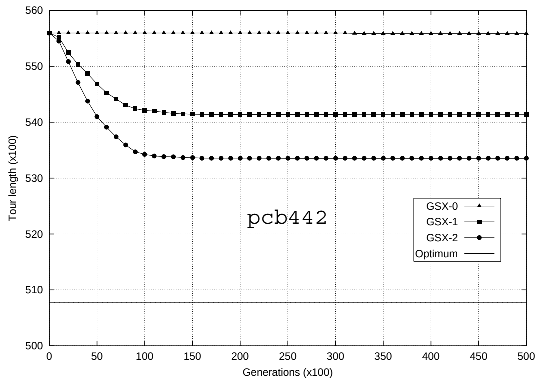

Fig. 9 Comparative convergence for pcb442.

Table 1 CPU times of GSX operators (sec).

Operator	gr96	d198	pcb442
GSX-0	7.65	22.48	54.29
GSX-1	4.36	15.76	36.76
GSX-2	6.14	19.96	42.31

Table 2 Measured *AoC* and *AoD* values of GSX operators.

	Operator	gr96	d198	pcb442
AoC	GSX-0	28.720	21.384	13.270
	GSX-1	97.627	98.608	99.293
	GSX-2	97.943	98.671	99.212
AoD	GSX-0	75.172	82.568	89.459
	GSX-1	7.081	5.787	3.694
	GSX-2	7.992	7.069	5.036

The second set of experiments was performed to address the question why GSX-2 is better than GSX-0 and GSX-1. We define two values, *AoC* and *AoD*, for each crossover. These values are calculated by the following equations.

$$AoC = 100\left(1 - \frac{\sum fe(O,A,B)}{ncross \times ncities}\right) \quad (1)$$

$$AoD = 100\frac{\sum min\big(d(O,A),d(O,B)\big)}{ncross \times ncities} \quad (2)$$

AoC is the average number of edges inherited from parents, per crossover per edge. It shows the ability of inheriting characteristics from parents. *AoD* is the average hamming distance between the offspring and the nearest parent, per crossover per edge. It shows the ability of preserving diversity for the population.

The function $fe(O, A, B)$ here returns the number of foreign edges in the offspring O created by crossover from two parents A and B. The function $d(O, A)$ returns the number of different edges of the offspring O and parent A. *ncross* is the number of times crossover is performed and *ncities* is the number of cities of the problem. (In the TSP, the number of edges in a tour equals to the number of cities.) The same three problems above were used for experiments. Experimental parameters were set as the same with the first set of experiments, except that each crossover was performed 100,000 times. The results are given in **Table 2**.

We can see from the results that GSX-0 has much smaller *AoC* and much bigger *AoD* compared to GSX-1 and GSX-2 operators. This means that GSX-0 has quite small ability of inheriting characteristics from parents while its ability of preserving diversity for the population is big. However, at the stage of highly fit individuals, the ability of inheriting characteristics has more strong effect than the ability of preserving diversity. This explains why GSX-0 is the worst operator.

On the other hand, GSX-1 and GSX-2 operators have very high *AoC* and quite small *AoD* values. They can transmit more than 99% of edges from the parents to the offspring in the `pcb442` problem. GSX-2 has slightly higher *AoC* in the two small problems and slightly smaller *AoC* for the biggest problem. However, its *AoD* values are consistently higher than those of GSX-1 operator. This shows that GSX-2's superior ability of preserving diversity is the main reason why it performed better than GSX-1, especially for the `pcb442` problem.

4.2 Validating the Hybrid GA

We performed another set of experiments for measuring the performance of the hybrid GA. The newly improved operator GSX-2 was used as the crossover of the method. This time, due to tour improvement heuristics are used, a smaller population size of 500 individuals was employed. The population was divided into 10 subpopulations, each having 50 individuals. Other parameters were set empirically. For all problems, the selection bias for selecting parents was set to 1.25 and the mutation rate to 0.1. After every 500 generations, three best individuals of each subpopulation are exchanged between subpopulations. The program stopped after 50,000 generations or after 3,000 (STSP) or 1,000 (ATSP) generations of non-improving of the best individual. Both symmetric and asymmetric TSP benchmarks taken from the TSPLIB with sizes up to 3,795 cities were used for experiments. All benchmarks were performed 20 runs. **Table 3** and **Table 4** show the results of symmetric and asymmetric benchmarks, respectively.

In these tables, *Name* is the problems file names in TSPLIB. Except for some asymmetric problems, the suffix number of a problem name indicates the number of cities of that problem. For example, the `lin318` problem has 318 cities. *Optimum* denotes the optimal tour length. *Best quality*, *Avg. quality*, and *Worst quality* show the percentages excess over the optimum of the best, the average and the worst tour length of 20 runs, respectively. These qualities are calculated using equation (3). *Avg. gen.* displays the average number of generations needed until the program converged. *Ratio of opt.* shows the number of times an optimal solution is found within 20 runs. *CPU time* indicates the run-

Table 3 Results of STSP benchmarks.

Problem	Optimum	Best quality	Avg. quality	Worst quality	Avg. gen.	Ratio of opt.	CPU time
lin318	42,029	0.000%	0.000%	0.000%	3,179	20/20	57.42
pcb442	50,778	0.000%	0.000%	0.000%	3,405	20/20	41.61
att532	27,686	0.000%	0.000%	0.000%	3,808	20/20	88.86
gr666	294,358	0.000%	0.000%	0.000%	4,052	20/20	187.39
rat783	8,806	0.000%	0.000%	0.000%	3,659	20/20	70.14
pcb1173	56,892	0.000%	0.000%	0.000%	4,135	20/20	167.95
fl1577	22,249	0.000%	0.004%	0.031%	5,556	17/20	299.45
d2103	80,450	0.000%	0.000%	0.000%	4,155	20/20	433.38
pcb3038	137,694	0.000%	0.008%	0.034%	7,449	8/20	1,200.85
fl3795	28,772	0.000%	0.001%	0.007%	6,274	18/20	1,034.38

Table 4 Results of ATSP benchmarks.

Problem	Optimum	Best quality	Avg. quality	Worst quality	Avg. gen.	Ratio of opt.	CPU time
ry48p	14,422	0.000%	0.000%	0.000%	1,156	20/20	0.27
ft53	6,905	0.000%	0.000%	0.000%	1,133	20/20	0.43
ftv64	1,839	0.000%	0.000%	0.000%	1,198	20/20	0.32
ft70	38,673	0.000%	0.000%	0.000%	1,321	20/20	0.94
kro124	36,230	0.000%	0.000%	0.000%	1,318	20/20	0.54
ftv170	2,755	0.000%	0.000%	0.000%	1,640	20/20	1.42
rgb323	1,326	0.000%	0.000%	0.000%	1,069	20/20	29.49
rgb358	1,163	0.000%	0.000%	0.000%	1,126	20/20	27.87
rgb403	2,465	0.000%	0.000%	0.000%	1,106	20/20	26.88
rgb443	2,720	0.000%	0.000%	0.000%	1,105	20/20	34.67

ning time for one run in seconds.

$$quality(\%) = 100\frac{(len - opt)}{opt} \tag{3}$$

For the symmetric problems, Table 3 shows that all optimal solutions are found within 20 runs. All of the optimal solutions except for the `fl1577`, `pcb3038`, and `fl3795` problems are found in all runs. The average qualities of solution for these three problems are also very high (0.004% for `fl1577`, 0.008% for `pcb3038`, and less than 0.001% for `fl3795`). From the viewpoint of quality of solutions, these results are the best so far obtained by applying GA to the TSP. Merz and Freisleben[19] reported an average tour length of 28,868.5 (0.335% above the optimum) for the `fl3795` problem using the Genetic Local Search, which also uses the Lin-Kernighan heuristic as the local search. Gorges-Schleuter[20] reported an average tour length of 28,850 (0.271% above the optimum) for the `fl3795` problem using a hierarchy population GA with the maximal preservative crossover and a weaker local search heuristic. Both methods could not find the optimal solution for this problem within 20 runs. Nagata and Kobayashi[21], using a GA with the complicated edge assembly crossover, found an average tour length of 0.029% above the optimum for the `pcb3038` problem and found the optimal solution once within 20 runs. Jung and Moon[22] found an average tour length of 0.052% excess over the optimum for the `pcb3038` problem by using a GA with the natural crossover and Lin-Kernighan heuristic. Their method could not find the optimum for this problem even within 100 runs. So far, all of these methods are considered to be the state-of-the-art algorithms for solving the STSP by using GA-based methods.

From the viewpoint of running time, our method is also quite fast compared to some other methods. In average, it needed about 17 minutes to solve the biggest 3,795-city problem. Merz and Freisleben's method needed about 5 hours of a DEC Alphastation 255 running Digital Unix V4 to solve the same problem. Gorges-Schleuter reported an average CPU time of more than 5 hours of a SUN UltraSparc with 170 MHz for this problem. Nagata and Kobayashi's method needed 8,707 seconds of an Intel Pentium 200 MHz processor for the `pcb3038` problem while ours CPU time for this problem was 1,200 seconds. Although different computer types were used, it seems that our method is at least as fast as these methods. The only method that is faster than ours is Jung and Moon's method. It needed 816 seconds of a Pentium III 450 MHz processor to solve the `pcb3038` problem.

Table 4 summarizes the results for ATSP benchmarks. Note that in this table, problems prefixed with `ftv` have sizes equal to their suffix numbers plus one. For example, the size of the `ftv64` problem is $64+1 = 65$. The `kro124` problem has a size of 100 cities. The results show that the proposed method is capable of finding always the optimal solutions for each of the ATSP benchmarks. Obviously, no any methods could be better than ours from the viewpoint of quality of solution. The running time of ATSP benchmarks is slightly faster than that of STSP benchmarks, given the same problem size. In average, our method needed only 35 seconds to solve the biggest 443-city problem. Nagata and Kobayashi's method spent 286 seconds (Intel Pentium 200 MHz processor) for this problem. Merz and Freisleben's method reported spending 100 seconds (DEC Alphastation 255 running Digital Unix V4) to solve the `ftv170` problem while the running time of our method for this problem was 1.42 seconds.

It is notable that the running time of our method does not depend only on the problem size. For example, the CPU time of the `gr666` problem is more than two times slower than the CPU time of the `rat783` problem. The average number of generations required to converge shows the difficulty of problem. Here, the hardest symmetric and asymmetric problems for our method were `pcb3038` and `ftv170`, respectively.

The number of calls to the mutation and crossover operators can be approximately calculated as follows:

$$Nm = avg\_gen \times sub\_pop \times mu\_rate \quad (4)$$

$$Nc = avg\_gen \times sub\_pop - Nm \quad (5)$$

Here, Nm and Nc denote the number of calls to mutation and crossover, respectively. *avg_gen* denotes the average number of generations needed to converge. *sub_pop* is the number of subpopulations (in this paper, 10) and *mu_rate* is the mutation rate applied (0.1). The number of calls to the tour improvement heuristics (Lin-Kernighan or fast 3-Opt) does not necessary equal to the sum $Nm+Nc$. This is because the crossover sometimes produces an offspring with no foreign edge. For the tested problems, the number of calls to the tour improvement heuristics is about $(Nm + Nc)/3$.

Finally, one may ask whether the genetic search approach is more efficient than the iterative search approach, if both approaches use the same local search heuristic. We performed one more experiment to address this question. Actually, the iterative search approach is a special case of our method, in which the population size is just one and the mutation rate is 1.0. The symmetric `att532` problem was used for experiment. Since the genetic search approach needed roughly 4,000 generations to converge and there are 10 subpopulations, we set both the maximum number of generations and the maximum number of non-improving generations to 40,000 for the iterative search approach. That is, 40,000 calls will be made to both the mutation operator and the local search heuristic (Lin-Kernighan, in this case). As expected, the iterative search approach did not find the optimum in all runs. There were four times the algorithm got stuck with a local minimum of 27,703, leading to an average tour length of 0.012% above the optimum. The genetic search approach found the optimum for this problem in all runs. It is also not surprising that the running time of the iterative approach was 187 seconds, more than two times slower than the genetic search approach, if we remember that the number of calls to the tour improvement heuristic for the genetic approach is only one third of that for the iterative search approach. The result confirms that the genetic search approach is more efficient than the iterative search approach.

5. Conclusions

This paper presents a GA-based method for solving both symmetric and asymmetric TSPs. The method is a multiple population GENITOR-type GA, which uses the newly improved versions of the GSX and involves local search heuristics. New improvements to the GSX are proposed so that it can work more effectively at the stage of highly fit individuals. The random insertion heuristic is used to generate the initial population and the Lin-Kernighan heuristic (STSP) or fast 3-Opt heuristic (ATSP) is applied after the crossover and mutation operators to improve the quality of individuals during the search.

The method has been validated on a number of symmetric and asymmetric TSP benchmarks, ranging in size up to 3,795 cities, taken from the TSPLIB. The benchmark tests that all optimal solutions have been found in a reasonable computing time lead us to the conclusion that the improved operators work very well in the hybridizing environment and the method is powerful for solving both types of TSPs, which have many applications in the real world.

The general framework of the method can help to develop competent GA-based methods for other combinatorial optimizations tasks. By extending this method, we have succeeded in developing a high performance GA-based method for the multiple-sequence-alignment problem in genome informatics, which is also NP-hard[29].

There are several issues for future works. First, since the running time of our algorithm is largely occupied by the local search heuristics, we want to investigate them further to improve the time-quality tradeoff of our algorithm. Second, we want to extend the method so it can be run in parallel. Third, we want to test the method with some bigger problems.

Acknowledgments The authors would like to thank the anonymous referees for their useful comments and suggestions. This research is partly supported by the 2001 and 2002 grants of the MESSC 13208001 and 14015206.

References

1) Holland, J.H.: *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor (1975).

2) Goldberg, D.E.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley (1989).

3) Baeck, T., Fogel, D.B. and Michalewicz, Z.: *Handbook of Evolutionary Computation* (1997).

4) Goldberg, D.E. and Lingle, R.: Alleles, Loci, and the Traveling Salesman Problem, *Proc. Int'l Conf. on Genetic Algorithms and Their Applications*, pp.154–159 (1985).

5) Oliver, I., Smith, D. and Holland, J.: A Study of Permutation Crossover Operators on the Traveling Salesman Problem, *Proc. 2nd Int'l Conf. on Genetic Algorithms*, pp.224–230 (1987).

6) Davis, L.: Job Shop Scheduling with Genetic Algorithms, *Proc. Int'l Conf. on Genetic Algorithms and Their Applications*, pp.136–140 (1985).

7) Muhlenbein, H.: Evolution in Time and Space — The Parallel Genetic Algorithm, *Foundations of Genetic Algorithms*, pp.316–337 (1991).

8) Whitley, D., Starkweather, T. and Fuquay, D.: Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator, *Proc. 3rd Int'l Conf. on Genetic Algorithms*, pp.133–140 (1989).

9) Whitley, D., Starkweather, T. and Shaner, D.: The Traveling Salesman and Sequence scheduling: Quality Solutions Using Genetic Edge Recombination, *The Handbook of Genetic Algorithms*, pp.350–372 (1991).

10) Starkweather, T., McDaniel, S., Mathias, K., Whitley, D. and Whitley, C.: A Comparison of Genetic Sequencing Operators, *Proc. 4th Int'l Conf. on Genetic Algorithms*, pp.69–76 (1991).

11) Mathias, K. and Whitley, D.: Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem, *Parallel Problem Solving from Nature*, pp.219–228 (1992).

12) Dzubera, J. and Whitley, D.: Advanced Correlation Analysis of Operators for the Traveling Salesman Problem, *Parallel Problem Solving from Nature — PPSN III*, pp.68–77 (1994).

13) Nguyen, H.D., Yoshihara, I. and Yasunaga, M.: Modified Edge Recombination Operators of Genetic Algorithms for the Traveling Salesman Problem, *Proc. 3rd Asia-Pacific Conf. on Simulated Evolution and Learning*, S-WP4-3: CD-ROM (2000).

14) Nguyen, H.D., Yoshihara, I., Aoyama, T. and Yasunaga, M.: Directed Edge Recombination of Genetic Algorithms for Asymmetric Traveling Salesman Problems, *Proc. Int'l Conf. on Artificial Intelligence in Science and Technology*, pp.56–61 (2000).

15) Sengoku, H. and Yoshihara, I.: A Fast TSP Solver Using GA on JAVA, *Proc. 3rd Int'l Symposium on Artificial Life, and Robotics*, pp.283–288 (1998).

16) Takeda, A., Yamada, S., Sugawara, K., Yoshihara, I. and Abe, K.: Optimization of Delivery Route in a City Area using Genetic Algorithm, *Proc. 4th Int'l Symposium on Artificial Life, and Robotics*, pp.496–499 (1999).

17) Grefenstette, J.J., Gopal, R., Rosmaita, B. and Van Gucht, D.: Genetic Algorithms for the Traveling Salesman Problem, *Proc. Int'l Conf. on Genetic Algorithms and Their Applications*, pp.160–168 (1985).

18) Freisleben, B. and Merz, P.: A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems, *Proc. 1996 IEEE Int'l Conf. on Evolutionary Computation*, pp.616–621 (1996).

19) Merz, P. and Freisleben, B.: Genetic Local Search for the TSP: New Results, *Proc. 1997 IEEE Int'l Conf. on Evolutionary Computation*, pp.159–164 (1997).

20) Gorges-Schleuter, M.: On the Power of Evolutionary Optimization at the Example of ATSP and Large TSP Problems, *European Conf. on Artificial Life '97* (1997).

21) Nagata, Y. and Kobayashi, S.: Edge Assembly Crossover: A High-power Genetic Algorithm for the Traveling Salesman Problem, *Proc. 7th Int'l Conf. on Genetic Algorithms*, pp.450–457

(1997).
22) Jung, S. and Moon, B.: The Natural Crossover for the 2D Euclidean TSP, *Proc. Genetic and Evolutionary Computation Conf. 2001*, pp.1003–1010 (2001).
23) Whitley, D.: The GENITOR Algorithm and Selective Pressure: Why Rank-Based Allocation of Reproductive Trials is Best, *Proc. 3rd Int'l Conf. on Genetic Algorithms*, pp.116–121 (1989). Source code available from GENITOR group homepage, http://www.cs.colostate.edu/~genitor/.
24) Lin, S. and Kernighan, B.: An Effective Heuristic Algorithm for the Traveling Salesman Problem, *Operation Research*, Vol.21, pp.498–516 (1973).
25) Johnson, D.S. and McGeoch, L.A.: The Traveling Salesman Problem: A Case Study in Local Optimization, *Local Search in Combinatorial Optimization*, pp.215–310 (1997).
26) Reinelt, G.: TSPLIB — A Travelling Salesman Problem Library, *ORSA Journal on Computing*, Vol.3, No.4, pp.376–384 (1991). http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95
27) Martin, O., Otto, S.W. and Felten, E.W.: Large-Step Markov Chains for the Traveling Salesman Problem, *Complex Systems*, Vol.5, No.3, pp.299–326 (1991).
28) Bentley, J.L.: Experiments on Traveling Salesman Heuristics, *Proc. 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pp.91–99 (1990).
29) Nguyen, H.D., Yoshihara, I., Yamamori, K. and Yasunaga, M.: A Parallel Hybrid Genetic Algorithm for Multiple Protein Sequence Alignment, *Proc. 2002 World Congress on Computational Intelligence*, CD-ROM 2002.

(Received February 4, 2002)
(Revised April 11, 2002)
(Accepted July 1, 2002)

Hung Dinh Nguyen received the B.S. degree in electronics from Hanoi University of Technology in 1991 and the M.S. degree in computer science from Miyazaki University in 2000. From 1991 to 1997, he was a researcher of the Forest Inventory and Planning Institute, Vietnam. He is now a doctorate candidate at Miyazaki University. His current interests include evolutionary computation, combinatorial optimization, and genome informatics. He is a student member of IEEE.

Ikuo Yoshihara received the B.S. degree in physics from Tokyo Institute of Technology in 1969, the M.S. degree in physics from Tokyo University of Education in 1971, and the Dr. Eng. degree in control engineering from Tokyo Institute of Technology in 1989. He researched in fire protection system, environmental control, numerical simulation, neural networks and genetic algorithms at Systems Development Laboratory of Hitachi Ltd. from 1971 to 1999. He has been a professor of Miyazaki University since 1999 and his research interests include evolutionary computation, genome informatics, neural networks and high performance computing. He is a member of IEEE, IPSJ, IEICE, IEEJ, JSIAM, SICE, and JAFSE.

Kunihito Yamamori received the B.S. degree in electronics and information engineering from Kanazawa University in 1992. He received the M.S. and Ph.D. degrees in information science from Japan Advanced Institute of Science and Technology in 1994 and 1997, respectively. From 1997 to 2001, he was an associate at Japan Advanced Institute of Science and Technology. Currently, he is an associate professor at the Faculty of Engineering, Miyazaki University. His research focuses on parallel computing, neural network and fault-tolerant systems. He is a member of IEEE, IPSJ, and IEICE.

Moritoshi Yasunaga received the B.E., M.E. and Dr.Eng. degrees from University of Tsukuba, Japan, in 1981, 1983, and 1993, respectively. From 1983 to 1994, he was a researcher of the Central Research Laboratory, Hitachi Ltd., Japan. He joined the faculty of the Institute of Information Science and Electronics, University of Tsukuba in 1994 where he is currently an associate professor. His research interests include reconfigurable computer, evolvable systems, evolutionary computation, genome informatics, and artificial neural networks. Dr. Yasunaga is a member of IEEE, INSS, IPSJ, and IEICE.