

# Mechanism Design and Analysis of Genetic Operations in Solving Traveling Salesman Problems

Hongwei Ge<sup>1</sup>, Yanchun Liang<sup>1,2</sup>, Maurizio Marchese<sup>2</sup>, and Lu Wang<sup>1</sup>

<sup>1</sup>College of Computer Science, Jilin University, Key Laboratory of Symbol Computation and Knowledge Engineering of the Ministry of Education, Changchun 130012, China

<sup>2</sup>Department of Information and Communication Technology, University of Trento, Via Sommarive 14, 38050, Povo (TN) Italy  
ycliang@jlu.edu.cn

**Abstract.** In this paper, some novel improved genetic operations are presented, several combinations of genetic operations are examined and the functions of these operations at different evolutionary stages are analyzed by numerical experiments. The essentiality of the ordering of the gene section, the significance of the evolutionary inversion operation and the importance of the selection model are discussed. Some results provide useful information for the implementation of the genetic operations for solving the traveling salesman problem.

## 1 Introduction

The Traveling Salesperson Problem (TSP) is a well-known non-polynomial time or NP-hard combinatorial optimization problem that has many potential applications and has been studied thoroughly. Early efforts in combinatorial optimization in the field of evolutionary computation focused on the TSP, which is a generalization and simplified form of various complicated problems that have appeared in many fields. Hence, a successful solution for the TSP has not only theoretical significance in computational theory but also important practical value. The first attempt to use simulated evolutionary optimization to address the TSP began in the 1980s. Since then the study on solving the TSP by simulating natural evolutionary processes has been very active, especially, in the recent decade. Among those studies most are based on the genetic algorithm [1-3] and a few on the evolutionary programming [4, 5] and the evolution strategy [6]. In this paper we present some improved genetic operations, examine some combinations of genetic operations, and analyze the functions of these operations.

## 2 Genetic Algorithms for Solving the TSP

The TSP is conceptually simple: a traveling salesperson must visit every city of a specific set of cities exactly once and then return to the starting point. Mathematically, the TSP is one of a search of a permutation  $\pi = \{c_1, c_2, \dots, c_n\}$  of the integer subset  $X = \{1, 2, \dots, n\}$ , where an element of  $X$  represents a serial number of one of  $n$  cities, such that

$$T_d = \sum_{i=1}^{n-1} d(c_i, c_{i+1}) + d(c_1, c_n) \quad (1)$$

reaches a minimum, where  $d(c_i, c_{i+1})$  represents the distance from city  $c_i$  to city  $c_{i+1}$ . Note that the size of the search space for the symmetric TSP is  $(n-1)!/2$ , where  $d(c_i, c_{i+1}) = d(c_{i+1}, c_i)$ , a number that grows faster than any finite power of  $n$ .

### 2.1 Coding Pattern and Fitness Function

It is natural to adopt coding methods according to the order of the cities to be visited when using a GA to solve TSPs. When using the city-order coding method, the legal constraint that a city number cannot appear more than once, is implied on initialization of a population of feasible solutions. In this section the linear scaling technique for calculating the fitness function is used.

$$f = \alpha M \sqrt{n} / T_d \tag{2}$$

Where  $\alpha$  is a predetermined constant,  $n$  the number of cities,  $M$  the side-length the smallest square covering all cities and  $T_d$  the length of the tour calculated using (1).

### 2.2 Crossover Operations

The crossover is widely acknowledged as critical to the successful operation of a GA and is responsible for exploring the solution space and exploiting or retaining favourable characteristics to yield feasible solutions of higher fitness values. In the simulation experiments seven crossover operators were used in different combinations of genetic operations to compute solutions of the TSP:

- 1) Partially matched crossover (PMX)<sup>[7]</sup>.
- 2) Order Crossover (OX)<sup>[8]</sup>.
- 3) Order Crossover-like1 (OX-1)<sup>[9]</sup>.
- 4) Order Crossover-like2 (OX-2).

This last operator is an improved version of the OX operator proposed in this paper. Here the gene-segment is replaced with several random positions, as follows.

The two parent strings  $A$  and  $B$  are:

$$A = 1 \underline{2} \underline{3} \underline{4} 5 6 7 8 9 \text{ and } B = 9 \underline{8} \underline{7} \underline{6} 5 4 3 2 1.$$

It is assumed that the second and the fourth positions are randomly selected. The corresponding cities concerning these positions are those numbered 2, 4, 8, 6, whose positions are retained in the offspring, and the other cities in the corresponding positions of the two parent strings are swapped. Hence on implementing crossover operator approach one obtains the offspring

$$A' = 9 2 7 4 5 6 3 8 1 \text{ and } B' = 1 8 3 6 5 4 7 2 9.$$

- 5) Mapping approach of single-point crossover.

This is an improved approach to the PMX. Only one crossover point is used instead of two, and is described as follows: two parent strings are denoted  $A$  and  $B$ , and a crossover point is selected randomly as

$$A = 9 1 4 5 6 | 7 8 3 2 \text{ and } B = 6 8 1 2 3 | 9 5 4 7.$$

A simple single-point crossover operation, applied to the above two strings yields

$$A' = 9 1 4 5 6 | 9 5 4 7 \text{ and } B' = 6 8 1 2 3 | 7 8 3 2.$$

Examine the repeat for the number in front of the crossover point and implement the crossover according to the mapping relation of the position in the rearward of the

crossover point. For string  $A'$ , we have that  $9 \rightarrow 7$ ,  $5 \rightarrow 8$ ,  $4 \rightarrow 3$  and  $7 \rightarrow 2$ . From the first and last replacements we have the final result  $9 \rightarrow 2$ . Hence, one obtains

$$A'' = 2 \ 1 \ 3 \ 8 \ 6 \ 9 \ 5 \ 4 \ 7 \text{ and } B'' = 6 \ 5 \ 1 \ 9 \ 4 \ 7 \ 8 \ 3 \ 2 .$$

6) Order crossover approach of a single point<sup>[10]</sup>.

7) Position information<sup>[10]</sup>.

### 2.3 Mutation Operations

Mutation refers to the process of increasing diversity in the population by introducing random variations in the individuals of the population. Four kinds of mutation operations are employed in different genetic operation combinations, of which two improved crossover operations are proposed in this paper. The four mutation operations are as follows:

1) Reciprocal exchange mutation. Select two points in the string and swap their values.

2) Insertion mutation. Select a number in the string randomly and insert the number into the point between two randomly selected insertion points.

3) Translocation mutation. It is an improved approach proposed in this paper. A decision is made to mutate each number in the string using a Bernoulli approach. The numbers without passing Bernoulli probabilities are kept in their original positions and the others are translocated in turn. For example, for string  $A = 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9$ , if the numbers 3, 6 and 7 pass Bernoulli probabilities, then they are translocated in turn and the numbers in the other positions are unchanged: the mutated gene is:  $A' = 1 \ 2 \ 7 \ 4 \ 5 \ 3 \ 6 \ 8 \ 9$ .

4) Object-oriented mutation. It is also a novel approach proposed in this paper. A heuristic mutation operator based on a greedy algorithm is proposed, where a local minimization is performed at each step. If the distance of some two adjoining numbers is larger than that of any other adjoining numbers in the string, then the adjoining for the two numbers is irrational. The steps of the algorithm are as follows:

Step 1. Find two adjoining numbers with the largest distance between them and randomly select one: set indicator  $i = 0$ .

Step 2. Exchange the selected number with another different number selected randomly from the string. If the fitness of the given string does not increase, the operation is implemented again:  $i++$ .

Step 3. If the fitness increases then stop, else if  $i < 2$ , go to Step 4, else go to Step 5.

Step 4. Select the other number from the two adjoining numbers to repeat Step 2.

Step 5. Implement reciprocal exchange mutation.

### 2.4 Evolutionary Inversion Operator

The evolutionary inversion operator involves selecting two points randomly from the string, and reversing the sub-string between these points. Numerical simulations show that whether or not to implement an evolutionary inversion operator between two numbers in a given string should be determined from a pre-computed judgment. Let the string  $A$  be

$$A = c_1, c_2, \dots, c_i, c_{i+1}, \dots, c_j, c_{j+1}, \dots, c_n \tag{3}$$

$$\text{If } d(c_i, c_j) + d(c_{i+1}, c_{j+1}) < d(c_i, c_{i+1}) + d(c_j, c_{j+1}) \tag{4}$$

then the inversion section of  $A$  is the set of values  $\{c_{i+1}, \dots, c_j\}$ . The evolutionary inversion operator is then implemented to yield

$$A = (c_1, c_2, \dots, c_i \mid c_j, c_{j-1}, \dots, c_{i+1} \mid c_{j+1}, \dots, c_n) \tag{5}$$

For the implementation of the above mentioned evolutionary inversion judgment, we should first determine the length of inversion judgment according to the requirement and the different genetic operation combinations, then select a sub-string with length  $l$  randomly in the parental string and implement the judgment for each number in the sub-string with all others using inequality Eq.(4). The evolutionary inversion adopted here is a continuous inversion operation with increasing fitness, and is implemented continuously until there is no such an operation.

### 3 Experimental Analysis of Genetic Operations

In order to examine and analyze the effect of different operations in computing the TSP, experiments were implemented using certain genetic operations and their combinations. To describe briefly, the names of some operations are abbreviated as follows: mapping approach of single-point crossover (MSPX), order crossover approach of a single point (OSPX), position information (PIX), reciprocal exchange mutation (REM), insertion mutation (IM), translocation mutation (TM), and object-oriented mutation (OOM). Besides, PMX-1 represents the PMX operation without the same numbers required in gene-sequence, and PMX-2 represents the PMX operation no restrictions on numbers in the gene-sequence.  $A + B$  represents that operators  $A$  and  $B$  are used randomly in the evolutionary process, and  $A \parallel B$  represents that operator  $A$  is used in the early iterations and operator  $B$  is used in the later iterations. The algorithms using combinations of genetic operations are shown in Table 1. In all the algorithms, the fitness proportional model and the condition of worst death are adopted.

**Table 1.** Algorithms using different combinations of genetic operations

| GA0  | GA1  | GA2    | GA3  | GA4  | GA5 | GA6  | GA7   | GA8   | GA9     | GA10 | GA11 | GA12 |
|------|------|--------|------|------|-----|------|-------|-------|---------|------|------|------|
| OX-1 | OX-1 | OX-1   | OX-1 | OX-1 | OX  | OX-2 | PMX-1 | PMX-2 | PMX  OX | MSPX | OSPX | PIX  |
| REM  | IM   | REM+IM | TM   | OOM  | REM | REM  | IM    | IM    | IM      | REM  | REM  | REM  |

#### 3.1 Analysis of Crossover and Mutation on Optimization Results

The above algorithms are used in computational experiments and the results are in Table 2. The related parameters employed in the algorithms are as follows: the population size is 100 individuals, the crossover and mutation probabilities are 0.95 and 0.003 respectively, the inversion length  $l$  is 1/3 the length of the original string, the

permitted renewable times of population is taken as 3000 (units) and the length of a tour is the relative length defined as

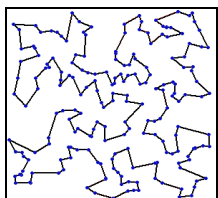
$$T = 10^4 / f \quad (6)$$

Where  $f$  is the fitness function given by (2), the constant  $\alpha$  in (2) is 76.5. For more comprehensive study, we have constructed new instances using randomly-generated two-dimensional point sets. Numerical experiments are performed on a PC with Pentium IV 1.4 GHz processor and 256MB memory.

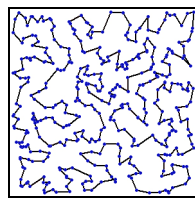
The data listed in Table 2 are the average values obtained from ten measurements. Each algorithm uses different search techniques for different numbers of cities: 200, 400, 600 and 800. In general, GA4 and GA11 have relatively superior qualities of solution and execution times to the others. From the summation of the relative lengths and elapsed times it can be seen that GA11 and GA12 are the best and worst algorithms respectively. Figs. 1-4 show the best tours for the different sizes and algorithms.

**Table 2.** Comparisons of simulation times and relative lengths using different genetic operator combinations

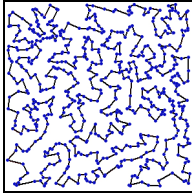
| Scale | 200    |       | 400    |       | 600    |        | 800    |        | Summation |        |
|-------|--------|-------|--------|-------|--------|--------|--------|--------|-----------|--------|
| Alg   | $T$    | Time  | $T$    | Time  | $T$    | Time   | $T$    | Time   | $T$       | Time   |
| GA0   | 99.80  | 21.12 | 99.82  | 25.57 | 104.41 | 37.58  | 111.03 | 59.09  | 415.06    | 143.36 |
| GA1   | 99.80  | 23.37 | 99.68  | 25.14 | 104.41 | 37.49  | 110.87 | 59.23  | 414.76    | 145.23 |
| GA2   | 99.80  | 22.14 | 99.64  | 25.56 | 104.37 | 37.44  | 110.65 | 59.01  | 414.46    | 144.15 |
| GA3   | 99.92  | 26.59 | 101.22 | 25.54 | 105.89 | 37.55  | 110.79 | 59.23  | 417.82    | 148.91 |
| GA4   | 98.90  | 23.42 | 99.68  | 26.47 | 103.07 | 38.21  | 110.69 | 59.52  | 412.34    | 147.62 |
| GA5   | 99.80  | 21.01 | 99.92  | 25.52 | 104.35 | 37.59  | 111.06 | 58.57  | 415.13    | 142.69 |
| GA6   | 100.33 | 23.44 | 100.47 | 26.03 | 105.78 | 37.23  | 111.56 | 58.49  | 418.14    | 145.19 |
| GA7   | 101.78 | 19.57 | 102.38 | 30.10 | 104.42 | 38.42  | 113.53 | 73.37  | 422.11    | 161.46 |
| GA8   | 99.76  | 26.51 | 102.81 | 25.07 | 105.54 | 30.29  | 113.47 | 69.36  | 421.58    | 151.23 |
| GA9   | 99.57  | 26.39 | 99.97  | 29.35 | 103.09 | 36.27  | 111.01 | 71.38  | 413.64    | 163.39 |
| GA10  | 100.63 | 49.50 | 106.19 | 25.26 | 105.27 | 31.35  | 112.65 | 73.59  | 424.74    | 179.70 |
| GA11  | 100.01 | 22.47 | 99.57  | 25.16 | 102.36 | 28.10  | 110.30 | 50.50  | 412.24    | 126.23 |
| GA12  | 106.12 | 87.57 | 107.01 | 52.38 | 109.62 | 161.37 | 123.33 | 201.14 | 446.08    | 502.46 |



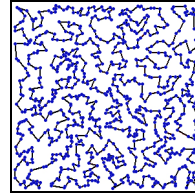
**Fig. 1.** The best tour for 200 cities



**Fig. 2.** The best tour for 400 cities



**Fig. 3.** The best tour for 600 cities



**Fig. 4.** The best tour for 800 cities

From the results in Table 2, the following conclusions may be made:

1) It can be seen that there are few differences between the best results obtained using GA0, GA1 and GA2. This suggests that the mutation operator plays a relative minor role in the search process. In fact, the essentially identical results are due to the small mutation probability, which makes the mutation occur only a few times during the search process. However, it is noticeable that the results of GA2 are better than those of GA0 and GA1, and indicates that using reciprocal exchange mutation and insertion mutation randomly promotes a diversity of the search over the space of possible solutions.

2) The best solutions of GA3 are worse than those of GA0, GA1 and GA2. This is due to the limited use of mutation in the search process. For each position in the string we judge whether it should be mutated by the probability of 0.003 on this condition that the string has passed Bernoulli probabilities. Hence the chance of mutation is greatly reduced, and diversification is minimal resulting in only a small part of the search space being explored.

3) The fitness values of GA4 are better than those of the first four algorithms because of the effect of object-oriented mutation operator presented in this section, but the time of GA4 is greater than those of GA0, GA1 and GA2.

4) The results of GA5 and those of GA0 are almost the same. The order crossover in GA5 retains the position of the gene-sequence while the crossover in GA0 relocates the sequence to the front of the string. Here there is a difference in position of the parent tail forwarded to the offspring. But since there is only one place in the gene sequence, or sequence of cities where the ordering of the cities is rearranged, it does not alter the tour length. Hence the relative lengths are similar.

5) The crossover of GA6 uses several random cities instead of a random point selected to specify the location of the tail segment, as used in GA0 and GA5. However this random selection of cities does not preserve tour order and hence, tours of greater length may result.

6) Both GA0 and GA7 are based on two-point crossover; however, in general, the best solutions in GA0 are superior to those in GA7. If one neglects the differences caused by the mutation, one finds that the algorithms differ in the method of constraint satisfaction. In order to satisfy the constraints, the crossover in GA0 adopts a way of removing the repeated cities outside the transferred gene-sequence, whereas the crossover in GA7 uses a mapping approach to change the city position outside the region of the transferred gene-sequence. For a superior TSP tour, if one retains the city order and remove some cities then the tour may still be of smaller length. However, if cities are removed and replaced with others, tours of greater length may result. Hence, the

crossover method in GA0 is more suitable for solving the TSP than that of GA7, and the fitness values reflecting the tour lengths from GA0 support this conclusion.

7) The crossover methods selected in GA7, GA8, and GA10 use the method of mapping change to satisfy the constraints. These kinds of methods have an adverse effect on maintaining gene structure when crossing over to produce offspring and the results from these algorithms are not better. The difference between GA7 and GA8 is solely that GA7 specifies that there should not be the same city numbers in the tail gene-sequences of two parental strings. If the gene-sequences are too long, then repeated city numbers tends to occur; here the tail sequences in GA7 are typically shorter than that in GA8. A good crossover operator should have the following properties: In order to promote the generation of high fitness individuals at early stages of evolution, the diversity of searching solution space should be greater and the length of the gene section to be ordered should be smaller. In order to retain the structural properties of high fitness individuals at later stages of evolution, the search diversity should be relatively constrained and the length of the gene section to be ordered should be relatively greater. GA7 and GA8 have their respective advantages at the different stage of evolution.

8) GA9 uses the PMX approach in initial evolution and the OX approach at a later stage. The results of GA9 are superior to those of GA 8 and GA5. The ordering ability of PMX is weaker than that of OX but the destructiveness is stronger. GA 9 uses PMX to simulate the process of initial evolution in the early iterations by extending the distribution of offspring in the search space. When fitter genes are obtained, GA 8 using OX has greater potential of retaining their structure and passing this on to future generations. The results of GA 9 support the observations concerning favourable properties of the crossover operator in 7) again.

9) The unique difference between GA0 and GA11 is that two-point crossover and one-point crossover are used respectively. GA11 employs a special two-point crossover operation, where the second point of crossover is not selected randomly but is fixed in the final position. Therefore, the methodology is identical in GA0 and GA11. Their difference is in the average length of the tail gene-sequence. The expectation of the length of the tail sequence in GA0 is one-third of the length of the whole string, while that in GA11 is one half. Relatively speaking, GA11 has greater structural preserving qualities. On one hand the superior gene section is apt to survive, on the other hand the inferior gene section is difficult to evolve.

10) The search ability and efficiency of GA12 are worse than those of the other algorithms. This is because the tail gene-sequence is not retained from one generation to the next in GA12 and much inherited information is lost. It can be seen that in general a suitable selection of the length of the tail gene-sequence, i.e. the ordering of a gene section, is essential.

### 3.2 Effect of Selection Model on Optimization Results

In order to examine the effect of the selection operator using a GA for optimization of the TSP, the fitness proportional model was exchanged for the tournament selection model. The new algorithm is denoted GA0\*. The measured process and parameter selection are the same as those used in GA0. The data listed in Table 3 are the average values obtained from ten measurements. The execution time is not listed in these tables as it is similar for both GA0 and GA0\*, as both algorithms perform an equal number of iterations.

**Table 3.** Comparisons of effect of selection model on optimization results (600 cities)

| Iterations (n) | 400      | 800      | 1600     | 2500     |
|----------------|----------|----------|----------|----------|
| Algorithm      | <i>T</i> | <i>T</i> | <i>T</i> | <i>T</i> |
| GA0            | 123.54   | 112.73   | 107.96   | 104.41   |
| GA0*           | 120.82   | 110.06   | 108.57   | 106.03   |

From Table 3 it can be seen that the convergence speed of GA0\* is faster than GA0's, while the final fitness value of GA0\* is worse than that of GA0.

For the case that the initial population is randomly generated, the probability that the best individual exists in the initial population is

$$p_o^0 = 1 - (1 - m/w)^N \tag{7}$$

Where *w* is the size of the solution space, *m* the number of the best individual in the solution space, and *N* a given population scale. The number of new individuals generated at the *t*<sup>th</sup> iteration, when we consider the actions of the crossover and mutation operators, is

$$n = N(p_c + p_m)p_r p_t \tag{8}$$

Where *p<sub>c</sub>* is the crossover probability, *p<sub>m</sub>* the mutation probability, *p<sub>r</sub>* the probability that the individuals generated at the *t*<sup>th</sup> iteration are not duplications of each other, *p<sub>t</sub>* the probability that the individuals generated at the *t*<sup>th</sup> iteration are not duplicated with individuals generated in all previous generations. The probability that the best individual can be generated at the *t*<sup>th</sup> iteration is

$$p_o^t = [1 - (1 - m/w)^{N(p_c + p_m)p_r p_t}] \cdot \lambda_t \tag{9}$$

Where  $\lambda_t$  is the ratio that the best individual is easier to be generated at the *t*<sup>th</sup> iteration than in the (*t*−1)<sup>th</sup> iteration because of the effect of the selection models. The term  $N(p_c + p_m)p_r p_t$  is not only related to the given parameters, but also to the intricate mechanism of crossover and mutation operators. This term is the same for both GA0\* and GA0. That is, the selection model determines whether the best individual can be generated as early as possible. The tournament model has larger value of  $\lambda_t$  than the proportional model, but it does not ensure the convergence property of the algorithm. However, the selection model of GA0 does ensure convergence. So although GA0\* has a relatively fast speed of convergence, it can easily get into local optimization solutions and accordingly it can not find a very good solution.

### 3.3 Effect of Evolutionary Inversion Operation on Best Results

The evolutionary inversion operation is used to improve a given string such that it reaches its local extreme value. To examine the local search function of the GA using the evolutionary inversion operation, the length of the gene sequence to be inverted in GA0 is increased from 1/3 of the string length to the entire string length; the new algorithm is named GA0'. In this way any two city numbers in a string are eligible to

be inverted. For GA0' the measured process and parameter selection are the same as those adopted in GA0. The relative length  $T$  corresponding to the best tour length for GA0', for  $n=800$ , is 108.42; this is better than 111.03 using GA0. However, the execution time of GA0' is 127.72, which is about twice times that obtained using GA0 of 59.09 (s). It can be seen that the evolutionary inversion operation is more optimal but more time consuming. Its local search ability enhances diversification, but for the same number of population renewal steps, if the inversion length is extended, greater optimality may be acquired. To increase the speed of computation the number of population renewal steps is reduced from 3000 using GA0, to 1400 using GA0'. The computation was rerun and the results obtained are shown in Table 4.

**Table 4.** Comparison of effect of evolutionary inversion operation on best results

| Scale     | 200    |         | 400    |         | 600    |         | 800    |         |
|-----------|--------|---------|--------|---------|--------|---------|--------|---------|
| Algorithm | $T$    | Time(s) | $T$    | Time(s) | $T$    | Time(s) | $T$    | Time(s) |
| GA0       | 99.80  | 21.12   | 99.82  | 25.57   | 104.41 | 37.58   | 111.03 | 59.09   |
| GA0'      | 99.76  | 19.53   | 99.68  | 23.19   | 104.35 | 33.09   | 110.79 | 56.30   |
| GA12      | 106.12 | 87.57   | 107.01 | 52.38   | 109.62 | 161.37  | 123.33 | 201.14  |
| GA12'     | 104.85 | 61.42   | 106.64 | 43.52   | 107.38 | 98.77   | 118.06 | 135.18  |

From the results of Table 4, it can be seen that in general the tour lengths and the execution time of GA0' are shorter than those of GA0. The same experiments were also performed for the most worst of the 13 algorithms, that is, GA12. The corresponding modified algorithm denoted GA12' was compared to GA12 and the results obtained are listed in Table 4. It can be seen from the tour lengths and the execution times, that the modified algorithm GA12' is better than the original GA12. It shows that the evolutionary inversion is effective on speeding up the convergence of the search process. The effect of the evolutionary inversion on the entire running time can be reduced to a great extent by decreasing the renewal steps of population. Since the evolutionary inversion operation needs long running time, a suitable string length for the inversion operation may be selected depending on the total number of cities so as to refine the search quality and algorithmic efficiency.

In the experiments it is found that the fluctuation among the best solutions of tours measured each time is large, whereas the fluctuation among the running time is small. These phenomena show that the evolutionary inversion operation plays a part in search process, but some feasible solutions may stick at local optima. In order to change this trend we increase the mutation probability from 0.003 to 0.3, let the renewal steps of population be still 1400, and repeat the above experiment for  $n = 600$ . We denote the algorithm as GA12". The best relative tours and variance obtained using GA12' and GA12" are shown in Table 5.

**Table 5.** Comparison of effect of evolutionary inversion operation on best results

|       | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | 10     | $\sigma^2$ |
|-------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|------------|
| GA12' | 108.08 | 107.17 | 107.15 | 108.29 | 108.20 | 107.09 | 107.81 | 107.69 | 108.31 | 106.71 | 0.38       |
| GA12" | 107.63 | 107.45 | 107.37 | 107.82 | 106.90 | 107.15 | 107.55 | 107.43 | 107.19 | 107.66 | 0.08       |

In Table 5 the mean and variance

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \quad (10)$$

are calculated to compare the methods and are used to arrive at conclusions about their relative performance: here  $x_i$  is the observed measurement and  $n$  the size of the data set.

In the 10 respective measurements using GA12' and GA12", the variance of GA12" is about 21% that of GA12'. The obvious improvement of GA12" over GA12' shows the effect of increasing the mutation probability.

## 4 Conclusions

Some simulation experiments are performed using the improved genetic operations and some combinations of genetic operations in order to analyze the functions of these operations. Genetic operations have a direct effect to optimization result. The same operation is not always favorable to individual evolution at different stages. It has a favorable effect on the genetic evolution to use different operators in different evolutionary stages or to use different operators randomly in the same evolutionary stage. Different matching range, i.e. different average length of gene section to be ordered, has also a direct impact on the best results of the TSP. A suitable selection on the length of the matching range is essential in genetic algorithms for solving the TSP. The relationship between selection of matching range and different genetic operators is a further question. The evolutionary inversion operation is significant in genetic algorithms for solving the TSP. A suitable selection of the length of the inversion judgment enables the search quality and efficiency to be improved simultaneously.

## Acknowledgment

The authors are grateful to the support of the National Natural Science Foundation of China (60433020), the science-technology development project for international collaboration of Jilin Province of China (20050705-2), the doctoral funds of the National Education Ministry of China (20030183060), and "985" Project of Jilin University. The second and the third authors would like to thank the support of the EuMI project of European Commission.

## References

1. Choi, I.C., Kim, S.I., Kim, H.S.: A Genetic Algorithm with a Mixed Region Search for the Asymmetric Traveling Salesman Problem. *Computers & Operations Research*, 30 (5) (2003) 773-786
2. Yoon, H.S., Moon, B.R.: An Empirical Study on the Synergy of Multiple Crossover Operators. *IEEE Transactions on Evolutionary Computation*, 6 (2) (2002) 212-223
3. Tang, L.X., Liu, J.Y., Rong, A.Y., Yang, Z.H.: Modelling and a Genetic Algorithm Solution for the Slab Stack Shuffling Problem When Implementing Steel Rolling Schedules. *International Journal of Production Research*, 40 (7) (2002) 1583-1595

4. Fogel, D.B.: Applying Evolutionary Programming to Selected Traveling Salesman Problems. *Cybernetics and Systems*, 24 (1) (1993)27-36
5. Chellapilla, K., Fogel, D.B.: Exploring Self-adaptive Methods to Improve the Efficiency of Generating Approximate Solutions to Traveling Salesman Problems using Evolutionary Programming. *Lecture Notes in Computer Science*, 1213 (1997) 361
6. Nurnberg, H.T., Beyer, H.G.: Dynamics of Evolution Strategies in the Optimization of Traveling Salesman Problems. *Lecture Notes in Computer Science*, 1213 (1997) 349
7. Goldberg, D.E., Lingle, R.: Alleles, Loci and Traveling Salesman Problem. In: *Proc. of the Int. Conf. on Genetic Algorithms and Their Applications*, Carnegie-Mellon University, (1985) 154-159
8. Davis, L.: Job Shop Scheduling with Genetic Algorithms. In: *Proc. of the Int. Conf. on Genetic Algorithms and Their Applications*, Carnegie-Mellon University, (1985) 136-140
9. Chen, G. L., Wang, X.F., Wang, D.S.: *Genetic Algorithms and Applications*. Press of Post and Telecommunications, Beijing, (1996) 137-147
10. Liang, Y.C., Ge, H.W., Zhou, C.G., Lee, H.P., Lin, W.Z., Lim, S.P., Lee, K.H.: Solving Traveling Salesman Problems by Genetic Algorithms. *Progress in Natural Science*, 13 (2) (2003) 135-141