A Sequence Based Genetic Algorithm with Local Search for the Travelling Salesman Problem

Shakeel Arshad, Shengxiang Yang, and Changhe Li

Department of Computer Science
University of Leicester
Leicester LE1 7RH, UK
{ *saa29, s.yang, ch160* }*@mcs.le.ac.uk*

Abstract

The standard Genetic Algorithm often suffers from slow convergence for solving combinatorial optimization problems. In this study, we present a sequence based genetic algorithm (SBGA) for the symmetric travelling salesman problem (TSP). In our proposed method, a set of sequences are extracted from the best individuals, which are used to guide the search of SBGA. Additionally, some procedures are applied to maintain the diversity by breaking the selected sequences into sub tours if the best individual of the population does not improve. SBGA is compared with the inver-over operator, a state-of-the-art algorithm for the TSP, on a set of benchmark TSPs. Experimental results show that the convergence speed of SBGA is very promising and much faster than that of the inver-over algorithm and that SBGA achieves a similar solution quality on all test TSPs.

1 Introduction

The travelling salesman problem (TSP) is a classic combinatorial optimization problem, which has been extensively studied by numerous researchers. For a TSP, a salesman needs to visit each of a set of cities exactly once, completing a tour by arriving at a city that is also the start and travelling the minimum distance. More formally, given N cities, the TSP requires to search for a permutation, using a cost matrix $C = [c_{ij}]$, where c_{ij} denotes the cost (assumed to be known by the salesmen) of travelling from city i to city j, which minimizes the path length defined as:

$$f(\pi, C) = \sum_{i=0}^{N} c_{\pi(i),\pi(i+1) mod N}$$

where $\pi(i)$ denotes the city at the i-th location in the tour.

TSPs can be classified into different classes according to the properties of the cost matrix. A TSP is called symmetric if we have $c_{ij} = c_{ji}, \forall i, j$; otherwise, it is referred to as asymmetric. If the cities of a TSP lie in a metric space, i.e., satisfying the triangle inequality, the TSP is referred to as a metric TSP. Assuming that a city i in a tour is marked by its position (x_i, y_i) in the plane, and the cost matrix C contains the Euclidean distance between the i-th and j-th city, defined as follows:

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Then, the TSP is both symmetric and metric.

The search space of a TSP is giant, containing *N!* permutations, and the TSP was identified by Garey *et. al.* [2] to be NP-hard. There are many exact and approximation algorithms developed for solving TSPs. Since the TSP has a variety of application areas, such as, vehicle routing, robot control, crystallography, computer wiring, and scheduling, etc and is a typical combinatorial optimization problem, it has attracted the interest of the genetic algorithm (GA) community [1].

In this paper, we propose a sequence based genetic algorithm (SBGA) for solving the TSP. SBGA uses a reverse approach of fragment assembly in DNA sequencing. In DNA sequencing, all possible base pairs of genome are putting together in pieces that match and the sequence becomes bigger and bigger [11, 12]. In the proposed reverse approach, first a set of best individuals are selected from the population. The individuals are broken into equal size sub tours that have the same number of cities. The sub tour with the shortest length is selected, further optimized by a 2-opt improver [10], and then stored in a sequence set. This set of sequences are further used to guide the crossover, mutation and local search operators. A random sequence is selected from the set of sequences for crossover, mutation and local search in every iteration. Similar work has been done by Ray in

[9], where the individual is broken into parts and then reconnected in a random way.

The proposed SBGA is compared with the inver-over algorithm [3], a state-of-the-art algorithm for TSPs, on a set of benchmark TSPs. Experimental results show that is superior to the inver-over algorithm in terms of the convergence speed and achieves a similar solution quality as the inver-over algorithm.

The rest of the paper is outlined are as follows. The next section describes the detail framework of (SBGA). Section 3 presents the complete experimental setup and study. Finally, section 4 concludes this paper with discussions on future work.

2 Sequence Based Genetic Algorithm (SBGA) for the TSP

In this section, the proposed SBGA for solving TSP is described in details with several new operators, like the Sequence Based Local Search (SBLS), Sequence Based Order Crossover (SBOX), and Sequence Based Inversion Mutation (SBIM). The structure of the proposed SBGA is shown in Algorithm 1.

Algorithm 1 Sequence Based GA (SBGA)

1: Initialize Pop of the size $popsize$
2: **for** each individual $ind_i \in Pop$ **do**
3: $ind_i := 2_{Opt}(ind_i, K)$
4: **end for**
5: **repeat**
6: $GenerateSequence(Num_{seq})$
7: $mating\_pool := TournamentSelect(Pop)$
8: **//Crossover**
9: **for** $j = 0$ to $popsize$ **do**
10: Select two parents i_a, i_b from the $mating\_pool$.
11: **if** $(rand() < p_c)$ **then**
12: Create $child_a$ and $child_b$ by $SBOX(i_a, i_b)$
13: Apply $SBLS(child_a, child_b)$
14: Add $child_a$ and $child_b$ to Pop_{tmp}
15: **end if**
16: **end for**
17: **//Mutation**
18: **for** each individual $ind_i \in Pop_{tmp}$ **do**
19: **if** $(rand() < p_m)$ **then**
20: $SBIM(ind_i, inversions)$
21: **end if**
22: **end for**
23: $Pop := SelectNewPop(Pop + Pop_{tmp})$
24: **until** $Termination\ condition = true$

The first step of SBGA is to initialize the population. A simple 2-Opt improver is applied to each individual $indi_i$ for K iterations to give a nice start to SBGA. The next step is the generation of number of sequences Num_{seq} from a set of best individuals that are selected from the population. Then, a mating pool is generated using tournament selection with tournament size of 7. The details of each operation are given in the following subsections, respectively.

2.1 Sequence Generation

After initializing the population, we apply the 2-opt improver [10] for K iterations to give a good start to SBGA. Then, next step is to generate sequences. For the sequence generation, a certain percentage of the best individuals from the population are selected and stored into a sequence set. From these best individuals, the first individual is selected. The individual is broken into equal parts with the same number of nodes. The next step is to find the shortest sub tour among the candidate sub tours. SBGA starts from the initial node and goes into the end like $(i = 0)$ and goes to $(i < n - nodes_{seq})$. Here $nodes_{seq}$ represents the total number of nodes of the sequence. For the individual, ABCDE-

Algorithm 2 $GenerateSequence(Num_{seq})$

1: Select best individuals from Pop and store it in $Best_{indi}[\,]$.
2: **for** $i = 0$ **to** Num_{seq} **do**
3: Split $Best_{indi}[i]$ into n sub parts with the same number of nodes
4: Calculate the length of each sub path
5: Further optimize the sup path by $2 - opt$ improver which has minimum length
6: Store the sub path into the set of Seq_{Numseq}
7: **end for**

FGHIJKLMNOP in this case total number of nodes of the sequence i-e. $nodes_{seq}$ is 4. The candidate sequences for the shortest path in this individual are $ABCD$, $BCDE$**...and** $MNOP$. In this breaking procedure, one node comes in and one goes out. So almost every node participates. Let suppose the shortest sequence is $BCDE$, which is identified i-e, length of the of the nodes is calculated and then further optimized to e.g: $CDBE$. Then this sequence is preserved in a set of sequnces Seq_{Numseq}. The same procedure for the rest top individuals, if the percentage % is 100 then the total number of sequences would be equal to population size.

2.2 Sequence Based Order Crossover (SBOX)

The Order Crossover (OX) [4, 5] is a sexual reproduction operator. It is the variant of "two point crossover". It is a classical "blind" heuristic, which does not depend on the local city to city distance information, but only on the global "whole genome" fitness to achieve progress. It is observed to be one of the best in terms of quality and speed, and simple to implement.

Our modified OX operator, SBOX, works as follows. First, a random sequence Seq_{sel} is selected from the set of sequences. Two individuals are selected from the *mating_pool*, which is created through the tournament selection as mentioned above. If the crossover condition is satisfied, then nodes of the sequence are removed from the nodes of both the individuals. Now the available number of nodes for crossover is $(nodes_{ind} - nodes_{seq})$. The pseudocode of SBOX is given as follows: The above algorithm

Algorithm 3 $SBOX(i_a, i_b)$

1: Randomly select Seq_{sel};
2: Remove the *nodes* of Seq_{sel} from the $nodes_{ind}$ of individual i_a, i_b;
3: Perform crossover on the remaining $nodes_{ind}$ of i_a and i_b do $SBOX(i_a, i_b)$;
4: Select the random location and re-insert the sequence Seq_{sel};
5: Evaluate($child_a, child_b$);

is implemented on the following example. Let i_a and i_b represent the parents ($P1$, $P2$) and $child_a$, $child_b$ represent the children ($C1$, $C2$). The crossover performs like this. Let the sequence Seq_{sel} be (CDBE) and individuals are:

$P1$ = ABCDEFGHIJKLMNOP
$P2$ = PONMLKJIHGFEDCBA
After removing (CDBE)
$P1_{temp}$ = AFG | H I J K | LMNOP
$P2_{temp}$ = PON | M L K J | IHGFA
After crossover
$C1_{temp}$ = GFAHIJKPONML
$C2_{temp}$ = NOPMLKJAFGHI
After inserting (CDBE) in a random location
$C1$ = GFAHIJK**CDBE**PONML
$C2$ = NOPMLKJA**CDBE**FGHI

2.3 Sequence Based Inversion Mutation (SBIM)

After recombination, each offspring undergoes mutation with a small probability p_m. For TSPs, the Simple Inversion Mutation (SIM) operator is one of the best performers [5]. In our approach, we perform SBIM on the selected individual to some allocated number of times, and preserve those inversions which have positive effect on the performance gain. This increases the convergence speed although involving an extra overhead on the mutation operator. The number of iterations of SBIM is dependent on whether the best fitness changes. If the best fitness changes after each generation, then SBIM will not execute. So, the number of executions will lie in the range $0 < i < nodes_{seq}$. The algorithm of SBIM is shown as follows: The

Algorithm 4 $SBIM(i_m, inversions)$

1: $i_{temp} = i_m$;
2: Randomly select Seq_{sel};
3: Reverse select Seq_{sel};
4: Remove the *nodes* of Seq_{sel} from the $nodes_{ind}$ of individual i_{temp};
5: **for** $i = 0$ **to** *inversions* **do**
6: Randomly selects two points p_1 and p_2 then do inversion on the remaining $nodes_{ind}$;
7: **for** $j = p_1$ to p_2 **do**
8: Perform inversion on the remaining $nodes_{ind}$ of i_{temp};
9: **if** $f(i'_{temp}) < f(i_{temp})$ **then**
10: $i_{temp} = i'_{temp}$;
11: **end if**
12: **end for**
13: **end for**
14: Select a random location and re-insert the sequence Seq_{sel};
15: Evaluate(i_{temp});
16: $i_m = i_{temp}$;

overall procedure of SBIM is similar to that of SBOX. The difference lies in that SBIM inverts the sequence before inserting it in the individual. Here, if the passing parameter *inversions* is equal to 0, the SBIM will not execute; otherwise, SBIM will be executed for *inversions* iterations by selecting two random points in the remaining nodes of the individual, perform the inversion. Then the fitness of the individual is calculated $f(i'_{temp})$. If it is less then previous $f(i_{temp})$ that inversion is made permanent; otherwise, the inversion is rejected. Here, the execution of SBIM is variable. Furthermore, the inverted Seq_{sel} is inserted into a random location. Our approach guarantees possibly fruitful individual as the sequence to be inserted is optimized. Here, i_m denotes the P, i_{temp} denotes the C for the be-

Table 1: Comparison results of Inver-Over, SBGA+IO, and SBGA

Instance	Results	IO	SBGA+IO	SBGA
EIL51 (426)	Best	429.53	439.184	**429.48**
	Err	0.0083	0.0309	**0.0082**
	AVG	430.66	439.84	437.80
	Err	0.0109	0.0325	0.0277
EIL76 (538)	Best	552.22	579.194	562.97
	Err	0.0264	0.0766	0.0464
	AVG	552.37	571.56	570.7
	Err	0.0267	0.0624	0.0608
EIL101 (629)	Best	654.26	675.2	**650.89**
	Err	0.0402	0.0741	**0.0348**
	AVG	656.78	685.7	666.6
	Err	0.0442	0.0902	0.0598
KROA100 (21282)	Best	21285.4	22193	**21282**
	Err	0.0002	0.0428	**0.00**
	AVG	22392.10	2239 2.10	**22382**
	Err	0.0522	0.0522	**0.0492**
KROC100 (20749)	Best	20820	21647.3	21008
	Err	0.0034	0.0433	0.0125
	AVG	20888.10	21942.10	21903.6
	Err	0.0067	0.0575	0.0560
KROD100 (21294)	Best	21517	22180	**21317**
	Err	0.0105	0.0416	**0.0011**
	AVG	21523.00	22504.30	22674.6
	Err	0.0108	0.0568	0.0648
LIN105 (14397)	Best	14432.6	14491.8	14782
	Err	0.0037	0.0078	0.0280
	AVG	14510.90	15531.20	15118.0
	Err	0.0092	0.0801	0.0514
CHN144 (30347)	Best	31253	32613.3	31782
	Err	0.0299	0.0747	0.0473
	AVG	31542.90	33268.20	32513.6
	Err	0.0394	0.0963	0.0714

low demonstration of the mutation procedure.
P = ABCDEFGHIJKLMNOP
After removing (ECDB)
P_{temp} = AFG | HIJKLMN | OP
After inversion
C_{temp} = AFG | MNLKJIH |OP
C = AFGMNLKJIHBDCEOP

2.4 Sequence Based Local Search (SBLS)

Local search algorithm is effective heuristics technique for lots of Combinatorial Optimization Problems[6]. In our approach SBGA information is gathered from various good individuals and exploited to guide the overall performance of GA towards promising area of the search space. The algorithm are as follow: In algorithm Inc_{Length} is calculated for the insertion of sequence in proper location, where the decrease in fitness is low in case of TSP. The LS which is applied in this simply takes the i_{th} individual and the sequence which is going to be inserted Seq_{sel}. In the above algorithm the distance between the first and last node of the sequence is calculated from the distance matrix relevent to the adjacent nodes of the individual where the sequence will be inserted. The LS procedure utilized in this approach is quite simple like crossover and mutation operator explained above. Similarly, the nodes of the sequence is removed from the individual and 2-opt is applied to the rest of nodes if there is gain in fitness from those swaps. Finally, the best location is searched and Seq_{sel} is inserted in that location. This approach totally depends on the generation

Algorithm 5 $SBLS(indi_i, Seq_{sel})$

1: Create sub tour X' by removing the nodes of Seq_{sel} from individual X
2: Perform 2-Opt improver on X'
3: Check the *best_position* of X' which gives the minimum length increase after inserting Seq_{sel}, according to the following equation:

$$Inc_{Length} = (Min_{j=0}^{N-M} dist[seq[0]][X'_j] + dist[seq[M-1]][X'_{j+1}] - dist[X'_j][X'_{j+1}])$$

where N is the total number of cities of the test problem and M is the number of cities of Seq_{sel}
4: Insert the sequence Seq_{sel} into X' at *best_position*
5: X=X'
6: Evaluate(X)

of new sequences when the fitness of the individuals changes, which causes the generation of a new set of sequences for coming generations of SBGA.

2.5 Maintaining Diversity

Maintaining the diversity of the population throughout the run is a major approach to avoid the premature convergence [7]. This section describes some techniques used in SBGA to prevent the premature convergence to local optima. These considerations work on avoiding the loss of genetic diversity of the whole population, and in principle, will not damage the convergence process. Initially, we set some parameters like *MaxLS*, i.e, how many times the local search will run. We keep the value in the range of [0, 20], the another parameter associated with LS is *MaxStepSize* of LS. We keep the value in the range of [5, 50]. It means that if the best fitness does not change within *MaxStepSize* LS operations, SBGA will explore more search space. If the best fitness changes, SBGA resets all the parameters to the initial state, i.e., all the parameter resets to a pre-assigned state of SBGA.

We have associated the length of the sequence with total iteration executed of *MaxLS*. The sequence length is reduced based on the following criteria. If $(MaxLS_{running} \leq (MaxLS \times 75\%))$, then the length of the sequence will be 75% of its original length. That is, if $MaxLS$ is assigned to 100 and $MaxLS_{running}$ is 75 and the Sequence length is (12 node in case CHN144.tsp) it becomes 9. Similarly, for 50% 6 and 25% 3. If the total assigned iteration becomes equal to 0 the length of the sequence becomes two, which is an edge (*i,j*) and the LS searches for the shortest edge and re-insert in a proper location. The probability of crossover p_c and mutation p_m are also changing when the best fitness does not increase.

For selecting new population from Pop and Pop_{tmp}, we use the social disasters technique (SDT) called Packing, That is, among all the individuals that have the same fitness value, only one remains unchanged and the other individuals are fully randomized [7]. We keep the identical percentage of individuals $50 \simeq 90\%$.

3 Experimental Study

In this section we present the experimental results of the proposed algorithm (SBGA). The proposed approach has been implemented in C++. All test cases (except CHN144) were chosen from TSPLIB [8]. The number of cities in these cases varies from 51 to 144. The parameters setting for the algorithm had the following values. The population sizes for the first four instances were set to 50, and 20 for the remaining. The crossover probability initially set to $p_c = 0.25$ and mutation probability $p_m = 0.0025$. The parameters which have been chosen is arbitrary , as in (SBGA) the initial set of parameters are increasing for mutation and crossover by a small value to maximum value less then 1 when the fitness does not change , if fitness changes values again resets to the initial one. For local search LS, the $MaxLS = 20$ and the stepsize initialy set to 5 and the upper maximum limit for stepsize set to $MaxStepSize = 20$ for each and every TSP instances. In Table 1, we present the results of IO, SBGA+IO and SBGA over 20 independent runs. In this table, the results of "best" rows show the best tour found and "AVG" rows display the average result *(fitness)* of 20 runs. The "Err" rows give relative deviation to the global optima *(fitness)* list in the table after the instance name.

The experimental results are compared with IO operator [3]. From Table 1, It can be seen that the SBGA achieves better solutions than IO on 5 test instances, while slightly worse on the other instances. However, from the results of table 2 and figure 1, we can see that SBGA outperforms IO in convergence speed.

We have also studied the combined approach of SBGA and IO by shifting control from SBGA to IO under a certain condition. It is also con-

firmed that SBGA can compete to the maximum level of optimality and most of the runs SBGA keep the control within itself. The plot of tour length *vs* number of generations is illustrated in Figure.1 has been shown that the (SBGA) has almost the same characteristics on different TSP benchmarks test problems. The combined effect of the (SBGA+IO) shown same level of optimization with both the operators applied together one after another, which shows that if SBGA applied at the initial level of optimization process which do fast convergence and then control given to IO can thus enhance the performance and solution quality, simply at early stage of evolution, the SBGA can drive the population to local optimum more rapidly and then IO further be applied so number of generation could be reduced as the IO has the ability to increase the population diversity, avoiding common diversity loss in common crossover operators.

Table 2: Comparison of the acceleration ratio of IO and SBGA

Instance	**Fitness**	**SBGA**	**IO**	**AR**
EIL51	501	153	512	3.35
EIL76	627	450	1110	2.50
EIL101	745	600	1801	3.00
KROA100	30225	448	1277	2.85
KROC100	30310	298	1241	4.16
KROD100	26870	595	1521	2.55
LIN105	19818	456	1386	3.03
CHN144	49953	174	1687	9.7

Table 2 shows the acceleration ratio which is the ratio of the number of generations needed to gain the specified fitness of two algorithms. The second column shows the fitness of best individual before convergence of population. The third and fourth column show the number of generation needed of SBGA and IO operator to obtain the fitness shown in the first column. The acceleration ratios is from 2.5 to 9.7 times faster than IO in the early stage of evolutionary process on all test instances.

Further improvement of solution requires greater computational efforts in terms of the number of generation. The best results SBGA got for Chn144 is 31086.4 ($Err = 0.0243648$) when the we increase the number of $gen > 5000$. In Figure 1, this improvement is sensible for the instances of TSPLIB [8] respectively, which is a good indication of the convergence behaviour of SBGA.

4 Conclusion

In this study, we presented a sequence-based genetic algorithm with local search for solving small and medium scale TSP instances. However, for larger scale instance more time is needed and the speed is comparatively slow. The behaviour of SBGA is exploitable by making use of information extracted from the population.

Some effective ideas are proposed for preserving the population diversity, preventing premature convergence and enhancing the speed of convergence at the initial stage of SBGA. Our proposed mutation operator takes SBGA to promising areas of the search space as well as contributing in the fitness increase.

The crossover operator exhibits a behaviour of displacement mutation but here extracted sub tour is optimized one. Our concept is totally dependent on the formation of set of sequences how that area should be further improved. Another on that algorithm tune the crossover and mutation probability is directly. So, other kind of approaches may be useful. Obviously, we compared our SBGA with Inver Over, one does not intend that it can compete with as running time is still a big question, but it do fast conversion at the initial stages of GA.

References

[1] E. Ozcan, and M. Erenturk. A brief review of memetic algorithms for solving Euclidean 2D travelling salesrep problem, *Proc. of the 13th Turkish Symposium on Artificial Intelligence and Neural Networks*, pp. 99-108, 2004.

[2] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems, *in Porc. of the 8th Annual ACM Symposium on Theory of Computing*, pp. 10-22. 1976.

[3] G. Tao and Z. Michalewicz. Inver-over operator for the TSP, *In Parallel Problem Solving from Nature–PPSN V*, pp. 803-812, 1998.

[4] L. Davis. Applying adaptive algorithms to epistatic domains, *Proc. of the 1985 Int. Joint Conference on Artificial Intelligence*, vol. 1, pp. 161-163, 1985.

[5] J. A. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators, *Artificial Intelligence Review*, vol. 13, pp. 129-170, 1999.

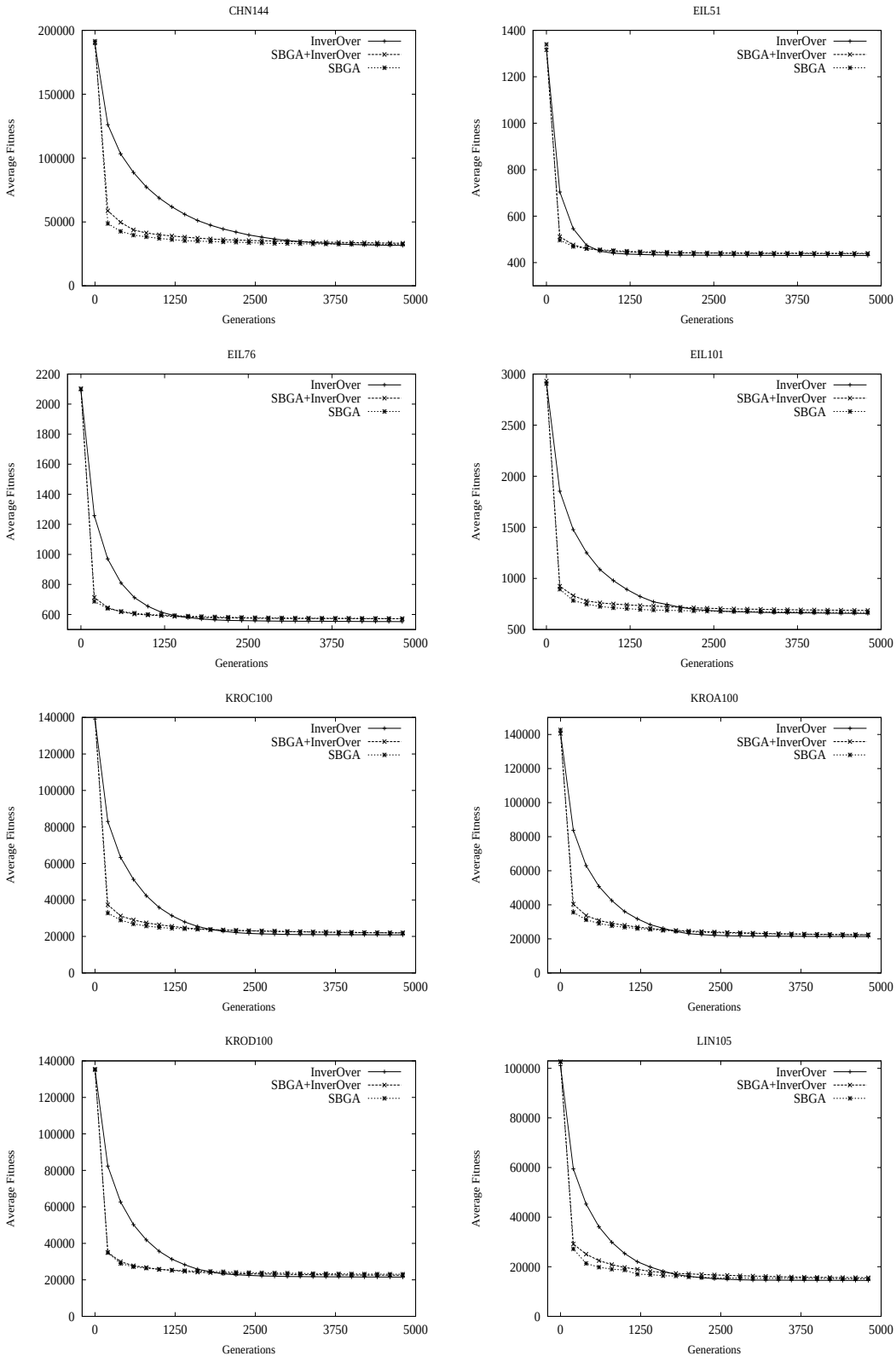

Figure 1: Experimental results of IO, SBGA+IO, and SBGA

[6] P. Merz and B.Freisleben. Memetic algorithms for the travelling salesman problem, *Complex Systems*, vol. 13, pp. 297-345, 1997.

[7] M. Rocha and J. Neves. Preventing premature convergence to local optima in genetic algorithms via random offspring generation, *In Proc. of the 12th Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Multiple Approaches to Intelligent Systems*, pp. 127-136, 1999.

[8] G. Reinelt. TSPLIB95, University Heidelberg (http://www.iwr.uniheidelberg.de/ groups/comopt/software/TSPLIB95), 1995.

[9] S. S. Ray, S. Bandyopadhyay, and S. K. Pal. New operators of genetic algorithms for the travelling salesman problem. *In Proc of the 2004 Int. Conf. on Pattern Recognition (ICPR)*, vol. 2, pp. 497-500, 2004.

[10] S. Lin and B. Kernighan. An effective heuristic algorithm for the travelling salesman problem, *Operations Research*, vol. 21, pp. 498-516, 1973.

[11] P. A. Pevzner, H. Tang, and M. S. Waterman. An Eulerian path approach to DNA fragment assembly, *Proc. of Natl Acad Sci USA*, vol. 98, pp. 9748-9753, 1998.

[12] R. J. Parsons, S. Forrest, and C. Burk. Genetic algorithms, operators, and DNA fragment assembly, *Machine Learning*. vol. 21, pp. 11-33, 1995.