A Hybrid Genetic Algorithm and Inver Over Approach for the Travelling Salesman Problem

Shakeel Arshad, and Shengxiang Yang, *Member, IEEE*

***Abstract*— This paper proposes a two-phase hybrid approach for the travelling salesman problem (TSP). The first phase is based on a sequence based genetic algorithm (SBGA) with an embedded local search scheme. Within the SBGA, a memory is introduced to store good sequences (sub-tours) extracted from previous good solutions and the stored sequences are used to guide the generation of offspring via local search during the evolution of the population. Additionally, we also apply some techniques to adapt the key parameters based on whether the best individual of the population improves or not and maintain the diversity. After SBGA finishes, the hybrid approach enters the second phase, where the inver over (IO) operator, which is a state-of-the-art algorithm for the TSP, is used to further improve the solution quality of the population. Experiments are carried out to investigate the performance of the proposed hybrid approach in comparison with several relevant algorithms on a set of benchmark TSP instances. The experimental results show that the proposed hybrid approach is efficient in finding good quality solutions for the test TSPs.**

I. Introduction

The travelling salesman problem (TSP) is probably the most widely studied combinatoral optimization problem and attracted a large number of researchers over the last five decades. 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 the start and by travelling the minimum distance or to find a minimum weight Hamiltonian cycle in a graph. More formally, given N cities, the TSP requires to search for a permutation $\pi = \{\pi(0), \pi(1), \cdots, \pi(N-1)\}$, using a cost matrix $C = \{c_{ij}\}$, where c_{ij} denotes the cost (known to the salesmen) of travelling from city i to city j, which minimizes the path length defined as follows:

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

where $\pi(i)$ denotes the i-th city in the tour. 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 cities, 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.* [3] 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 [7]. Various techniques have been considered when solving TSP by using GAs. These include crossover, mutations, selections and local searches. In most GAs, the operations are simple; but more complex techniques are also used and are still in progress.

This paper proposes a two-phase hybrid approach for the TSP. In the first phase, a sequence based genetic algorithm (SBGA) which was proposed in [1], but here we have used an embedded local search scheme to solve the TSP. Here the word ***"embedded"*** means that the local search technique is applied before the final evaluation of the individual in crossover and mutation. Within the SBGA, a memory is introduced to store good sequences (sub-tours) extracted from previous good solutions. The stored sequences are used to guide the generation of offspring during the crossover and mutation operation. After each crossover and mutation operation, local search runs to improve the fitness of newly created child based on the set of sequences stored in the memory. 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. After SBGA finishes, the hybrid approach enters the second phase, where the inver over (IO) algorithm [11], which is a state-of-the-art algorithm for the TSP, is used to further improve the solution quality of the population. IO is a kind of blind operator. Along with good adaptive power it suffers from random inversions which does not give better individuals finally. We modified the original IO by making the inversions restrictive, only those inversions with better fitness scores will be retained and replace the original tour *i-e.* a kind of local refinement with IO.

In order to investigate the performance of the proposed hybrid approach for the TSP, experiments are carried out to compare it with other relevant algorithms on a set of small and large benchmark TSP instances. Experimental results show that the proposed hybrid approach is superior to the IO algorithm in terms of the convergence speed solution quality and time as well.

The rest of the paper is outlined as follows. Section II describes the proposed hybrid approach, including the SBGA and the IO operator. Section III presents the experimental study. Finally, concludes this paper with discussions on future work.

The authors are with the Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, United Kingdom (email: {saa29, s.yang}@mcs.le.ac.uk).

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of the United Kingdom under Grant EP/E060722/1.

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

Algorithm 1 Sequence Based GA (SBGA)

```
Initialize Pop of the size popsize
for each individual ind_i ∈ Pop do
    ind_i := 2-Opt(ind_i, K)
end for
repeat
    Adapt the size of sequences L_seq
    GenerateSequence(Num_seq)
    mating_pool := TournamentSelect(Pop)
    // Crossover
    for j := 0 to popsize do
        Select two parents i_a and i_b from the mating_pool
        if (rand(0,1) < p_c) then
            Create child_a and child_b by e − SBOX(i_a, i_b, FreqOfinsertion)
            Add child_a and child_b to Pop_tmp
        end if
    end for
    // Mutation
    for each individual ind_i ∈ Pop_tmp do
        if (rand(0,1) < p_m) then
            e − SBIM(ind_i, inversions, FreqOfinsertion)
            Add ind_i to Pop_tmp
        end if
    end for
    Pop := SelectNewPop(Pop + Pop_tmp)
until Termination condition = true
```

II. The proposed hybrid approach for the TSP

A. Phase 1: SBGA with embedded local search

As mentioned before, the first phase of the hybrid approach is based on the SBGA proposed in [1]. The structure of the SBGA used in this paper is shown in Algorithm 1. 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 [9], [8]. In SBGA, sub-tours are used to construct the whole tour. Similar work has been done in [10], where an individual is broken into parts and then reconnected in a random way.

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 for SBGA. Then, a set of Num_{seq} sequences are generated as follows. A set of best individuals are selected from the population. Each selected individual is then broken into sub-tours, each of which has the same number of cities. The sub-tour with the shortest length is selected, further optimized by a 2-opt improver [5], and then stored in a sequence set. This set of sequences will be used to guide the crossover and mutation by putting the sequence in a proper location among a set of random locations.

After the construction of sequences, a mating pool is generated using the tournament selection with the tournament size 3. Then, crossover and mutation are performed based on the set of sequences to generate offspring. In the proposed hybrid approach, we integrate the local search operator as an additional part within crossover and mutation not a separate one as in our previous study [1]. Simply, the local search runs after each crossover and mutation operation to improve the fitness of newly created child. The details of each operation are given in the following subsections, respectively.

1) Sequence generation: In SBGA, after the set of sequences are initialized, we update the set of sequences when the fitness of the best individual of the population improves. For the generation or update of the set of sequences, a certain percentage of the best individuals from the population are selected. Each of these selected individuals is checked one by one to generate sequences as follows. An individual is broken into equal parts (sub-tours), each 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 first node $i = 0$ and goes until the node $(i < n - S_{seq})$, where S_{seq} represents the size of a sequence, i.e., the total number of nodes in a sequence (sub-tour).

Algorithm 2 $GenerateSequence(Num_{seq})$

```
Select the best individuals from the population and store them in Best_indi
for i := 0 to Num_seq do
    Split Best_indi[i] into n sub-tours, each with the same number of nodes
    Calculate the length of each sub-tour
    Further optimize the sub-tour with the minimum length by a 2-Opt improver
    Store the sub-tour into the set of Seq_Numseq
end for
```

Algorithm 3 $e - SBOX(i_a, i_b, F_{insert})$

```
Randomly select S_sel;
Remove the nodes of S_sel from the nodes_ind of individual i_a and i_b
Perform OX on the remaining nodes_ind of i_a and i_b to create children child_a and child_b
Apply SBLS(child_a, S_sel, F_insert)
Apply SBLS(child_b, S_sel, F_insert)
Evaluate(child_a, child_b)
```

For example, given an individual ABCDEFGHIJKLMNOP and the number of nodes in a sequence $S_{seq} = 4$, then the candidate sequences for the shortest path in this individual are *ABCD*, *BCDE*, $\cdots$, *MNOP*. In this spliting procedure, one node comes in and one goes out. So almost every node participates. The length of each sub-tour is calculated. Suppose the shortest sequence is *BCDE*. Then, *BCDE* is further optimized by the 2-Opt improver to, say, *CDBE*. Finally, this sequence *CDBE* is stored in a set of sequnces Seq_{Numseq}. The same procedure applies for all selected top individuals to generate Num_{seq} sequences.

2) embedded-Sequence Based Order Crossover (e-SBOX): The Order Crossover (OX) [2], [4] is a sexual reproduction operator. It is the variant of "two point crossover" and 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.

In SBGA, we use a modified OX operator, the embedded-sequence based OX (e-SBOX). The pseudocode of e-SBOX is shown in Algorithm 3. e-SBOX works as follows. First, a random sequence S_{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 those nodes of the sequence S_{sel} are removed from both of the individuals and crossover occurs between the remaining partial individuals to create two partial children. These two partial children then undergoes sequence based local search (SBLS) to create two complete children.

e-SBOX is illustrated using the following example. Let i_a and i_b represent the parents ($P1$, $P2$) and $child_a$, $child_b$ represent the children ($C1$, $C2$). Let the sequence S_{sel} be *CDBE*, then the crossover performs as follows:

Algorithm 4 $e-SBIM(i_m, inversions, F_{insert})$

```
Randomly select S_sel
S'_sel := reverse selected S_sel
i_temp := remove the cities in S_sel from individual i_m
for i := 0 to inversions do
    Randomly select two points p1 and p2 (p1 < p2)
    i'_temp := inverse cities in between points p1 and p2 of i_temp
    if f(i'_temp) < f(i_temp) then
        i_temp := i'_temp
    end if
end for
Apply SBLS(i_temp, S'_sel, F_insert)
Evaluate(i_temp)
i_m := i_temp
```

Select Perents:
P1:=ABCDEFGHIJKLMNOP
P2:=PONMLKJIHGFEDCBA
Remove (CDBE):
$P1_{tmp}$:=AFG | HIJK | LMNOP
$P2_{tmp}$:=PON | MLKJ | IHGFA
After crossover:
$C1_{tmp}$:=GFAHIJKPONML , $C2_{tmp}$:= NOPMLKJAFGHI
Apply:
$SBLS(C1_{tmp}, S_{sel}, F_{insert})$, $SBLS(C1_{tmp}, S_{sel}, F_{insert})$
$C1$:=GFAHIJK**CDBE**PONML
$C2$:= NOPMLKJA**CDBE**FGHI

The parameter F_{insert} supplied to the crossover operator denotes the percentage of random locations that will be checked for inserting the senquence S_{sel} in an individual. The F_{insert} varies from 5% to 15% to the size of a TSP instance. For example, for the $eil101$ TSP instance used in the experimental study, 5 random locations will be checked if $F_{insert} = 5\%$ since there are 101 cities in the $eil101$ TSP instance.

3) embedded-Sequence based inversion mutation (e-SBIM): After crossover, each offspring undergoes mutation with a small probability p_m. For TSPs, the Simple Inversion Mutation (SIM) operator is one of the best performers [4]. In our approach, we perform embedded sequence based inversion mutation (e-SBIM) on an individual for some iterations, and preserve those inversions which have a positive effect on the performance. This increases the convergence speed although involving an extra overhead on the mutation operator. The number of iterations of e-SBIM depends on whether the best fitness changes. If the best fitness changes for each generation, e-SBIM will not execute. So, the number of executions will be in the range $[0, S_{seq}]$. The details of e-SBIM is shown in Algorithm 4.

The overall procedure of e-SBIM is similar to that of e-SBOX. The difference lies in that e-SBIM inverts the sequence before inserting it in an individual. Here, if the passing parameter $inversions$ is equal to 0, e-SBIM will not execute; otherwise, e-SBIM will be executed for $inversions$ iterations by selecting two random points in the remaining nodes of the individual and performing the inversion. Then, the fitness of the resultant individual i'_{temp}, $f(i'_{temp})$, is calculated. If it is better than the fitness $f(i_{temp})$ of i_{temp}, that inversion is accepted; otherwise, the inversion is rejected. Thereafter, SBLS is executed, where the inverted S_{sel} is inserted into a best position among a set of different random positions. Our approach guarantees possibly fruitful individual as the position to insert the inversed sequence is optimized.

A demonstration of a mutation operation is given below, where i_m denotes the parent P and i_{temp} denotes the child C.

Algorithm 5 $SBLS(ind_i, S_{sel}, F_{insert})$

```
X := 2-Opt(ind_i, K)
Create a set of |X| × F_insert random locations, where |X| denotes the
number of cities in X
Find the best_position in X which gives the minimum length increase
after inserting S_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 in the TSP and M is the number
of cities in S_sel
ind_i := insert S_sel into X at the position best_position
Evaluate(ind_i)
```

Parent: P :=ABCDEFGHIJKLMNOP
After removing (ECDB): P_{tmp} :=AFG | HIJKLMN | OP
After inversion: C_{tmp} :=AFG | MNLKJIH |OP
Apply: $SBLS(C_{tmp}, S_{sel}, F_{insert})$
C :=AFGMNLKJIH**BDCE**OP

4) Sequence based local search (SBLS): Local search (LS) is an efficient heuristics for combinatorial optimization problems [6]. In SBGA, the set of sequences stored in the memroy is applied in the LS to guide the generation of children towards promising area of the search space. The pseudocode of SBLS is shown in Algorithm 5.

SBLS takes an individual ind_i and first performs 2-Opt improver for K times. Then, SBLS finds the best position from a set of randomly selected positions to insert a selected sequence into ind_i. The distance between the first and last nodes of the sequence S_{sel} is calculated according to the distance matrix relevant to the adjacent nodes of the individual ind_i where the sequence may be inserted. The position corresponding to the minimum length increase value Inc_{Length} is used to insert the sequence S_{sel} to ind_i.

5) Adapting Parameters and Maintaining the diversity: In SBGA, we use adaptive techniques to adapt several key

parameters, including the step size K for the 2-Opt improver used in SBLS, the size of sequences S_{seq}, and the crossover and mutation probabilities. For the first parameter, K is initially set to 5. When the best fitness of the population does not imporve, the value of K is increased by 5 until it reaches 20. If the best fitness of generation improves, K is reset to 5.

The size of sequences S_{seq} stored in the memory is adapted also according to whether the best fitness of generation improves. Initially, S_{seq} is set to the value of $\lfloor N/\sqrt{N} \rfloor$, where N is the total number of cities in the TSP and $\lfloor x \rfloor$ returns an integer nearest or equal to x. We use a variable t_{us} to denote when to update S_{seq}, which is initialized to 20. When the best fitness of the population does not imporve, t_{us} is decreased by one. When $t_{us} = 15$, we set $S_{seq} := \lfloor 0.75 \times N/\sqrt{N} \rfloor$. When t_{us} is further reduced to 10, we set $S_{seq} := \lfloor 0.5 \times N/\sqrt{N} \rfloor$. When t_{us} is further reduced to 5, we set $S_{seq} := \lfloor 0.25 \times N/\sqrt{N} \rfloor$. When t_{us} is further reduced to 0, $S_{seq} := 2$, which means a sequence will become an edge (i, j) and SBGA searches for the shortest edge and re-inserts it in a proper position of an individual in SBLS. If the best fitness of the population improves, t_{us} is reset to 20 and S_{seq} is reset to $\lfloor N/\sqrt{N} \rfloor$.

Algorithm 6 Inver-Over($Pop(t)$)

```
for each route_i in Pop(t) do
    route* := route_i
    select randomly a city C from route*
    while TRUE do
        if rand(0,1) < p then
            select the city C* from the remaining cities in route*
        else
            select randomly a route from the population
            assign to C* the next city to C in the selected route
        end if
        if the next or previous city of city C is C* in route* then
            exit from the while loop
        end if
        inverse the section from the next city of city C to city C* in
        route*
        C := C*
    end while
    if Length(route*) < Length(route_i) then
        route_i := route*
    end if
end for
```

We also adapt the crossover probability p_c and mutation probability p_m as follows. Initially, we set $p_c = 0.55$ and $p_m = 0.05$. If the best fitness of the population does not imporve, we increase p_c with a step size 0.05 until it reaches 0.8 and p_m with a step size 0.005 until it reaches 0.5. If the best fitness of the population improves, p_c and p_m are reset to their initial values.

B. Phase 2: The modified Inver over (IO) algorithm

We have proposed two simple modification to the IO algorithm. we have explained these modification along with the origional IO algorithm in the following text repectively.

1) The Inver Over: Inver-over [11] is a smart operator based on simple inversion. However, adaptive in nature, knowledge taken from other individuals in the population. IO is an unary operator, since the inversion is applied to a segment of a single individual, however, the selection of a segment to be inverted is population driven, thus the operator displays some characteristics of crossover/recombination. The outline of the IO operator is shown in Algorithm 6 and it works with only two parameters, the population size and the probability of random inversion $p = 0.02$.

2) Restricted Inver Over: It is clear from the above Algorithm 6, that the main loop terminates only and only if the next or previous city of city C is C^* in $route^*$ then exit from the main loop. It does not consider which inversion contributes in fitness gain or not. We have made the inversion restricted by shifting the evaluating part of Algoithm 6 into the main *while()* loop before the assignment of $C := C^*$. At fist it seems to be an extra overhead on the IO and this would increase the execution time. But it should be interesting to compare with original IO or proposed restricted IO not even decrease the computation time but also contributes in good solution quality. From Table I this comparison is quite clear.

3) Restricted Inver Over with partial random initialization: As mentioned above, SBGA performs well at the early stage of evolution and do fast conversion which is obvious from Fig.3, which can get good fitness but reduce the diversity as well. So, when IO is employed in the second phase of our hybrid approach, it not only brings diversity but also contributes better in obtaining good results in terms of fitness.

Since the diversity of the population affects the performance of IO greatly, in our hybrid approach, before giving control to IO, along with previous parent and child populations, some percentage of random individuals are injected into the population. If the $popsize$ is 30, then the total size of the population for IO in the second phase will be 90 (30 parents, 30 children, and 30 random individuals respectively) and for large benchmark the population size is 60 (20 parents, 20 children, and 20 random individuals respectively). This approach is denoted by SBGA+IO+RI in this paper.

III. Experimental Study

A. Experimental setting

In this section, we present the experimental results of the proposed hybrid approach SBGA+IO+RI in comparison with other three relevant algorithms, which are the IO algorithm [11], the original SBGA proposed in [1], and SBGA+IO that is the proposed hybrid approach but without adding random individuals into the population when the second phase (i.e., IO) is started.

The proposed approach was implemented in C++ on a $2.66\ GHz$ PC under the Windows Visual Studio environment. All TSP problem instances (except CHN144) are obtained from TSPLIB[1] for the symmetric TSP. The number of cities in these cases varies from 51 to 144 for small problems and from 318 to 1173 for large problems. The SBGA is not perfomed for large problems, because it takes

[1]Available from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

TABLE I

THE EXPERIMENTAL RESULTS OF IO, SBGA, SBGA+IO, AND SBGA+IO+RI

Instance	Measure	IO	SBGA	SBGA+IO	SBGA+IO+RI
EIL76 (538)	Best	544.369	553.097	546.886	544.369
	Err	0.0118	0.0280	0.0165	0.0118
	Avg	550.304	557.2756	556.401	**548.294**
	Err	0.0228	0.0358	0.0342	**0.0191**
	Avg Time	6.50	19.00	4.80	5.7
EIL101 (629)	Best	644.275	663.867	645.919	645.205
	Err	0.0242	0.0554	0.0268	0.0257
	Avg	652.851	677.73	653.879	**651.444**
	Err	0.0379	0.0774	0.0395	**0.0356**
	Avg Time	6.80	23.00	5.60	6.3
KROA100 (21282)	Best	21285.4	21890.66	21285.4	21285.4
	Err	0.0001	0.0285	0.0001	0.0001
	Avg	21328.8	21896.66	21430.5	**21321.7**
	Err	0.0021	0.0288	0.0069	**0.0018**
	Avg Time	6.80	23.09	5.60	6.16
KROC100 (20749)	Best	20769.9	21732.1	20750.8	20750.8
	Err	0.0010	0.0473	0.00001	0.00001
	Avg	20879.1	21884.96	20921.7	**20822.1**
	Err	0.0062	0.0547	0.0083	**0.0035**
	Avg Time	6.70	23.00	5.40	6.2
LIN105 (14397)	Best	14397	14755	14397	14397
	Err	0	0.0248	0	0
	Avg	14446.5	15276.7	14505.2	**14426.4**
	Err	0.0034	0.0611	0.0075	**0.0020**
	Avg Time	6.90	24.50	5.60	5.7
CHN144 (30347)	Best	31388.1	32169	30661.1	**30353.9**
	Err	0.0343	0.0600	0.0103	**0.0002**
	Avg	31681.6	33470.9	30953.9	**30698.7**
	Err	0.0439	0.1029	0.0199	**0.0115**
	Avg Time	6.96	17.00	5.00	7.00
LIN318 (42029)	Best	43045.5	– –	42831.6	**42964.4**
	Err	0.02419	– –	0.01909	**0.0222**
	Avg	43174.8	– –	42955.8	**43070**
	Err	0.0272	– –	0.0220	**0.0247**
	Avg Time	47.35	– –	47.84	37.36
PCB442 (50778)	Best	55625.7	– –	51868.3	**51866.9**
	Err	0.0954	– –	0.0214	**0.0214**
	Avg	55868.9	– –	52013.46	**52236.8**
	Err	0.1002	– –	0.0243	**0.0287**
	Avg Time	74.26	– –	61.78	55.36
RAT575 (6773)	Best	12096	– –	7018.98	**7031.91**
	Err	0.7859	– –	0.0363	**0.03822**
	Avg	12721.4	– –	7031.45	**7048.58**
	Err	0.8782	– –	0.0381	**0.0406**
	Avg Time	110.36	– –	98.08	76.29
RAT783 (8806)	Best	34093.5	– –	9526.48	**9218.27**
	Err	2.8716	– –	0.05115	**0.0468**
	Avg	34946.3	– –	9267.84	**9244.28**
	Err	2.9684	– –	0.05244	**0.0497**
	Avg Time	190.57	– –	72.33	106.06
U724 (41910)	Best	140569	– –	43304.2	**43441**
	Err	2.3540	– –	0.0332	**0.0365**
	Avg	145847	– –	43519.3	**43485.8**
	Err	2.4800	– –	0.03839	**0.0376**
	Avg Time	157.80	– –	139.94	92.84
V1084 (239297)	Best	2.26e+06	– –	250757	**252305**
	Err	8.4344	– –	0.0478	**0.0543**
	Avg	2.29e+06	– –	251469	**252955**
	Err	8.5698	– –	0.0508	**0.0570**
	Avg Time	332.80	– –	196.98	159.20
PCB1173 (56982)	Best	432382	– –	60223.4	**60055.9**
	Err	6.5880	– –	0.0568	**0.0539**
	Avg	440485	– –	60476	**60481**
	Err	6.7302	– –	0.0613	**0.0614**
	Avg Time	347.17	– –	217.36	172.37

longer time which can be seen from Table: I. The parameters for the algorithms were set as follows. The population size

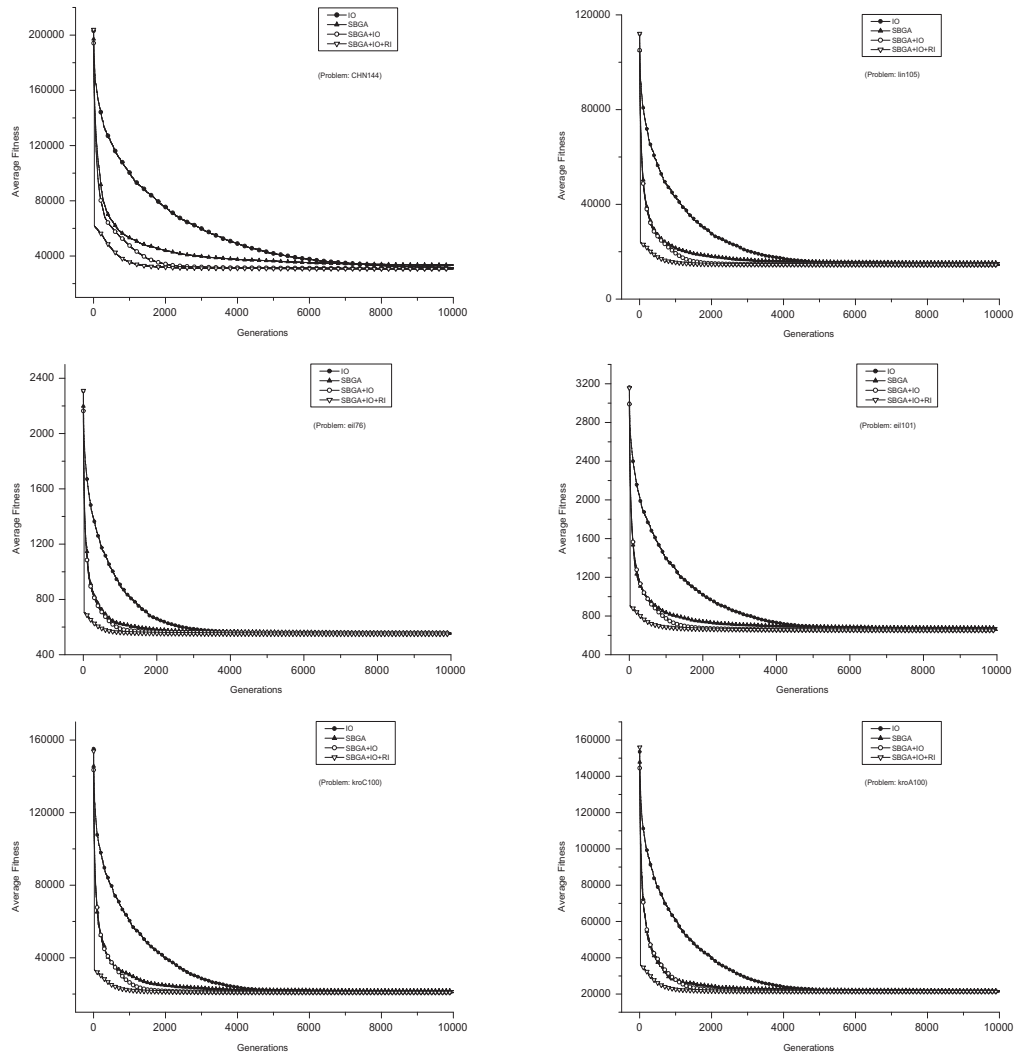

Fig. 1. Experimental results of IO, SBGA,SBGA+IO, and SBGA+IO+RI. The effect of our approach has more additive improvements the original IO.

was set to 30 for the first eight TSP instances in the four techniques. For IO and SBGA+IO the population size is 60 in large problems. And for SBGA+IO+RI algorithm, initialy the population size was set to 20 for the first phase and for second phase the population size were increased to 60 for making the population size consistent for the three techniques. The crossover probability and mutation probability were initially set to $p_c = 0.55$ and $p_m = 0.055$, respectively, which are adapted by a small value to a maximum value when the best fitness of the generation does not change. The percentage of random locations to insert a sequence, F_{insert}, was set to 5% for these experiments.

B. Experimental results and analysis

In Table I, we present the averaged results of IO, SBGA, SBGA+IO, and SBGA+IO+RI over 30 independent runs. In this table, "Best" denotes the best tour found and "Avg" denotes the average fitness over 30 runs. The "Err" rows give relative deviation to the global optimal fitness listed in the table after the instance name. Finally, "Avg Time" is the average time used by algorithms in seconds. Fig. 1 shows the dynamic performance of algorithms regarding the average fitness against the number of generations.

From Fig.1 and Fig.2 the behaviour of SBGA shows that SBGA gives a better convergence speed at the initial stage of the solving progress. The convergence speed is also visible from Fig.3, which is ploted from $10k$ evaluations of both the algorithms for *chn144* and *pcb1173* Then, it behaves similar to the IO operator. In terms of the number of evaluations, the ratio between IO and SBGA is 1:3, as SBGA using the traditional binary operator with an additional embedded SBLS. But due to adaptive behaviour of IO, it gives a better solution quality at the later stage of the hybrid approach. The hybrid approach combines both the features of SBGA and IO. First, SBGA brings the fitness to a near-optimal level in a few generations and then IO further works and improves

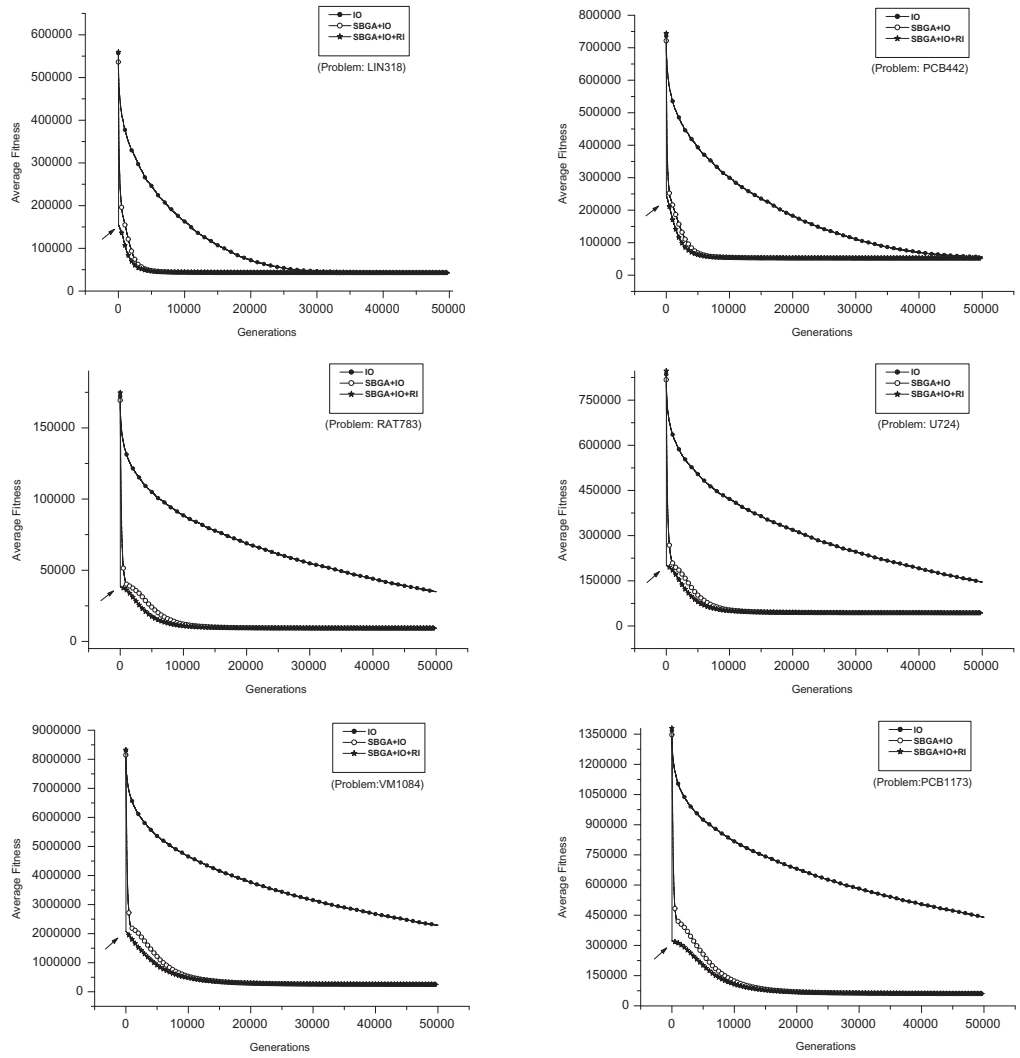

Fig. 2. Experimental results of IO,SBGA+IO, and SBGA+IO+RI. The effect of our approach is more prominent in larger problems. The arrow shows the switchover from Phase:1 to Phase:2.

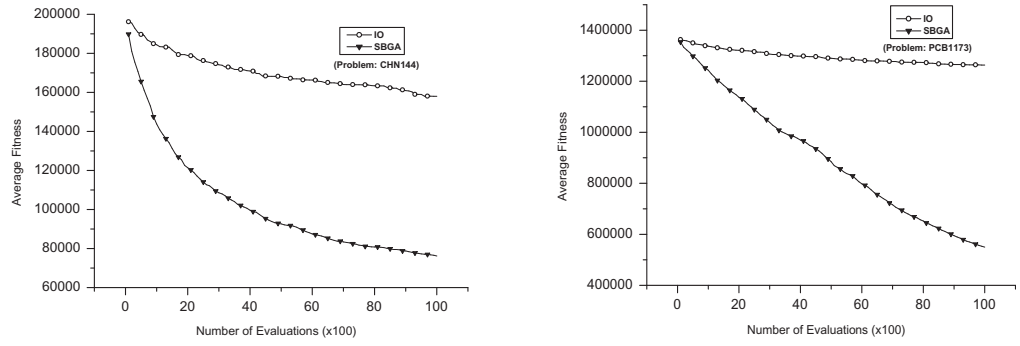

Fig. 3. Evaluation results of IO,SBGA. The best after 100 is recorded with 10k evaluations , which shows the convergence of SBGA vs IO.

the fitness to give a better solution quality.

It can also be observed that when the new population is selected from parent and child, lots of information is lost. So, by keeping parent and child population and introducing some random individuals,the SBGA+IO+RI outperforms SBGA and SBGA+IO. From Table 1, it can be seen that SBGA+IO+RI achieves better solutions than IO on all test instances, while SBGA+IO is slightly worse (first eight small and medium instances), but comparatively better then SBGA. However, from the results of Table 1 and Fig.1, we can see that our hybrid approach SBGA+IO+RI outperforms IO regarding both the convergence speed and solution quality and time.

It should be interesting to compare original IO and our various proposed approaches which is mentioned above. Figure 1 and 2 shows the plots between average tour length and generation for 13 problems. The three types of refinement can enhance the performance, both converged more rapidly than original IO. The use of SBGA, restrictive IO and RI also have additive effects on performance gain and the contribution is dominating. From these preliminary results, one may speculate that our approach is more effective and increase the *"Adaptive Power"* of the IO which is not fully contributed by the original IO in case of small as well as in large TSPs instances.

In terms of computational time, it is obvious that performing extra steps in our proposed approach would increase the execution time. But from Table I the characteristics are totally opposite. The IO takes longer time on all the instances small and large. In case of rat575,rat783,u724,v1084 and pcb1173 the IO is unable to achieve acceptable fitness, but due the additive effect of our approaches with IO, it not only gets better fitness but the execution time is reduced remarkably. We may speculate that most of the time in wasted in inversions which is not fruitful regarding fitness gain.

From Fig 1 and Fig 2, various algorithms are shown in different line styles. It is obvious from the plots that our proposed approaches are not overlapping. Which means that each and every refinement can contribute to additional performance gain. These contributions are more effective for large problems. However, with these refinement not only decrease the error rate but the CPU time used are reduced almost in all problems.

Conclusion and Future Work

This paper proposes a hybrid approach for the TSP based on a sequence based genetic algorithm (SBGA) and IO operator. In the first phase, SBGA is used with an embedded local search scheme to solve the TSP. Within the SBGA, a memory is introduced to store good sequences extracted from previous good solutions. The stored sequences are used to guide the generation of offspring during the crossover and mutation operations. After each crossover and mutation operation, a sequence based local search scheme runs to improve the fitness of newly created child. Some effective ideas are proposed for adapting the key parameters and maintaining the population diversity. After SBGA finishes, the second phase uses the inver over (IO) algorithm [11] with some extra refinements i-e. restrictive inversions and random immigrants like scheme, (IO) which is a state-of-the-art algorithm for the TSP, to further improve the solution quality of the population.

In order to investigate the performance of the proposed hybrid approach for the TSP, experiments are carried out to compare it with three relevant algorithms on a set of benchmark TSP instances. Experimental results show that the proposed hybrid approach is superior to the IO algorithm and the original SBGA regarding both the convergence speed and solution quality on most test TSP instances.

There are several issues for future research. Firstly, our concept totally depends on the formation of the set of sequences. It is interesting to further study when a new set of sequences should be introduced to SBGA in the first phase of our hybrid approach. Secondly, during the experiments it has been observed that some of the sequences contain redundant information which causes the premature convergence at some stage of the solving process. Finally, in this study, we investigate the hybrid approach for solving small, medium and large scale TSP instances. However, for larger scale instances, more time is needed and the speed is comparatively slow. One future research would be to study more complex TSPs and related problems like asymmetric TSPs, sequential ordering, and capacitated vehicle routing etc.

References

[1] S. Arshad, S. Yang, and C. Li. A sequence based genetic algorithm with local search for the travelling salesman problem. *Proc. of the 2009 UK Workshop on Computational Intelligence*, pp. 98–105, 2009.

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

[3] 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.

[4] 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.

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

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

[7] 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.

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

[9] 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.

[10] 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.

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