A HYBRID GENETIC ALGORITHM – A NEW APPROACH TO SOLVE TRAVELING SALESMAN PROBLEM

G.ANDAL JAYALAKSHMI
Computer Science and Engineering Department, Thiagarajar College of Engineering, Madurai, Tamilnadu, India
Email: andal_m@yahoo.com
S.SATHIAMOORTHY
Computer Science and Engineering Department, Thiagarajar College of Engineering, Madurai, Tamilnadu, India
Email: s.sathiamoorthy@sify.com
and
R.RAJARAM
Information Technology Department, Thiagarajar College of Engineering, Madurai, Tamilnadu, India
Email: andal_m@yahoo.com

This paper introduces three new heuristics for the Euclidean Traveling Salesman Problem (TSP). One of the heuristics called ***Initialization Heuristics*** **(IH)** is applicable only to the Euclidean TSP, while other two heuristics ***RemoveSharp*** and ***LocalOpt*** can be applied to all forms of symmetric and asymmetric TSPs. A ***Hybrid Genetic Algorithm*** **(HGA)** has been designed by combining a variant of an already existing crossover operator with these heuristics. One of the heuristics is for generating initial population, other two are applied to the offspring either obtained by crossover or by shuffling. The last two heuristics applied to offspring are greedy in nature, hence to prevent getting struck up at local optimum we have included proper amount of randomness by using the shuffling operator. We studied the effect of these heuristics by conducting experiments, which show that the results obtained by our Hybrid GA outperformed the results obtained by existing GA in certain problems. These heuristics matched "Best Known" solutions in most cases. In others it produced results with one% tolerance, when compared with those of nature-inspired algorithms such as Simulated Annealing (SA), Evolutionary Computation (EP) and Ant Colony System (ACS). Implementation of these heuristics is simple. Our convergence rate is found to be high and the optimal solution is obtained in a fewer number of iterations.

Keywords: Initialization Heuristics; RemoveSharp; LocalOpt; Hybrid Genetic Algorithm

1. Introduction

The Euclidean Traveling Salesman Problem (TSP) involves finding the shortest Hamiltonian Path or Cycle in a graph of *N* cities. The distance between the two cities is just the Euclidean distance between them. This problem is a classic example of ***Non Polynomial-hard*** problem and is therefore impossible to search for an optimal solution for realistic sizes of *N*. This motivated many researchers to develop heuristic search methods for searching the solution space. The heuristics are widely accepted even though they produce sub-optimal solutions, because they converge in polynomial time. The TSP is probably the most-studied optimization problem of all time. Applications of TSP include Circuit board drilling applications with up to 17,000 cities[4], X-ray crystallography instances with up to 14,000 cities[4] and instances arising in VLSI fabrication have been reported with as many as 1.2 million cities[4]. Moreover, 5 hours on a multi-million dollar computer for an optimal solution may not be cost-effective if one can get sub optimal solutions with acceptable error tolerance in seconds on a Personal Computer. Thus there remains a need for heuristics[4].

The theoretical foundations of genetic algorithms assume that there exists some (ideally binary) representation of a problem that can be manipulated by genetic operators. Each encoding is referred to as a “genotype”. When the problem can be represented as an n-bit string, it can be shown that genetic algorithms sample hyperplanes in an n-dimensional hypercube. When problem specific information exists it is advantageous to consider a Hybrid GA. They combine local search heuristics with crossover operators. Genetic algorithms may be crossed with various problem-specific search techniques to form a hybrid algorithm that exploits the global perspective of the GA and the convergence of the problem-specific technique. Hybrid Genetic Algorithms is a population-based approach for heuristic search in optimization problems. They execute orders of magnitude faster than traditional Genetic Algorithms for some problem domains.

This paper is organized as follows. In section 2 we describe the proposed Hybrid Genetic Algorithm (HGA) for solving the TSP. In section 3 the crossover algorithm used in the HGA is given. In section 4 the IH is explained. Section 5 introduces the RemoveSharp heuristic algorithm with an analysis of its time complexity. Section 6 describes the LocalOpt algorithm and its time complexity is also analyzed. Section 7 is dedicated to the study of some characteristics of the Hybrid GA, this includes the study of

- how the IH results in quicker convergence.
- optimal size of the population to be used.
- optimal probability of the shuffling operator.
- optimal size for the *NEARLIST* (introduced in Section 5).
- the parameter value for *LocalOpt.*

Section 8 gives implementation details and provides an overview of results on a set of standard test problems. Comparisons of the results obtained using HGA with results by well-known algorithms like Ant Colony System, Evolutionary Computation, Genetic Algorithm and Simulated Annealing have been done. The results are also compared with 'Best Known' results available in the TSPLIB site[3]. In the final section, we conclude the paper with a summary of observations.

2. The Hybrid Genetic Algorithm

The Hybrid Genetic Algorithm is designed to use heuristics for Initialization of population and improvement of offspring produced by crossover. The *InitializationHeuristics* algorithm is used to initialize a part of the population; remaining part of the population will be initialized randomly. The offspring is obtained by crossover between two parents selected randomly. The tour improvement heuristics: *RemoveSharp* and *LocalOpt* are used to bring the offspring to a local minimum. If cost of the tour of the offspring thus obtained is less than the cost of the tour of any one of the parents then the parent with higher cost is removed from the population and the offspring is added to the population. If the cost of the tour of the offspring is greater than that of both of its parent then it is discarded. For shuffling, a random number is generated within one and if it is less than the specified probability of the shuffling operator, a tour is randomly selected and is removed from the population. Its sequence is randomized and then added to the population. The algorithm works as below:

Step 1 :

- ♦ Initialize a part of population using *InitializationHeuristics* algorithm
- ♦ Initialize remaining part of population randomly

Step 2 :

- ♦ Apply *RemoveSharp* algorithm to all tours in the initial population
- ♦ Apply *LocalOpt* algorithm to all tours in the initial population

Step 3 :

- ♦ Select two parents randomly
- ♦ Apply Crossover between parents and generate an offspring
- ♦ Apply *RemoveSharp* algorithm to offspring
- ♦ Apply *LocalOpt* algorithm to offspring
- ♦ If TourCost(offspring) < TourCost(any one of the parents) then replace the weaker parent by the offspring

Step 4 :

Shuffle any one randomly selected tour from population

Step 5 :

Repeat steps 3 and 4 until end of specified number of iterations.

3. Crossover

The crossover operator that is used here is a slight variant of the crossover operator devised by Darrell Whitley[2]. The crossover operator uses an "edge map" to construct an offspring which inherits as much information as possible from the parent structures. This edge map stores information about all the connections that lead into and out of a city. Since the distance is same between any two cities, each city will have atleast two and atmost four edge associations (two from each parent).

3.1 *The crossover algorithm*

Step 1 :

Choose the initial city from one of the two parent tours. (It can be chosen randomly or according to criteria outlined in step 4). This is the "current city".

Step 2 :

Remove all occurrences of the "current city " from the left-hand side of the edge map.

Step 3 :

If the "current city" has entries in its *edgelist* go to step 4; otherwise, go to step 5.

Step 4 :

Determine which city in the *edgelist* of the "current city", has shortest edge with the "current city". The city with the shortest edge is included in the tour. This city becomes the "current city". Ties are broken randomly. Go to step 2.

Step 5 :

If there are no remaining unvisited cities, then STOP. Otherwise, randomly choose an unvisited city and go to step 2.

The difference between the Crossover algorithm of *Darrell Whitley* and this is only in the fourth step of the algorithm. He selected the city with least entries in its *edgelist* as the next city, while we choose the city nearest to the current city. This introduces greedy heuristic in the crossover operator too.

4. Initialization Heuristics

The *InitializationHeuristics* (IH) algorithm can be applied only to *Euclidean TSP*. It initializes the population depending upon a greedy algorithm. The greedy algorithm arranges the cities depending on their x and y coordinates.

The tours are represented in linked-lists. First an initial list is obtained in the input order (Input List). The linked-list that is obtained after applying the initialization heuristics is the "Output List". During the process of applying the initialization heuristics all the cities in the "Input List" will be moved one by one to the "Output List".

4.1 *The initialization heuristics algorithm*

Step 1 :

Select four cities, first one with largest x-coordinate value, second one with least x-coordinate value, third one with largest y-coordinate and fourth one with least y-coordinate value. Move them from the "Input List" to the "Output List".

Step 2 :

From among the possible sequences of the four cities find the sequence of minimum cost and change the sequence of four cities in the "Output List" to the minimum sequence.

Step 3 :

Randomize the elements in the "Input List".

Step 4 :

Remove the head element of the "Input List" and insert it into the "Output List" at the position where the increase in the cost of the tour is minimum. Suppose ***M*** is the cost of the tour before insertion and ***N*** be the cost of the tour after insertion. The position of insertion is selected such that ***N-M*** is minimum.

Step 5 :

Repeat Step 4 until all elements in the "Input List" are moved to the "Output List".

Depending on the sorting criteria in Step 3 of the above algorithm various results will be obtained. *RemoveSharp* and *LocalOpt* heuristics are applied to the offspring obtained by this method and added to the initial population. Experiments show that IH results in quicker convergence. The best offspring obtained by the IH varies from 'Best Known' solution to at most 15% error. In case of **Berlin52 (**a 52-city problem**)** and **Eil51 (**a 51-city problem**)** the optimum result was obtained during initialization itself. This shows the robustness of the initialization heuristics.

4.2 *An example*

Figure 1(a) shows a 10-city problem. Figure 1(b) shows the Boundary Tour formed from four extreme cities. Figure 1(c), 1(d), 1(e) & 1(f) shows the four possible tours that can be formed when city 'E' is moved to the “Output List”. It is obvious from the figures that the Tour in Figure 1(e) will result in minimum increase in the cost of the tour in the “Output List”. Similarly other cities will be moved one by one to the “Output List”.

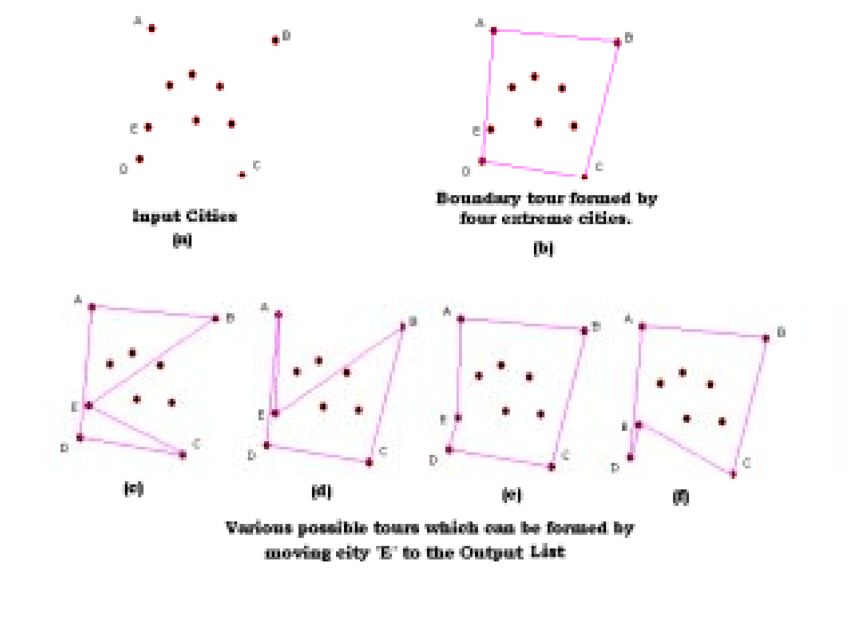

A
B
E
D
C
Input Cities
(a)
Boundary tour formed by four extreme cities.
(b)
(c)
(d)
(e)
(f)
Various possible tours which can be formed by moving city 'E' to the Output List

Fig. 1. IH applied to a 10-city Problem

5. The RemoveSharp Algorithm

The *RemoveSharp* algorithm removes sharp increase in the tour cost due to a city, which is badly positioned. The algorithm works as below:

Step 1: A list (*NEARLIST*) containing the nearest *m* cities to a selected city is created.
Step 2: RemoveSharp removes the selected city from the tour and forms a tour with *N-1* cities.
Step 3: Now the selected city is reinserted in the tour either before or after any one of the cities in *NEARLIST* and the cost of the new tour length is calculated for each case.
Step 4: The sequence, which produces the least cost, is selected.
Step 5: The above steps are repeated for each city in the tour.

5.1 *An example*

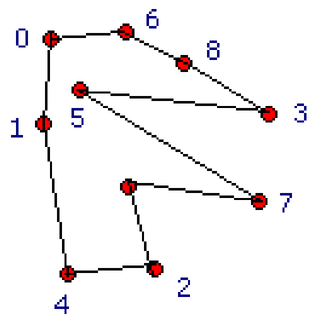


Fig. 2. A tour with a badly positioned city

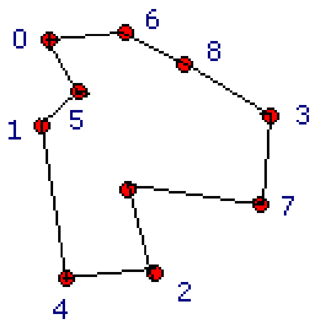


Fig. 3. The tour after *RemoveSharp* is applied

In Figure 2 the city 5 is in between the cities 3 and 7, while it is obvious that the nearest cities to it are city 0, 1, 6 and 8. *RemoveSharp* will move city 5 between the cities 0 and 1, resulting in a decrease in the tour cost as shown in Figure 3.

5.2 *Time complexity of RemoveSharp*

As discussed in Step 2 of the algorithm, when a city is removed during *RemoveSharp* there will be a decrease in the tour cost. Suppose the sequence of the cities be

$$- - - P - C - N - - - - - - - - A_P - A - A_N - - -$$

C is the city to be removed to perform *RemoveSharp*. Let ***P*** be the city previous to the city ***C*** and ***N*** the city next to it. *RemoveSharp* will move the city ***C*** to a new position, if the increase in the tour length after moving it to the new position is less than the decrease in cost caused due to removing it from the position between ***P*** and ***N***. If city ***A*** is in the near list then *RemoveSharp* will check possibility of moving to the locations before ***A*** i.e. A_P and after ***A*** i.e. A_N.

The decrease in tour length will be:

$$DECREASE = Dist(P,C) + Dist(C,N) - Dist(P,N)$$

If ***C*** is moved to the location previous to ***A*** i.e. A_P, increase in tour cost will be:

$$INCREASE_P = Dist(A_P,C) + Dist(C,A) - Dist(A_P,A)$$

If ***C*** is moved the location next to ***A*** i.e. A_N increase in tour cost will be:

$$INCREASE_N = Dist(A,C) + Dist(C,A_N) - Dist(A,A_N)$$

When *RemoveSharp* is applied, ***DECREASE*** is calculated once, while ***INCREASE***$_N$ and ***INCREASE***$_P$ are calculated for every city in the *NEARLIST*. Time complexities for ***DECREASE, INCREASE***$_N$ and ***INCREASE***$_P$ are same and let it be ***x***. ***INCREASE***$_N$ and ***INCREASE***$_P$ should be compared with ***DECREASE*** for each city in the *NEARLIST*. Let ***y*** be the time taken for one comparison. All these calculations need to be done for every city in the tour.

Time Complexity for RemoveSharp $= n * (x + 2m * x + 2m * y)$

Here, ***m*** the size of *NERALIST*, ***x*** the time taken by for ***DECREASE, INCREASE***$_N$ and ***INCREASE***$_P$ and ***y*** the time taken for comparison are constants. Therefore,

Time Complexity for RemoveSharp $\sim= O(n)$

6. The LocalOpt Algorithm

The *LocalOpt* algorithm will select ***q*** consecutive cities $(S_{p+0}, S_{p+1}, \ldots\ldots, S_{p+q-1})$ from the tour and it arranges cities $S_{p+1}, S_{p+2}, \ldots\ldots, S_{p+q-2}$ in such a way that the distance is minimum between the cities S_{p+0} and S_{p+q-1} by searching all possible arrangements. The value of ***p*** varies from ***0*** to ***n-q***, where ***n*** is the number of cities.

6.1 *An example*

In Figure 4 it is quite clear that the distance between the cities 6 and 1 can be reduced if some rearrangements are made in the sequence of the cities between them. *LocalOpt* will make all possible rearrangements and replace them to the sequence as shown in Figure 5.

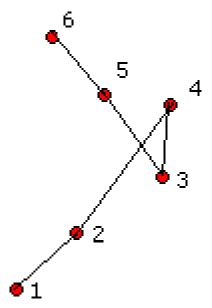


Fig. 4. A bad tour

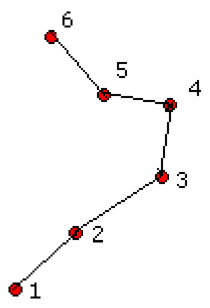


Fig. 5. The tour after *LocalOpt* is applied

6.2 *Time complexity of LocalOpt*

The time complexity of *LocalOpt* varies with value of ***q***, the number of consecutive cities taken for *LocalOpt* at a time. When ***q*** cities in a sequence are considered then all possible combinations of ***q -2*** cities need to be calculated. There will be ***(q-2)!*** combinations, in each case ***q-1*** additions need to be done to evaluate the cost of the sequence and one comparison to check whether the sequence is minimum or not. These need to be done for ***n*** consecutive sequence of ***q*** cities starting from each city in the tour. Therefore,

Time Complexity of LocalOpt = n * (((q-2)!* (q-1)) additions + (q-2)! comparisons)

As ***n*** alone is a variable,

Time Complexity ~= O(n) *(provided q is small (<=6))*

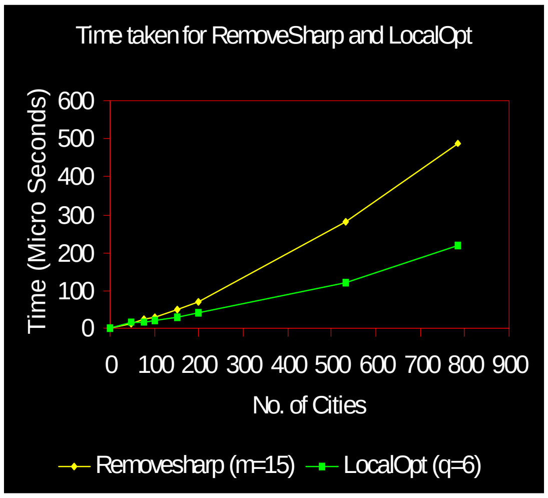


Fig. 6. Time taken by *RemoveSharp* and *LocalOpt*

The time taken by *RemoveSharp* and *LocalOpt* are plotted for 50,75,100,150,200,500 and 800 city problems.

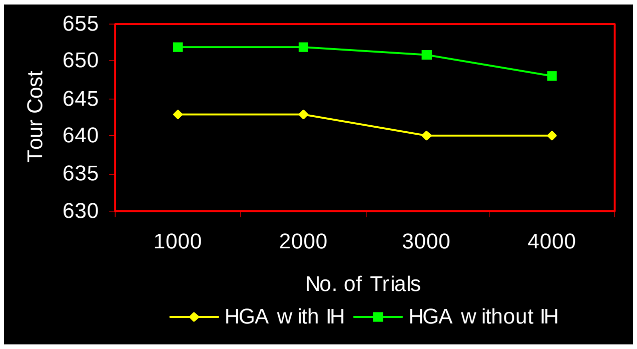
The plot shows that the time taken by *RemoveSharp* and *LocalOpt* are linear in nature. This ensures that the heuristics are best suited for even problems of large size.

7. Analysis of the Characteristics of The Hybrid GA

The effects of various parameters and heuristics are analyzed by keeping the values of the other parameters to the best values. The best performance of HGA is found when the values of the parameters are set as below:

RemoveSharp (m)	:	15	LocalOpt(q)	:	6
Probability of Shuffling operator	:	0.02	Population size	:	50
Initialization	:	IH used			

7.1 *Effects of InitializationHeuristics*



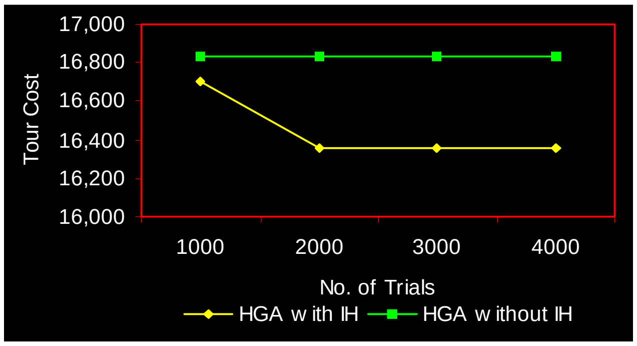


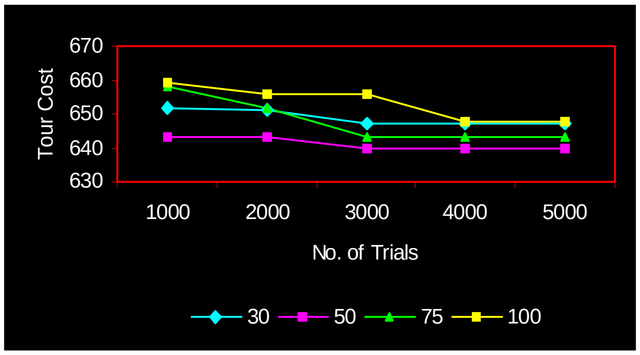
Fig. 7. Tour lengths obtained by HGA for Eil101 and d198 with and without IH

Table 1. Tour lengths obtained by HGA for Eil101 and d198 with and without IH

Condition	Problem Name	Best Integer Tour Length after				
		1000 Trials	2000 Trials	3000 Trials	4000 Trials	5000 Trials
HGA Without IH	Eil101	652	652	651	648	645
	d198	16834	16834	16834	16834	16795
HGA With IH	Eil101	643	643	640	640	**640**
	d198	16701	16357	16357	16357	**16322**

The experiments show that the use of IH for initialization results in faster convergence and better tour lengths.

7.2 *Effect of population size*



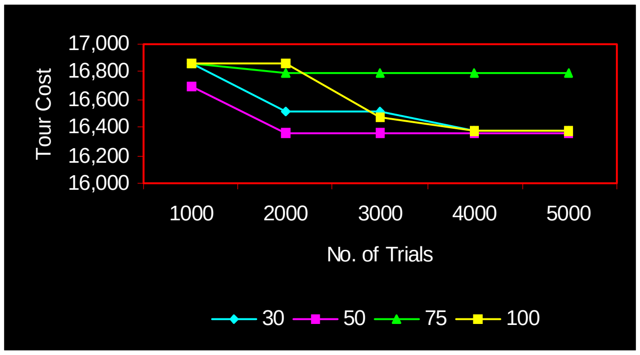


Fig. 8. Tour lengths obtained by HGA for Eil101 and d198 with a population size of 30, 50, 75 and 100

Table 2. Tour lengths obtained by HGA for Eil101 and d198 with a population size of 30, 50, 75 and 100

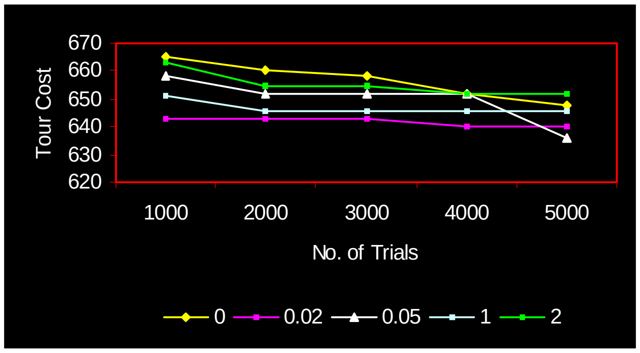
Population Size	Problem Name	Best Integer Tour Length after				
		1000 Trials	2000 Trials	3000 Trials	4000 Trials	5000 Trials
30	Eil101	652	651	647	647	647
	d198	16857	16509	16509	16381	16381
50	Eil101	643	643	640	640	**640**
	d198	16701	16357	16357	16357	**16357**
75	Eil101	658	652	643	643	643
	d198	16857	16791	16791	16791	16791
100	Eil101	659	656	656	648	648
	d198	16857	16857	16479	16373	16373

The experiments show that the optimal value for the size of the population is 50, values less than and greater than 50 result in greater tour length.

7.3 *Effect of shuffling operator*

Table 3. Tour lengths obtained by HGA for Eil101 and d198 with shuffling probability from 0% to 20%

Shuffling Operator probability	Problem Name	Best Integer Tour Length After				
		1000 Trials	2000 Trials	3000 Trials	4000 Trials	5000 Trials
0.00%	Eil101	665	660	658	652	648
	d198	16,791	16,791	16,509	16,509	16,509
2.00%	Eil101	643	643	643	640	640
	d198	16,701	16,357	16,357	16,357	**16,357**
5.00%	Eil101	658	652	652	652	**636**
	d198	16,509	16,509	16,509	16,509	16,509
10.00%	Eil101	651	646	646	646	646
	d198	16,509	16,509	16,509	16,509	16,509
20.00%	Eil101	663	655	655	652	652
	d198	16,480	16,480	16,476	16,476	16,476



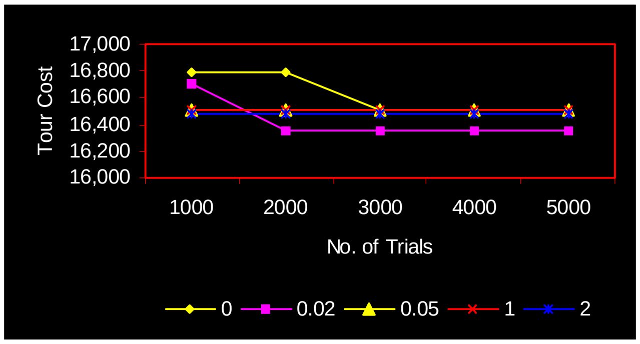
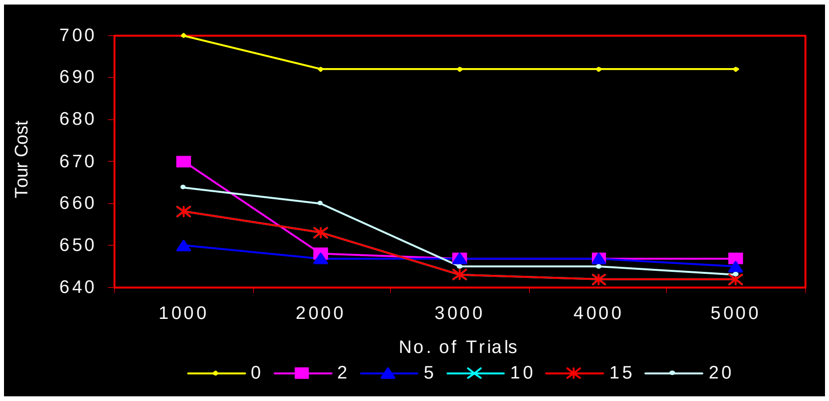


Fig. 9. Tour lengths obtained by for Eil101 and d198 with shuffling probability from 0% to 20%

The optimal value for the shuffling operator probability is found to be 0.02 as the values less than and greater than this do not contribute to the convergence.

7.3 *Effect of Remove Sharp*



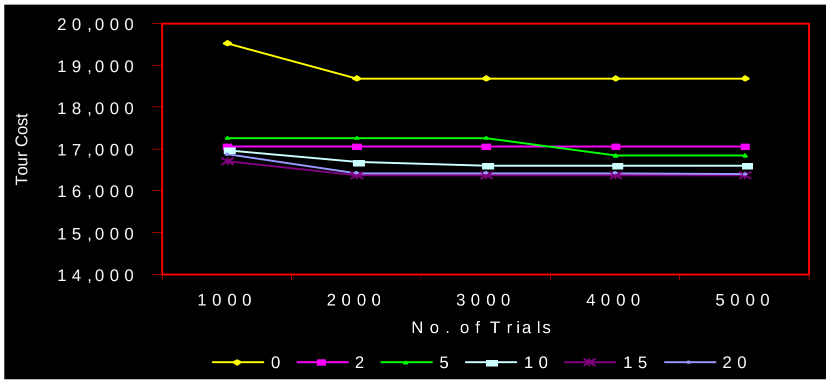


Fig. 10. Tour lengths by HGA for Eil101 and d198 with size of NEARLIST from 0 to 25

Table 4. Tour lengths obtained by HGA for Eil101 and d198 with size of NEARLIST from 0 to 25

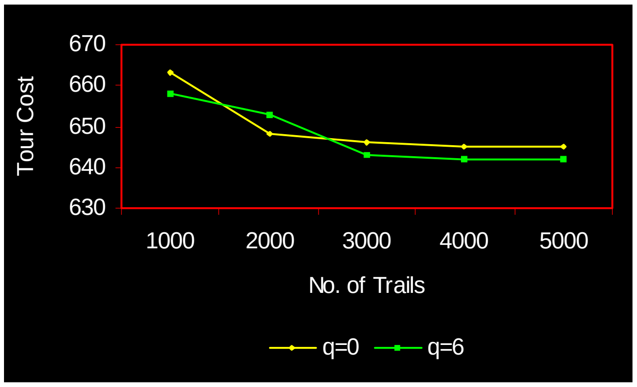
Size of NEARLIST	Problem Name	Best Integer Tour Length After				
		1000 Trails	2000 Trials	3000 Trails	4000 Trails	5000 Trails
0	Eil101	700	692	692	692	692
	d198	19,539	18,688	18,688	18,688	18,688
2	Eil101	670	648	647	647	647
	d198	17,059	17,059	17,059	17,059	17,059
5	Eil101	650	647	647	647	645
	d198	17,260	17,260	17,260	16,846	16,846
10	Eil101	658	653	643	642	**642**
	d198	16,977	16.696	16,599	16,599	16,599
15	Eil101	658	653	643	642	**642**
	d198	16,701	16,357	16,357	16,357	**16,357**
25	Eil101	664	660	645	645	643
	d198	16,857	16,425	16,425	16,425	16,399

The optimal size of NEARLIST is found to be 15 as sizes greater than 15 result in less improvement but takes a large amount of time.

7.4 *Effect of LocalOpt*

Table 5. Tour lengths by HGA for Eil101 and d198 with and without LocalOpt

Condition	Problem Name	Best Integer Tour Length after				
		1000 Trails	2000 Trials	3000 Trails	4000 Trails	5000 Trials
HGA without LocalOpt	Eil101	663	648	646	645	645
	d198	16708	16695	16404	16404	16404
HGA with LocalOpt (q=6)	Eil101	658	653	643	642	**642**
	d198	16701	16357	16357	16357	**16357**



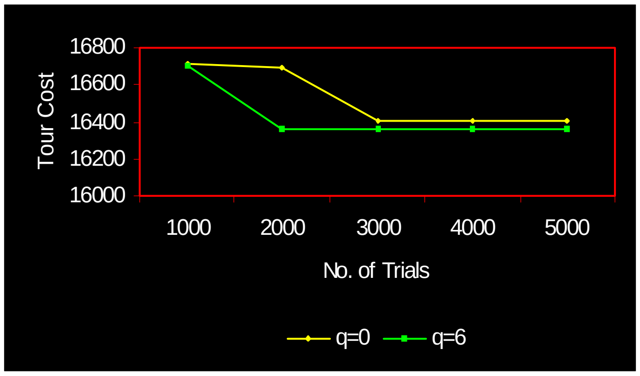


Fig. 11. Tour lengths by HGA for Eil101 and d198 with and without LocalOpt

Table VI shows that the effect of *LocalOpt* is not very high. *LocalOpt* is not tried with values of q>6 as the increase in time complexity is combinatorial.

8. Results and Discussion

Experimental runs of the RemoveSharp algorithm have shown that ***m = 20*** (***m*** is the number of cities in the NEARLIST) results in slight improvement in convergence rate when compared to ***m=15***. But, it takes considerably more amount of time. Hence we chose ***m* = *15***. As shown by the Time Complexity Equation of *LocalOpt*, the time complexity of *LocalOpt* varies combinatorial with the variation of ***q***. We chose ***q* = *6***. The initial population was kept at ***50***, as ***75*** and ***100*** decreased convergence rate and values less than ***50*** resulted in a local optimum. Of the ***50*** initial population ***22*** are obtained using IH (with different sorting criteria) and the rest are randomly generated. The probability of the shuffling operator was kept at ***0.02*** as higher shuffling rate will worsen the convergence rate. The algorithm has been implemented on an *IBM CYRIX 233Mhz* station with *Windows 98* operating system. The program, written in *Visual C++ 6.0* comprises approximately *3,000* lines of source code. The experimental results have been achieved with the machine running its normal daily loads in addition to our algorithm.

Table 6. Comparison of HGA with other heuristics on geometric instances of the symmetric tsp.

Problem name	ACS	HGA	EP	GA	SA	Optimum
Eil50 50-city problem	**425** (427.96) [1830]	426 (428.871) **[538]** **{for Eil51}**	426 (427.86) [100000]	428 (N/A) [25000]	443 (N/A) [68512]	425 (ACS)
Eil75 75-city problem	**535** (542.37) [3,480]	538 (544.36) **[6919]** **{for Eil76}**	542 (N/A) [325000]	545 (N/A) [80000]	580 (N/A) [173250]	535 (ACS)
KroA100 100-city problem	**21282** (21285.44) [4820]	**21282** (21285.44) **[851]**	N/A (N/A) [N/A]	21761 (N/A) [N/A]	N/A (N/A) [N/A]	21282 (ACS & HGA)
d198 198–city problem	15888 (N/A) [585000]	**15849** (15876.38) **[37367]**	N/A (N/A) [N/A]	N/A (N/A) [N/A]	N/A (N/A) [N/A]	15849 (HGA)

We report the best integer tour length, the best real tour length (in parenthesis) and the number of trials required to find the best integer tour length (in square brackets). The optimal length listed in the last column is available only for integer tour lengths. N/A means "Not Available".

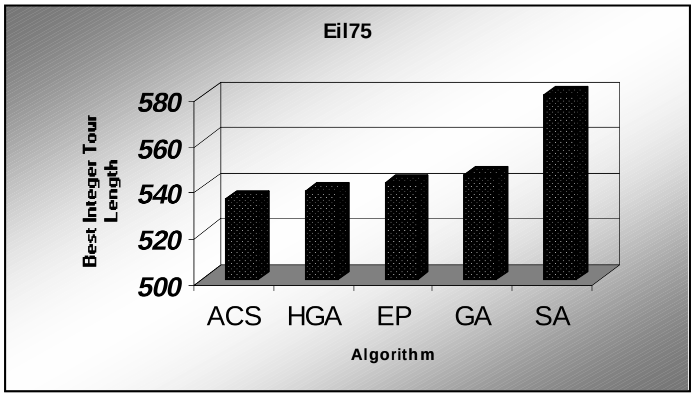


Fig. 12. Best Integer Tour Length obtained by ACS, HGA, EP, GA and SA for Eil75

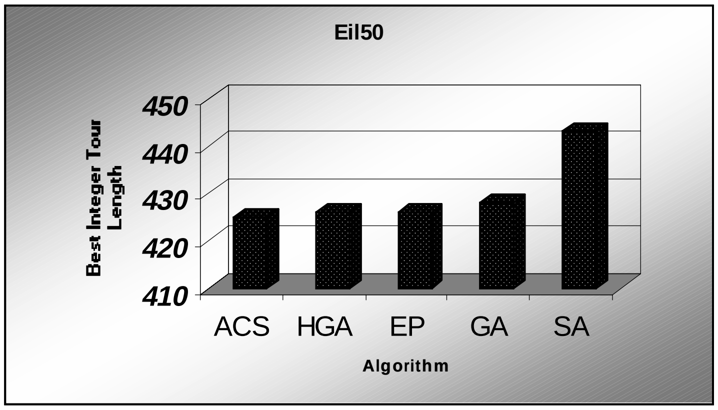


Fig. 13. Best Integer Tour Length obtained by ACS, HGA, EP, GA and SA for Eil50

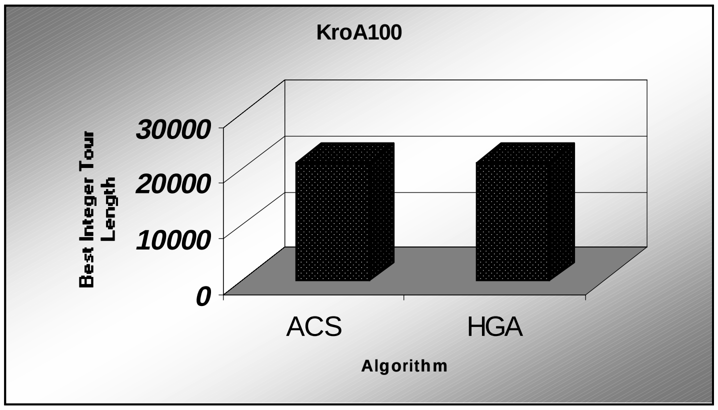


Fig. 14. Best Integer Tour Length obtained by ACS and HGA for KroA100

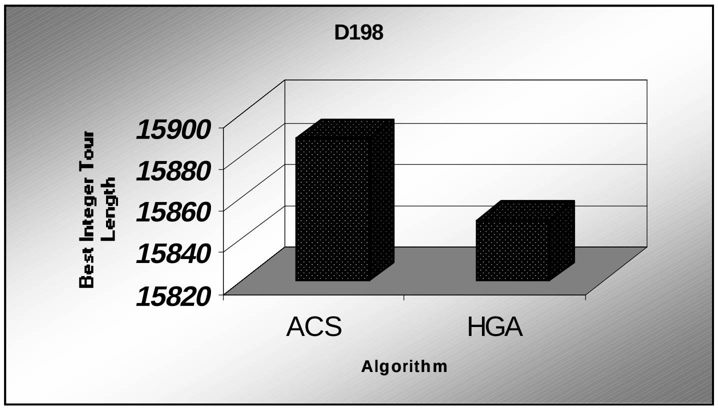


Fig. 15. Best Integer Tour Length obtained by ACS and HGA for d198

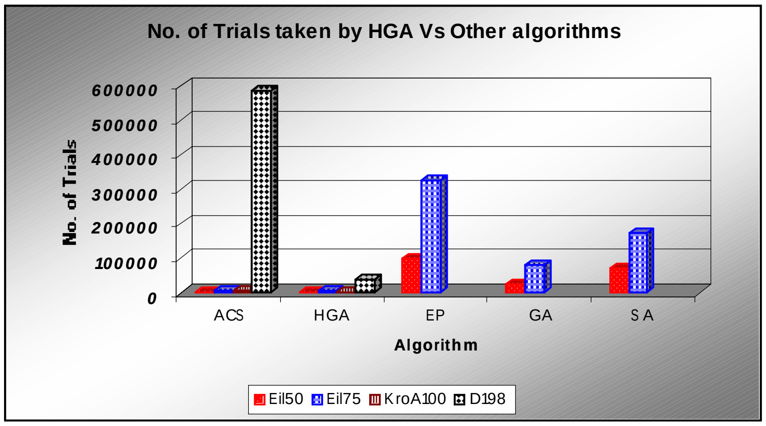


Fig. 16. Number of trials taken by different algorithms

Table 6 reports the results on the geometric instances. The heuristics with which we compare the Hybrid GA in this case are ACS, GA, EP, and SA. The difference between integer and real tour length is that in the first case distances are measured by integer numbers, while in the second case by floating point approximations of real numbers. All the results for ACS, GA, EP and SA are from[1]. In TSPLIB only Eil51 and Eil76 are available which have an additional city to Eil50 and Eil75 respectively.

Table 7. Comparison of results of HGA with Best Known results on geometric instances of the symmetric tsp

Problem Name	HGA	Best Known result	Relative error
Berlin52	7542 (7544.37)	7542 (7544.37)	**0.00 %**
Att48	10,628	10,628	**0.00 %**
Eil51	426 **(428.87)**[*]	426 (429.98)	**0.00 %**
Eil76	538 **(544.37)**[*]	538 (545.39)	**0.00 %**
Eil101	629 **(640.975)**[*]	629 (642.31)	**0.00 %**
KroA100	21,282	21,282	**0.00 %**
KroE100	22,068	22,068	**0.00 %**
Rat99	1211	1211	**0.00 %**
KroC100	20,749	20,749	**0.00%**
KroB100	21,141	21,141	**0.00%**
Bier127	118,282	**118,282**	**0.00%**
KroD100	21,306	21,294	0.07%
D198	15,788	**15,780**	0.05%
kroA200	29,368	**29,368**	**0.00%**

[*] **New Best Results given by Hybrid GA**

We report the best integer tour length and the best real tour length (in parenthesis - if available). In the last column the relative error is given.

Table 7 shows that the new HGA performs comparable to other existing algorithms. In case of Eil51, Eil76 and Eil101 new best real length tours are also obtained. All the 'Best Known' results are from reference 1, reference 2, and reference 3. The Real Tour Lengths are obtained from the sequence of Optimal Results given in TSPLIB[3].

8. Conclusion

We find that the implementation of these three heuristics result in near optimal solutions in most of the cases and improvement in a few cases. These heuristics are simple, straightforward and easy to implement when compared to other algorithms. In investigating the parameters of the algorithm the following conclusions have been reached

- The convergence rate is very fast when the IH is used for initialization when compared to random initialization.
- Size of NEARLIST can be varied from 10 to 20 depending on the distribution of the cities.
- An increase in the parameter q in *LocalOpt* results in a combinatorial increase in time complexity. The effect of it is very small on the convergence rate.

The algorithm compares favorably with previous attempts to apply other heuristic algorithms like Ant Colony System, Genetic Algorithms, Evolutionary Programming, and Simulated Annealing. Nevertheless, competition on the TSP is very tough, and a combination of a constructive method (IH) which generates good starting solutions with local search (which takes these solutions to a local optimum) seems to be the best strategy[1]. We have shown that the IH is a very good constructive heuristic to provide such starting solutions and *RemoveSharp* & *LocalOpt* are very good local optimizers.

References

1. Marco Dorigo and Maria Gambardella - "*Ant Colony System: A Cooperative Learning Approach To Traveling Salesman Problem*" - 1997
2. Darrell Whitley, Timothy Startweather and D'Ann Fuquay - "*Scheduling Problems And Traveling Salesman: The Genetic Edge Recombination Operator*" - 1989
3. TSPLIB: http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPIB95/TSPLIB.html.
4. Zbigniew Michalewicz - "*Genetic Algorithms + Data Structures = Evolution Programs*" – 1993
5. Prasanna Jog, Jung Y. Suh and Dirk Van Gucht – *"Effects Of Population Size, Heuristic Crossover And Local Improvement On A Genetic Algorithm For The Traveling Salesman Problem"* – ICGA'89 – 1989
6. Ellis Horowitz and Sartaj Sahni – *"Fundamentals Of Computer Algorithms"*, Galgotia Publications Pvt. Ltd., New Delhi - 1996
7. Melanie Mitchell – *"An Introduction To Genetic Algorithms"*, Prentice Hall of India Pvt. Ltd., New Delhi - 1998