**INTRODUCTION**

The **travelling salesman problem**(**TSP**) consist on "finding a complete tour of n given cities of minimal length" [1]. This project is about analysing a previously provided Genetic algorithm code to solve a TSP problem and implementing different representation and methods to improvise the solution.

**IMPLEMENTATION OF GENETIC ALGORITHM FOR TSP PROBLEM**

The table <<>> shows a summary of all the methods implemented by the default TSP algorithm and the alternative methods implemented by us. Because of the stochastic nature of genetic algorithm, repeating the same experiments, possibly elsewhere, may lead to different results [1]. A common way around this problem is to count the number of points visited in the search space. This is measured over a number of independent runs, and the **average number of evaluations to a solution** (AES) is used [1].

We implemented a parameter variation function to vary the different parameters of the existing genetic algorithm keeping the rest untouched. The default values of the various parameters are shown in Table2. It plots the average best solution (min distance travelled among all generations) for 10 independent runs as well as its variance(shaded region) vs. each parameter to show the effect of the parameter values on the convergence of the solution. To evaluate the efficiency of the algorithm, we also created a graph to measure the time taken by the algorithm to obtain the best solution vs the parameter value. We considered the number of the generations as a measure of time to plot.

Then we implemented few stopping criteria mentioned in table1. This is needed to avoid useless iterations after the convergence is achieved (based on the stopping criteria used). We decided to implement multiple methods for stopping criteria as we wanted to compare the performance. Then we can decide which one works best for a particular problem.

“When approaching a problem with GA, the representation of the problem has a great impact on the solution and on the algorithm itself” [ http://pubs.ub.ro/scssm/issues/173.pdf]. We implemented the path representation as it is the most natural representation of a tour and is easier to interpret compared to other representations. We have implemented the basic crossover and mutation methods suitable for this representation as specified in Table1. We used the same parameter variation function here, to tune the parameter values and get a combination of values which gives us the best results. We then used the already implemented local heuristic method to test whether it improves our result further.

After fine tuning the parameters, we used our algorithm for 2 of the benchmark problems provided. We changed a modified the no. of individuals and maximum generations values depending on the size of the benchmark problem. We switched off the scaling to compare the best solution obtained by our algorithm with the benchmark solutions.

We implemented a survival selection strategy other than the one already implemented. We used this to compare the effect of survival selection method on the convergence of algorithm.

Table 1

|  |  |  |
| --- | --- | --- |
|  | **Default methods** | **Our implemented methods** |
| **REPRESENTATION** | Adjacency | Path |
| **CROSSOVER** | Alternate edges | Order Crossover |
| **MUTATION** | Inversion | Insertion |
| **PARENT SELECTION** | Ranking with SUS | Fitness proportionate |
| **SURVIVAL SELECTION** | Elitism | Replace Worst (GENITOR) |
| **STOPPING CRITERIA** | The amount of individuals of equal fitness/cost reaches a pre-defined limit | 1. Maximal improvement of the solution over last N generations is lower than limit. 2. Diversity in phenotype space is lower than limit. 3. Efficiency drops below a limit. |
| **LOCAL OPTIMIZATION HEURISTIC** | 2opt method/ local loop detection | - |

**Table2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Range** | |  |
| **Parameters** | **Default Value** | **Min Value** | **Max Value** | **Step size** |
| No. of Individual Solutions | 200 | 10 | 1000 | 30 |
| Maximum no. of generations | 200 | 10 | 1000 | 30 |
| Probability of Crossover | 0.95 | 0 | 1 | 20 |
| Probability of Mutation | 0.05 | 0 | 1 | 20 |
| Elitism percentage | 0.05 | 0 | 1 | 20 |

**Results**

**Parameter Variation:**

1. **No. of individuals**: In Fig1.a, we observe that with the increase in the no. of individuals, we get better results as the best solution (min distance) across all generations decrease. For adjacency, we see that if the parameter is increased more than 538, there is not much improvement in the results. However, path representation converges to an even better solution with the same parameter settings.
2. **Maximum number of generations**: In Fig1.b, we see that with increase in the maximum no. of generations for which the algorithm is allowed to run, we get better solutions across runs. However, the cost of achieving the solution increases as we increase the no. of generations. Hence there is a trade-off between the optimal solution we need and the no. of generations we can allow our algorithm to run to give us the results. However, in path representation, we see that after a certain point (500-600), there is not much improvement in the solution with respect to the increase in the cost (in terms of no. of generations).
3. **Probability of Crossover**: In Fig1.c, we can see that the best solution converges till a certain value of the crossover probability and after that the results don’t improve further. In adjacency, the best result is achieved with a probability of 60% whereas for path representation, as we increase the probability to 80-90% it gives better result. Also, with higher value of crossover probability, the no. of generations needed to converge to a better solution is less. For adjacency, the algorithm converges with higher probability of crossover, but it doesn’t give a good result.
4. **Probability of Mutation**: In Fig1.d, we observe that the best solution is achieved with a lower value of mutation probability and the results get worsen as we increase the mutation probability. For adjacency, the best solution was found around 5% mutation while for path, we could get better results up to 10% mutation rate. Also, there is not much variation in the time taken for convergence to best solution.
5. **Elitism Percentage**: As shown in Fig1.e, with the increase in elitism percentage, the best solutions tend to improve in the beginning. However, after a certain threshold (0.2 for adjacency) the solutions don’t improve further. Also, the performance is better for path representation. The average no. of generations to converge doesn’t variate much after increasing to a certain value at the beginning.

|  |
| --- |
|  |
| (a) |
| (b) |
| (c) |
| (d) |
| (e) |

Fig1. Performance of parameter variation for Adjacency and Path Representation

**Stopping Criteria:**

**Case1**: If the maximal improvement of the solution over last N generations is below a certain predefined limit, then the iteration stops. This predefined value can be 20% to 50% of the generation number which the best solution has found so far. i.e. the algorithm reaches to a certain value at generation 50, then this value doesn't change for 25 generations (50% of 50), so the algorithm stops. We fine-tuned the stopping criteria such that, the maximum generation limit is 50% and the minimum improvement over last N generations is at least 5%. This is a trade-off between how much better solution we desire with how many generations we allow the algorithm to run.

**Case2**: If Diversity in the phenotype space is lower than a particular limit, the algorithm stops. The limit is set by manual experimentation by observing the variation in the fitness values which leads to the diversity in the population phenotype. We set the minimum diversity limit to 10% after fine tuning.

**Case3**: If Efficiency drops below a certain limit, the algorithm stops running. We obtained the limit value 0.0625 by fine tuning. We define fitness as 100/cost (distance travelled), and efficiency is fitness/generations. We observe that with increasing no. of generations, the value of fitness increases to a certain level, but the efficiency actually reduces as the improvement comes with a cost of increased no. of generations. However, set the minimum efficiency limit to 0.0625 by experimentation which gives us a good convergence point. Fig2. shows the comparison of all these stopping criteria along with the default one.

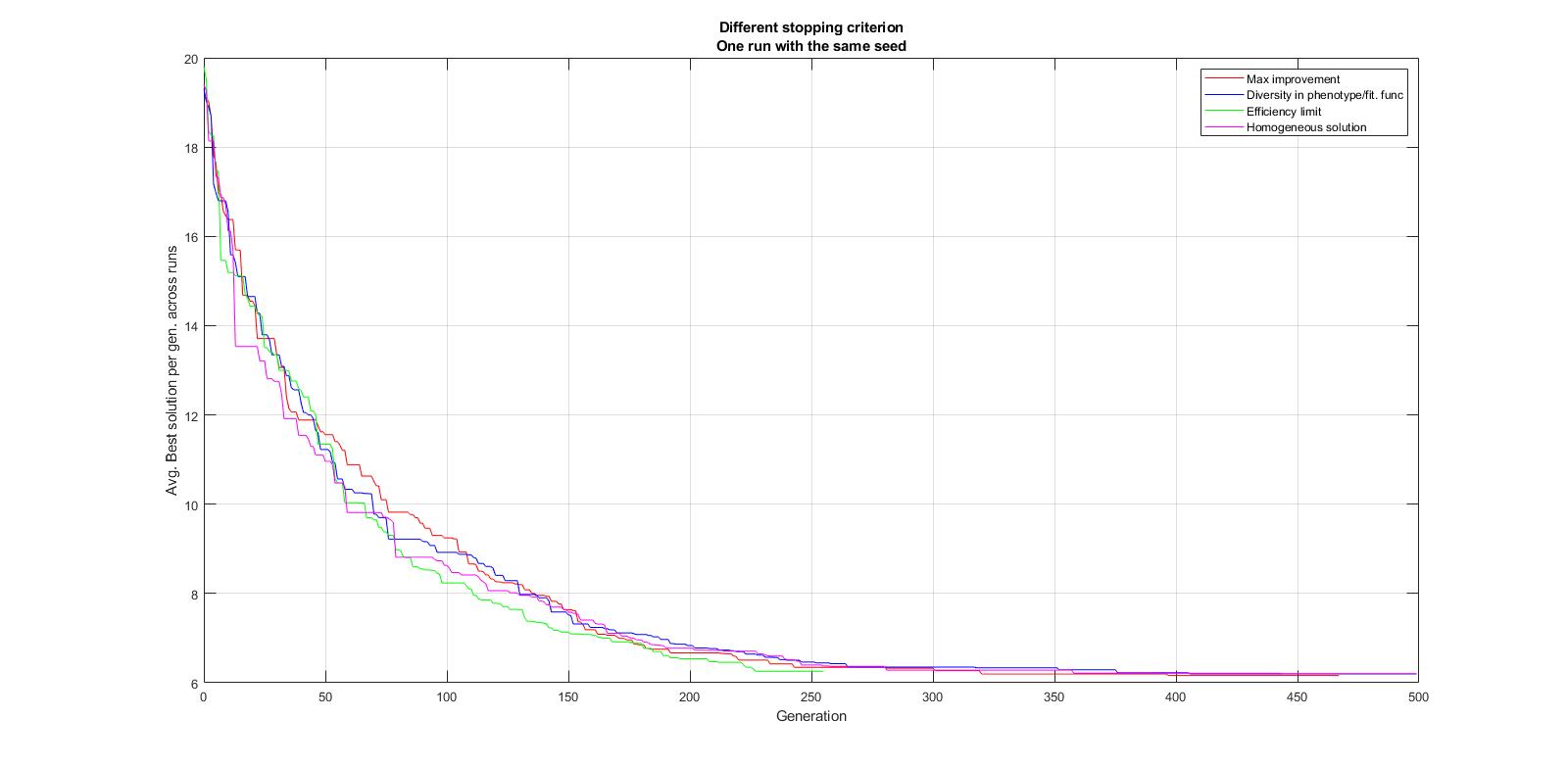


Fig2. Different stopping criterion run with same seed value for comparison.

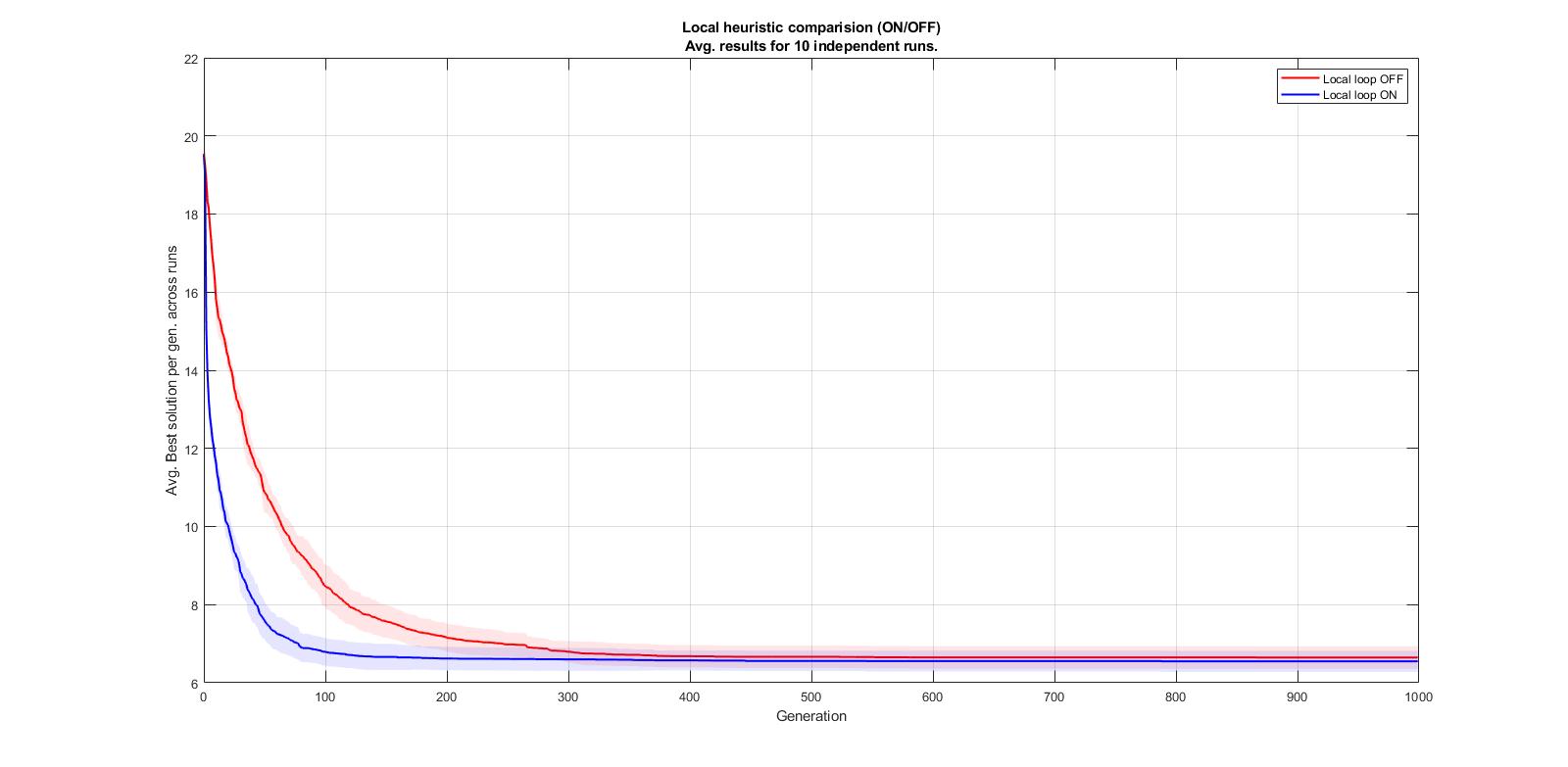
**Parameter Tuning for Path representation:**

We obtained the combination of optimal parameter values from the graph in Fig1 and fine tuned it further to get better results (minimum distance travelled). Table3 shows the optimal parameter values used for further analysis.

We used the optimal values to run our algorithm and we compared the results for 2 problems (16 cities and 51 cities). We observed that for a bigger problem, since the search space is higher, we need to increase the no. of individuals and max gen to higher values to obtain better results. So, we decided to use different combination of parameters for separate problems.

**Impact of Local Optimal Heuristic:**

We used the optimal parameter values to run the algorithm to obtain the best solutions. When local heuristic function was switched ON, the same best solution was found to be attained at a very early rate. The local heuristic converged to best solutions around 300 generations earlier, hence saving time. The graph in Fig3, shows the comparison of the performance with local heuristic switched ON and OFF.

****

**Fig 3. Local Heuristic Comparison (ON/OFF). Avg. results for 10 independent runs.**

**Benchmark Problems:**

We selected 2 of the benchmark problems to test our algorithm.

1. **131 Cities:**

Fig4a plots the performance of our algorithm (Path representation with order crossover) for the benchmark problem of 131 cities. The best value obtained by our algorithm is 728. The optimum length provided by the benchmark solution is 564. The relative error percentage is 29%. The parameter values used to obtain this result is shown in Table3.

|  |  |
| --- | --- |
| Parameter | Value |
| No. of Individuals | 700 |
| Maximum no. of generations | 1000 |
| Probability of crossover | 0.95 |
| Probability of Mutation | 0.1 |
| Elitism Percentage | 0.15 |
| Local Loop Heuristic | ON |
| Mutation Type | Insertion |

Table3. Optimal Parameter values for 131 cities Benchmark Problem.

1. **380 Cities:**

Fig4b plots the performance of our algorithm for the benchmark problem of 380 cities. The best value obtained by our algorithm is 2161. The optimum length provided by the benchmark solution is 1621. The relative error percentage is 33.3%.The parameter values used to obtain this result is shown in Table4.

|  |  |
| --- | --- |
| Parameter | Value |
| No. of Individuals | 1500 |
| Maximum no. of generations | 2000 |
| Probability of crossover | 0.97 |
| Probability of Mutation | 0.75 |
| Elitism Percentage | 0.15 |
| Local Loop Heuristic | ON |
| Mutation Type | Inversion |

Table4. Optimal Parameter values for 380 cities Benchmark Problem.

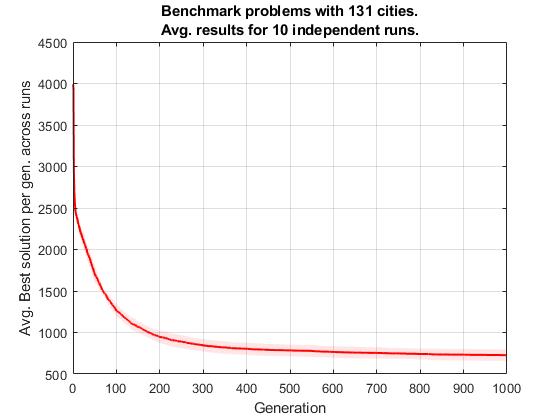


Fig4a. Benchmark Problem with 131 cities. Avg. results for 5 independent runs.

Fig4b. Benchmark Problem with 380 cities. Avg. results for 5 independent runs.

**Survival Selection Strategy:**

We implemented another survival selection strategy different from the already implemented Elitism. This is the Replace Worst (GENITOR) algorithm. In this scheme the worst λ members of the population are selected for replacement. It leads to early convergence and might not produce great results, as it focusses on the fittest individuals and hence doesn’t allow much diversity in the population. We plotted a graph between the Elitism and Replace worst selection strategy in Fig5. We can see that, for Replace worst algorithm, the curve is steep at the beginning showing faster convergence (at gen 14). But later the graph becomes shallower and do not get a good result. The elitism strategy converges to a better solution (at gen 100).

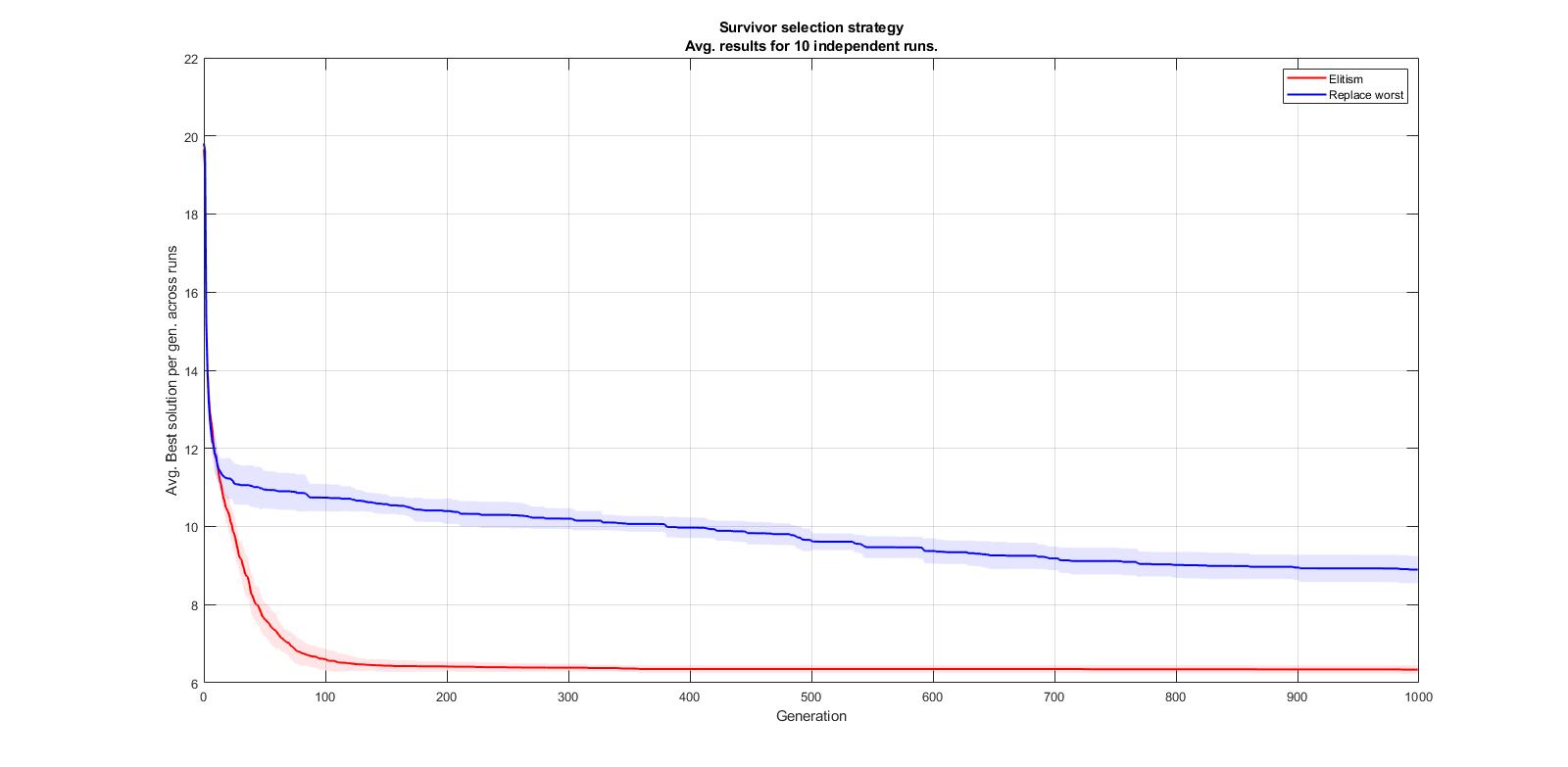


Fig5. Replace Worst vs Elitism Survival Selection strategy.   
Avg. results for 10 independent runs.

**Discussion**

For adjacency representation, we observed that as we increase the no. of individuals, we tend to get better results. For problems with larger search spaces (more no. of cities), we need to have more no. of individuals. This is required for exploration of the entire search space. With increase in the maximum no. of generations, the results keep improving at the cost of time of computation. Hence there has to be a trade off between the acceptable results and the time taken (in terms of no. of generations). Elitism for adjacency representation up to 20% gives improvement in the solution. If we set the elitism to 0%, there is a high chance of the loss of fittest individual from the generation due to crossover and mutation. However, if we increase the percentage beyond limit, the result becomes worse. This is because we have less individuals to undergo crossover and mutation and the entire search space is not used to look for a better solution. When more individuals undergo crossover, there is increase in diversity and it is more likely to find an optimal solution. But for adjacency, the representation denotes the cities adjacent to each other. So, higher crossover probability introduces more randomness in the solution and hence the result does not improve after a certain limit. And it leads to local loop problem. This problem is usually avoided by jumping to a random value outside the loop. However, this loop error is not addressed in the default program. This issue doesn’t occur for Path representation. Hence, we can see that the crossover can be set to higher value for Path and it yields better result. Probability of mutation introduces diversity in the solution. We see that with a low mutation rate, we already get a better solution. Hence by increasing the rate further, there is more diversity introduced and the best solution is lost in the process. So, keeping the mutation rate to a lower limit is better.

If we wish the algorithm to stop at some stage, we must also provide a termination condition [1]. This helps us to avoid useless iterations after a certain convergence is achieved. We observed from the results that after 200-250 generations, there is

**APPENDIX A**

**APPENDIX B**

**APPENDIX C**

**APPENDIX D**

**APPENDIX E**

**APPENDIX F**

# Bibliografía

|  |  |
| --- | --- |
| [1] | J. S. A.E. Eiben, Introduction to Evolutionary Computing, Springer, 2003. |