**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. On the tableN, we present the list of default methods, plus the new implemented methods used to modify the genetic algorithm for getting a better 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 as default. 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 vs. each parameter to show the effect of the parameter values on the convergence of the solution.

Then we implemented few stopping criteria mentioned in table1 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 in order to 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 to get a combination of values which gives us the best results.

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**

|  |  |
| --- | --- |
| **Parameters** | **Default Value** |
| No. of Individual Solutions | 200 |
| Maximum no. of generations | 200 |
| Probability of Crossover | 0.95 |
| Probability of Mutation | 0.05 |
| Elitism percentage | 0.05 |

**EXPERIMENTS PERFORMED ON EXISTING PROGRAM**

We implemented a function that automatically varies the parameters of the existing genetic algorithm. We take as input a range of values for each parameter, listed on TableN, and get as output a graph of average best solutions across runs with a shade of standard deviation (keeping the max. number of generations constant) and average number of generations to get the best solution vs. each parameter value.

**average number of evaluations to a solution** (AES) – for N runs

We varied one parameter at the time, by setting all the rest as a constant (default value in tableN), we also disabeled stopping criterions, so we can analyse all experiments under the same conditions.

Table

Parameter – default value – range of variation

In order to measure the efficiency of the algorithm, we also created a graph of the time that it took to the algorithm to reach the minim traveling distance vs the parameter value. We consider plotting the number of the best generation instead of the time, but as we are changing parameters the execution time for each generation might change as in we change the parameters (for instance, for more number of individuals, the execution time for single generation is longer). So, we decided to use time in order to have a standard measure among parameters.

1. Variation of the number of Individuals.

Graph

1. Optimal number of generations.
2. Optimal values for other parameters (Crossover, Mutation, Elite percentage).

The above optimal results were tested on the different number of cities. This optimal setting is used to proceed with the rest of the project henceforth.

**IMPLEMENTED ALGORITHM**

1. Stopping Criteria: Implementation of Stopping criteria to the existing algorithm to avoid useless iterations. Below, the stopping criterions that were implemented (Code in Appendix A):

**Case1**: If the maximal improvement of the solution over last N generations is lower than a predefined limit. The algorithm proceeds until the best solution during the evolution process doesn't change to a better value for a predefined value of generations. This predefined value can be 20% or 30% of the generation number which the best solution has found so far. i.e. the algorithm reaches to a value of 200 at generation 50, then this value doesn't change for 15 generations (30% of 50), so the algorithm stops.

**Case2**: If Diversity in the phenotype space is lower than a particular limit. The limit is set by manual experimentation by observing the variation in the fitness values which leads to the diversity in the population phenotype.

**Case3**: If Efficiency drops below a certain limit which is also tuned as per experimentation.

1. Representation: We implemented the Path representation as it is easier to interpret. This representation shows the path of the cities travelled starting from an initial position to the next city and ends at the city which is in turn connected to the initial city to complete the tour.   
   (Code in Appendix B)
2. Crossover. (Code in Appendix C)
3. Mutation. (Code in Appendix D)
4. Parent Selection. (Code in Appendix E)
5. Survival Selection. (Code in Appendix F)
6. Testing optimal combination of parameters on benchmark problems.

**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. |