Genetic Algorithms and Evolutionary Computing

Final Project

2018-2019

Authors:

Maria Camila Alvarez Trivino (r0731521)

Swarnalata Patra (r0729319)

Professor:

Prof Dirk Roose

Date:

January 7th 2019

INTRODUCTION

The **travelling salesman problem** (**TSP**) is based on "finding a complete tour of N given cities of minimal length" [1]. The current project consist on analysing and improving a previously provided Genetic algorithm code in MatLab to solve a TSP problem. This implementation used adjacency representation for the candidate solution to the problem. Moreover, it utilizes alternate edges crossover, inversion mutation, elitism as survival selection strategy and Ranking with 'SUS' (stochastic universal sampling) as parent selection strategy. We implemented an alternative representation (path representation) with its corresponding crossover and mutation methods in order to improve the results. In this report, we will present, compare and discuss about the results we got from the previous and the new implementations. Overall, the path representation implementation provided better solutions than the adjacency representation one, it means it gave complete tours with the minimal length.

IMPLEMENTATION OF GENETIC ALGORITHM FOR TSP PROBLEM

The **Table 1** 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 several 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, we varied one parameter at the time while keeping the rest untouched. The predefined values of the various parameters are shown in the **Table 2**. For the variation of each parameter, we generate a graph that 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 value. The graphs we will present correspond to the TSP problem of 51 cities. We plotted this graph with the aim of showing the effect of the change in parameter values on the convergence of the solution. Moreover, in order to evaluate the efficiency of the algorithm, we also created a graph to measure the average number of generations taken by the algorithm to obtain the best solution vs. the parameter value.

After, we implemented the stopping criteria methods mentioned in the **Table 1**. These methods are 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 their performance. In that way, we can decide which one works best for a particular problem. As for each run the random initialization generates different results, we set a specific seed while testing the stopping criteria methods, so that we can have the same pseudo-random initialization and consequently same results, for a better comparison. In this way, we can compare under the same possible results' behaviour, which stopping criteria finishes the algorithm at the better moment.

As "the representation of the problem has a great impact on the solution and on the algorithm itself" [2], we implemented the path representation, because it is the most natural representation of a tour and is easier to interpret compared to other representations. We used the basic crossover and mutation methods suitable for this representation as specified in **Table 1.** In order to tune the parameter values and get an optimal combination which gives us the best results for this representation, we used the same parameter variation function previously described. Furthermore, we also used the already implemented local heuristic method, and tested whether it improves our result further.

After fine tuning the parameters, we used our algorithm for testing two of the benchmark problems provided: 131 and 380 cities. We modified the number of individuals and maximum generation values depending on the size of the benchmark problem. With the purpose of being able to compare the results from our algorithm with the benchmark's optimal solutions, we switched off the scaling of the x and y coordinates of the cities. Finally, we implemented a survival selection strategy other than the one already implemented (see **Table 1**). We used this method to compare the effect of survival selection method on the convergence of algorithm.

Table 1. Summary of implemented methods

	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 individual solutions 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	-

Table 2. Selected default values for the parameter variation

		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

<u>Parameters' Variation</u> (51 cities problem)

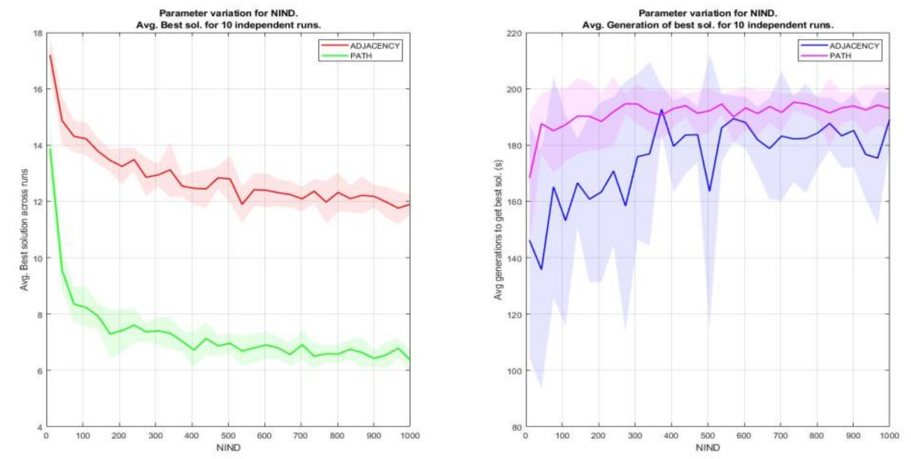
No. of individuals: In **Figure 1.a**, we observe that with the increase in the no. of individuals, the average best solution for the path representation curve (green line), which corresponds to the minimum distance travelled, decreases monotonically. While, for adjacency (red line), we see that if the parameter is increased more than 538, there is not much improvement in the results. Thus, path representation converges to an even better solution with the same parameter settings. Regarding the generation to get the best solution, for the path representation we can see that its value does not vary a lot independent of the number of individuals, remaining around 180-200 generations (magenta line). In general, taking more generations to converge than adjacency, but with a low standard deviation. However, the one for adjacency representation (blue line), shows an increasing behaviour with a huge standard deviation.

Maximum number of generations: In **Figure 1.b**, we see that with increase in the maximum no. of generations (for which the algorithm is allowed to run), both the path and the adjacency curves for the minimum travelled distance (green and red lines) present a decreasing behaviour. Which means that we get better solutions across runs. However, the cost of achieving the solution (in terms of time) increases as we increase the no. of generations (magenta and blue lines). Hence there is a trade-off between the optimal solution that we need, and the no. of generations we can allow our algorithm to run to give us the desired results. However, in path representation (green line), we see that after a certain point (500-600 gen), there is not much improvement in the solution with respect to the increase in the cost, in terms of no. of generations (magenta line).

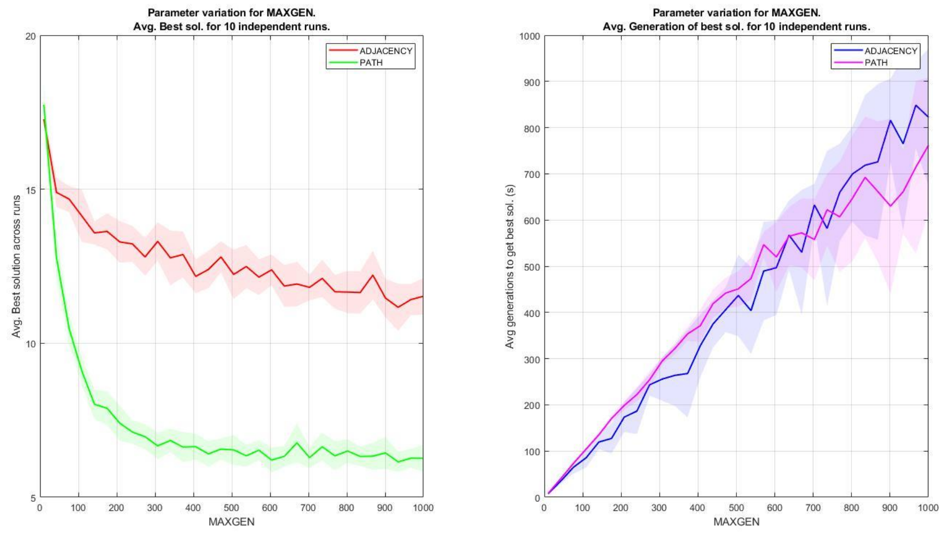
Probability of Crossover: In **Figure 1.c**, we can see that for the path curve (green line), the best solution improves till a certain value of the crossover probability and after that it does not improve any further. In adjacency (red line), the best result is achieved with a probability of 60%, whereas for path representation, as we increase the probability to around 80-90% it gives better result. Additionally, for path (magenta line) with higher value of crossover probability, the no. of generations needed to converge to a better solution is less. Also, for adjacency (blue line), the no. of generations needed to converge reduces (but with a huge variability) with increased crossover probability. However, for higher values but it doesn't give a good solution result.

Probability of Mutation: In **Figure 1.d**, we observe that for path and adjacency (green and red lines), the best solution is achieved with a lower value of mutation probability and the results deteriorate with increasing rate. For adjacency (red line), the best solution was found at around 5% mutation rate while for path, we could get better results up to 10% mutation rate. Also, there is not much variation in the time taken (no. of generations) for getting the best solution (magenta and blue lines).

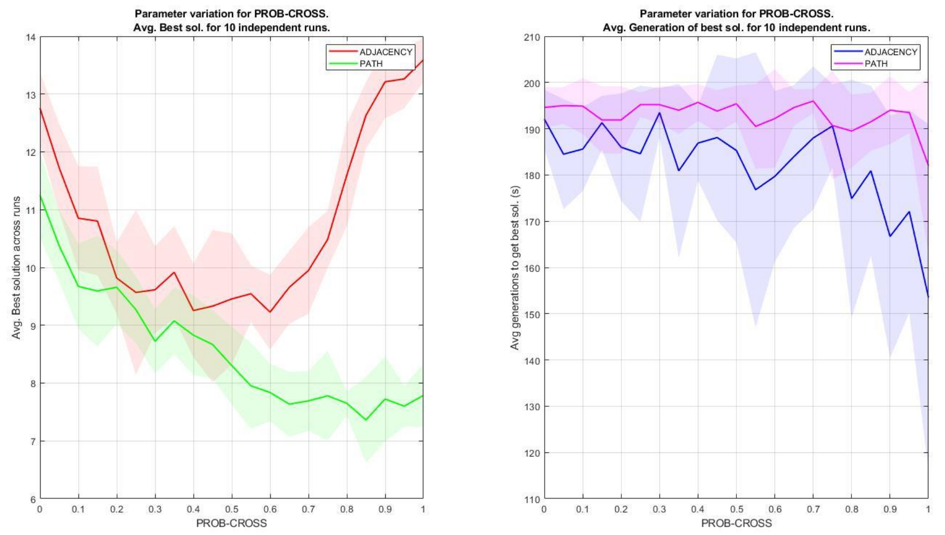
Elitism Percentage: As shown in **Figure 1.e**, with the increase in elitism percentage, the best solutions for both path and adjacency (red and green) tend to improve (decreasing) in the beginning. However, after a certain threshold (0.2 for adjacency) the solutions don't improve further. Also, the performance is better for path representation. The average no. of generations to get the best solution doesn't variate (magenta and blue) much after increasing to a certain value at the beginning.


Parameter variation for NIND.
Avg. Best sol. for 10 independent runs.
ADJACENCY
PATH
Avg. Best solution across runs
NIND
Parameter variation for NIND.
Avg. Generation of best sol. for 10 independent runs.
ADJACENCY
PATH
Avg generations to get best sol. (s)
NIND

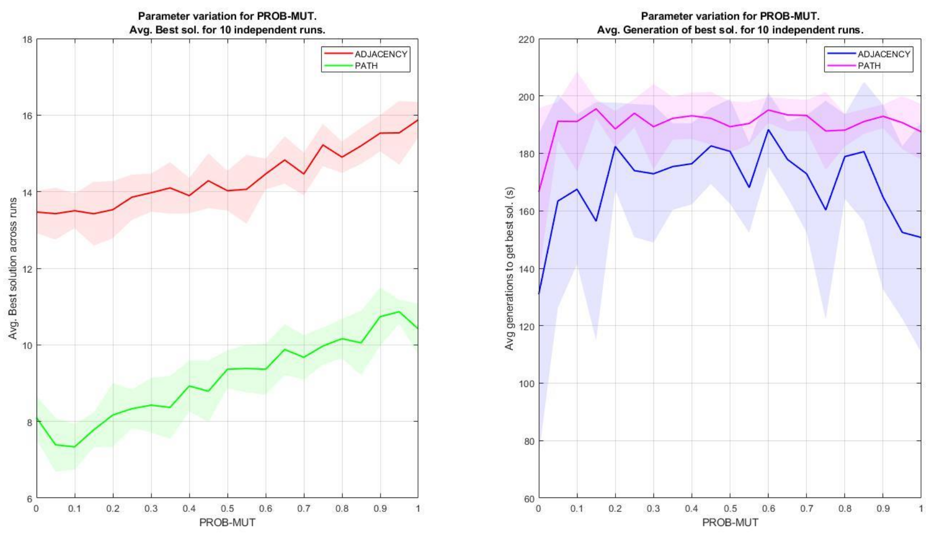
(a)


Parameter variation for MAXGEN.
Avg. Best sol. for 10 independent runs.
ADJACENCY
PATH
Avg. Best solution across runs
MAXGEN
Parameter variation for MAXGEN.
Avg. Generation of best sol. for 10 independent runs.
ADJACENCY
PATH
Avg generations to get best sol. (s)
MAXGEN

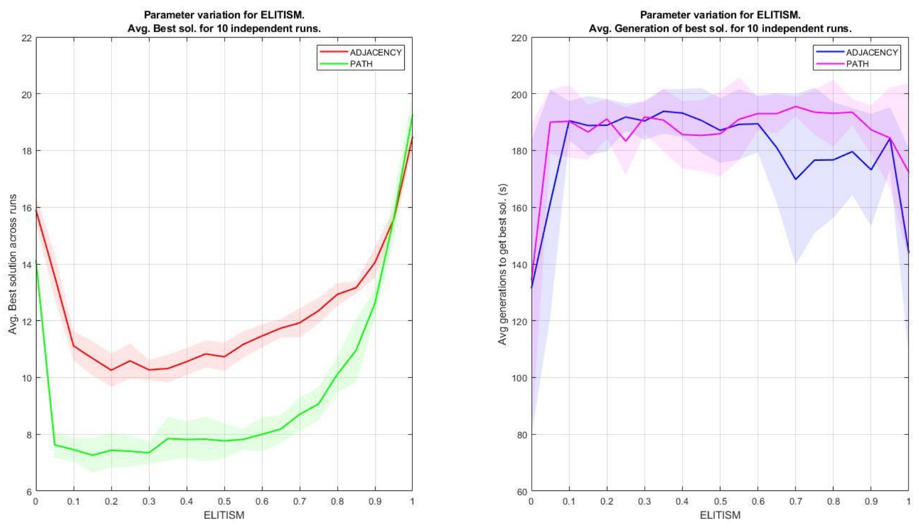
(b)


Parameter variation for PROB-CROSS.
Avg. Best sol. for 10 independent runs.
ADJACENCY
PATH
Avg. Best solution across runs
PROB-CROSS
Parameter variation for PROB-CROSS.
Avg. Generation of best sol. for 10 independent runs.
ADJACENCY
PATH
Avg generations to get best sol. (s)
PROB-CROSS

(c)



(d)



(e)

Figure 1. Performance of parameter variation for Adjacency and Path Representation. 51 cities problem

<u>Stopping Criteria</u>

Case1: If the maximal improvement of the solution over last N generations is below a certain predefined limit, then the iteration stops. The last N generations can be predefined as a percentage (20% to 50%) of the current generation number 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 required over last N generations is at least 5%. This is a trade-off between how much better solutions we desire with how many generations we allow the algorithm to run.

Case2: If the 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 0.1 after fine tuning.

Case3: If the 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 (fitness is inversely proportional to distance travelled), and efficiency is fitness/generations (fitness achieved at the certain number of generations). We observe that with increasing no. of generations, the value of fitness increases to a certain level, but the efficiency reduces as the improvement comes with a cost of increased no. of generations. However, we set the minimum efficiency limit to 0.0625 by experimentation which gives us a good convergence point.

The **Figure 2** shows the behaviour and the comparison between all these stopping criteria along with the default one previously implemented (homogenous solution). This results were obtained by setting the same seed when running each method.

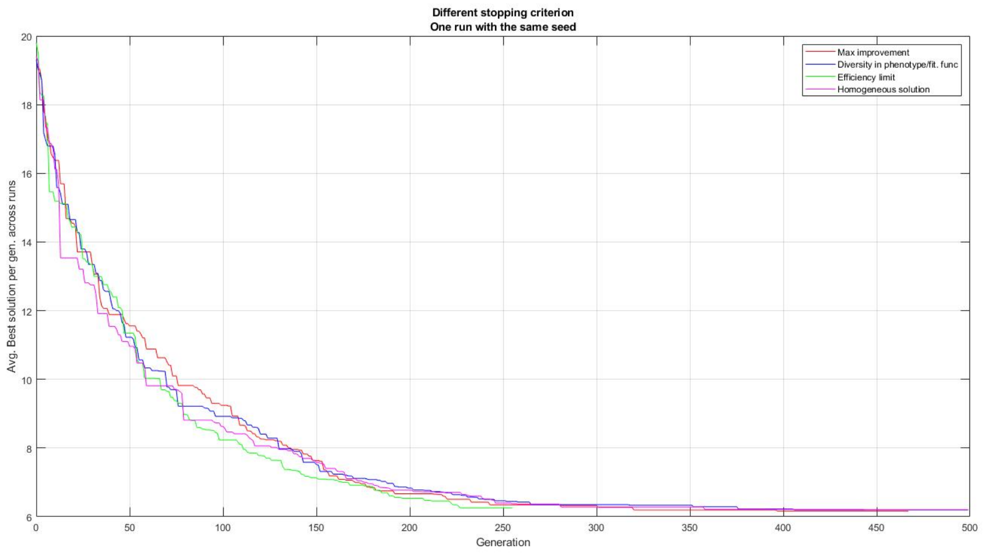


Figure 2. Different stopping criterions run with same seed value for comparison. 51 cities – path representation.

<u>Parameter Tuning for Path representation</u>

We obtained the combination of optimal parameter values from the path curves (green lines) of the graphs in **Figure 1**, and fine-tuned it further to get better results (minimum distance travelled). The **Table 3** shows the values used for the 51 cities analysis. While the **Table 4** 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 maximum no. of generations to higher values for better results. So, we decided to use experiment with different combination of parameters for separate problems.

Parameter	Value
No. of Individuals	200
Maximum no. of generations	200
Probability of crossover	0.95
Probability of Mutation	0.05
Elitism Percentage	0.15
Mutation Type	Insertion

Table 3. Parameter values for 51 cities

Impact of Local Optimal Heuristic

We used the optimal parameter values from **Table 4** and the path representation to run the algorithm to get the best solution. When local heuristic function was switched ON, the same best solution was found to be attained much earlier. The local heuristic converged to best solutions around 300 generations earlier than when it was turned OFF, hence saving time (more efficient). The graph in **Figure 3**, shows the comparison of the performance with local heuristic switched ON (blue line) and OFF (red line).

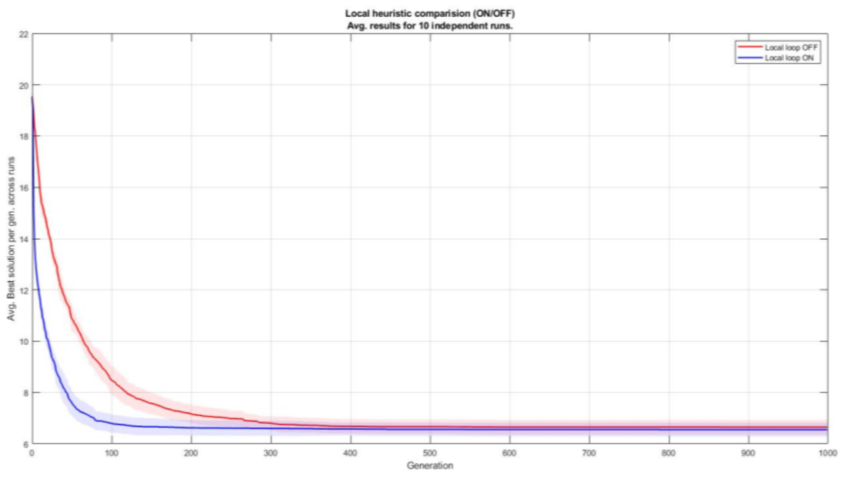


Figure 3. Local Heuristic Comparison (ON/OFF).

Avg. results for 10 independent runs. 51 cities – path representation.

Benchmark Problems

We selected two of the benchmark problems to test our algorithm with path representation.

(a) **131 Cities: Figure 4** plots the performance of our algorithm (Path representation) 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 are shown in **Table 4.**

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
Crossover Type	Order crossover

Table 4. Optimal Parameter values for 131 cities Benchmark Problem.

(b) **380 Cities: Figure 5** 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 are shown in **Table** 5.

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
Crossover Type	Order crossover

Table 5. Optimal Parameter values for 380 cities Benchmark Problem.

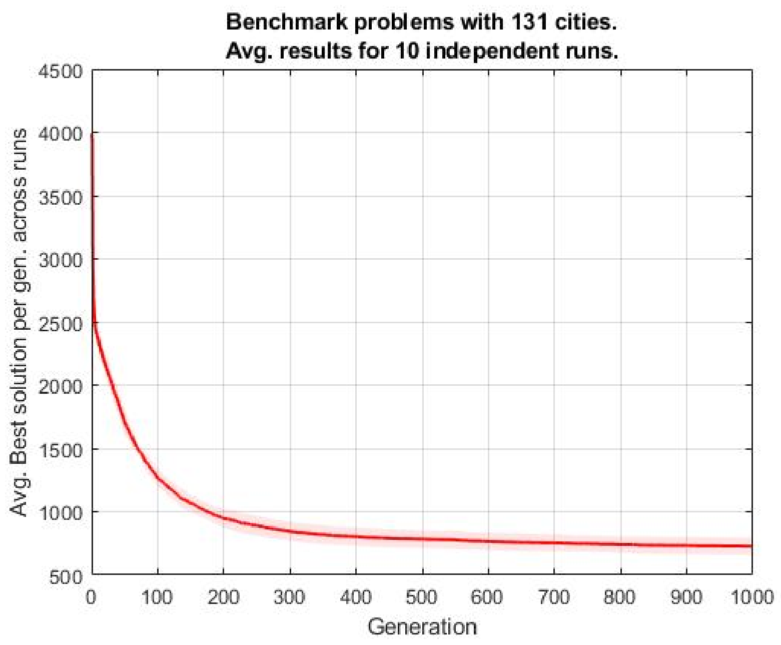


Figure 4. Benchmark Problem with 131 cities. Avg. results for 5 independent runs.

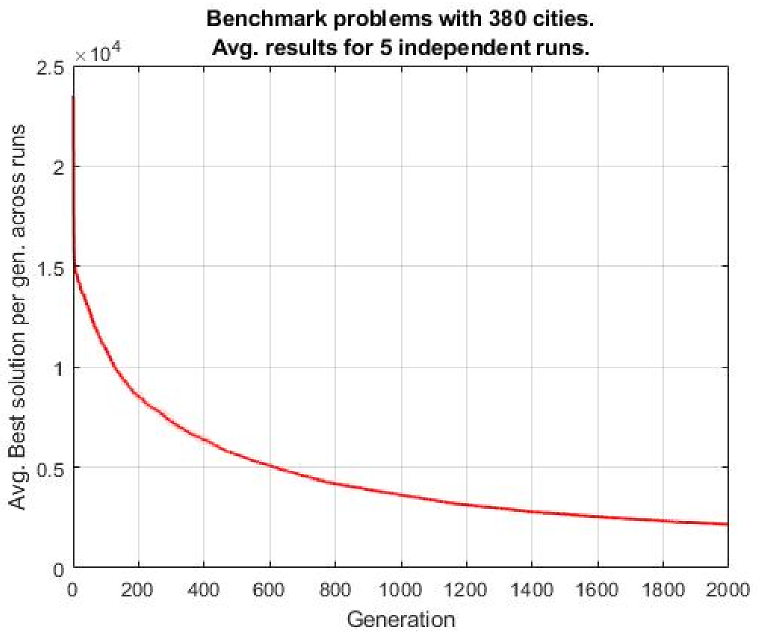


Figure 5. Benchmark Problem with 380 cities. Avg. results for 5 independent runs.

<u>**Survival Selection Strategy**</u>

We implemented an alternative 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. We plotted a comparison graph between the Elitism (red line) and Replace worst (blue line) selection strategy in **Figure 6**. We can see that, for Replace worst algorithm, the curve is steep at the beginning showing faster convergence to a worst value (at gen 14). Later, the curve becomes shallower and does not show significant improvement. The elitism strategy converges to a significantly better solution (at gen 100).

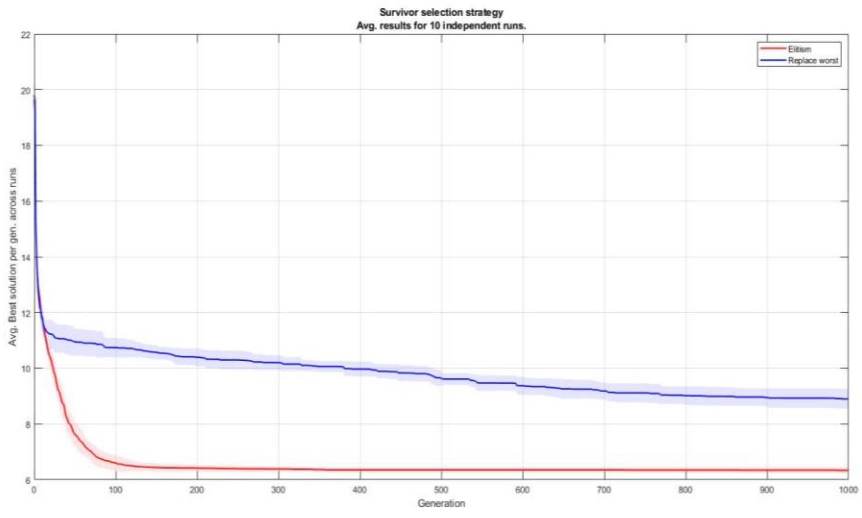


Figure 6. Replace Worst vs Elitism Survival Selection strategy.
Avg. results for 10 independent runs.

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. Moreover, 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). The percentage of Elitism for adjacency representation up to 20% generates and improvement in the solution. If we set the elitism to 0%, there is a high probability of losing the fittest individuals from the generation due to crossover and mutation, which is not good for finding the optimal solution. However, if we increase the percentage beyond certain 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.

Regarding the probability of crossover, we found that when more individuals undergo crossover for the path representation, there is increase in diversity and it is more likely to find an optimal solution. Also, the no. of generations for finding the best solution reduces. But for adjacency, as this representation denotes the cities adjacent to each other. When there is a higher probability of crossover, more randomness is induced in the solution. Therefore, the result does not improve after a certain limit. Moreover, for adjacency we observe that no. of generations reduces with an increase in crossover probability percentage, however the variance increases to a huge extent. This shows that the algorithm doesn't converge to any optimal solution. Rather, it introduces local loop puzzle. 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. Fortunately, this issue doesn't occur for Path representation. Thus, we can see that the crossover can be set to higher value for Path and it yields better result. On the other hand, regarding the probability of mutation, it introduces diversity in the solution. So, we see that with a low mutation rate, we already get a better solution. If it is increased too much, 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 want 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 level of convergence is achieved. We observed from the results (see **Figure 2**) that after 250 generations, we get the best solution. So, we want to avoid the unnecessary iterations. The efficiency limit criterion, stops the algorithm once the efficiency goes below 0.0625 (limit defined by experimentation) level. There is not much improvement in the solution after the efficiency limit goes below this minimum defined limit. With increasing no. of generations, the value of fitness increases to a certain level, but the efficiency reduces as the improvement comes with a cost of increased no. of generations, which is good according with the previous analysis. **Figure 2** shows that with this method, the algorithm stops after 255 generations. For the maximal improvement stopping criteria, we have tuned the minimum improvement to be 5%. So, if 50% of the generations doesn't have an improvement of at least 5%, it should stop. In the **Figure 2** we can see that with this criterion, the algorithm stops after 467 generations, it allowed more usless iterations that the previous method. Finally, for the diversity criteria, we are considering the variance of the fitness in the phenotype. The diversity limit is set to 0.1 by experimentation. This tells us that if the variance of the individuals' fitness in the possible solutions is lower than the limit, we stop. In the experiment of the **Figure 2**, this stopping criterion was not activated, which means the diversity was not under the limit.

We chose Path representation for an easier interpretability of the obtained solution. We plotted the parameter variation of path across adjacency representation to see the overall performance improvement. It helped us to obtain the optimal combination of parameter values for getting a better result (minimum travelled distance is around 7). For crossover, with probability 0.85 we obtain a better solution as shown in the graph **Figure 1.c**, but the variance is high. However, with probability 0.95, the variance is less for 10 independent runs and it gives similar results. With low variance the solution is more deterministic. The variance is monotonic throughout but more deterministic at 0.95 with good results. Hence, we decided to choose 0.95 as our optimal value. For mutation and elitism, the variance is not much informative as it is monotonic throughout. Thus, we decided to take the values for which we obtained the best solution (minimum distance covered) across 10 independent runs as can been seen in the graphs. **Table 3** shows the combination for 51 cities problem.

When the local optimization heuristic was switched OFF, the best solution is reached after 300 generations. However, when it was switched ON, we converged to similar best solution (the minimum distance travelled is around 7) at around100th generation which is a great improvement. So, by switching ON the local optimal heuristic function with the optimal combination of values, we converge to the best solution much faster.

Therefore, for the benchmark problem, we decided to keep the local heuristic switched ON, along with the other optimal parameter values. However, we increased the no. of individuals and generations, suitable for the size of problem (obtained through experimentation). We obtained an error of around 29% for 131 cities problem and 33% for 380 cities problem. So, we conclude from here that for a problem with higher search space, we need to scale the parameters accordingly to obtain better results. The crossover, mutation and survival selection strategies implemented also have an additional influence on the performance of the algorithm. For instance, we observed that the inversion mutation usually performs better than the insertion mutation.

Finally, the survival selection strategy provided by default is Elitism. It has a better performance compared to the Replace Worst method implemented by us. Elitism ensures that the fittest individual(s) (depending on the percentage) are always retained for the next generation, instead of lost. Therefore, this strategy converges to a better solution compared to the other one. Replace worst method, on the other hand, leads to too early convergence and might not produce great results, as it focusses on the fittest individuals and consequently doesn't allow much diversity in the population.

CONCLUSION

We have presented the experimental results of how the values of the parameters of a genetic algorithm influence its performance, in the context of the TSP problem. We conclude that it is very important to perform the suitable fine tuning for each the specific problem in order to get an optimal solution. Fortunately, we learnt that there are some heuristics to follow in order to modify those parameters and get a good result. Like for instance, that the probability of crossover is usually better to be high, but the one for mutation low; or that usually between more generations better the solution, and that the number of individuals is related with the size of the problem.

REFERENCES

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

[2] L. Ursache, «Representation models for solving TSP with genetic algorithms,» de *CNMI*, Bacău, 2007.

APPENDIX

Parameter Variation:

```
%Camila and Swarna method
% Setting all parameters as a constant and varying one parameter at the time,
% disabling stopping criterions, so we can analyse all experiments under the same conditions
%%%Parameter variation plotting for analysis:
%%General parameters for adjacency (Question 2), local heuristic LOCALLOOP ON/OFF:
%%(Question 5), and  survivor selection strategy: REPLACE_WORST (Question 7)
   %%inside parameter_variation method, select (uncomment) which set of
   %%parameters to variate and few/more amount of steps

function parameter_variation(x, y, DEF_NIND, DEF_MAXGEN, DEF_NVAR,
DEF_ELITIST, STOP_PERCENTAGE, DEF_PR_CROSS, DEF_PR_MUT,
MUTATION,CROSSOVER, DEF_LOCALLOOP, ah1, ah2, ah3 ,
STOP_CRIT,DEF_REPLACE_WORST,REPRESENTATION,number_of_runs);

%%------------------------------------------------------------------------------------
   %parameters to variate  comment/uncomment

   %(Question 2) and (Question 4)
   %parameters = ["NIND", "MAXGEN", "ELITIST", "PROB.CROSS", "PROB.MUT"];
   %(Question 5)
   parameters = ["LOCALLOOP"];
   %(Question 7)
   %parameters = ["REPLACE_WORST"];
%%------------------------------------------------------------------------------------
   ranges = containers.Map;

   %Select amount of stepts (comment/uncomment)

   %range definition for the parameters
   %range for few steps
   ranges("NIND") = 10:300:1000 ;%3 points
   ranges("MAXGEN") = 10:300:1000 ;%3 points
   ranges("ELITIST") = 0:0.2:1; %5 points
   ranges("PROB.CROSS") = 0:0.2:1; %5 points
   ranges("PROB.MUT") = 0:0.2:1; %5 points
   ranges("LOCALLOOP") =  0:1; %1:ON and 0: OFF

%   %ranges for higher amount of steps
%        ranges("NIND") = 10:33:1000; %30 points
%        ranges("MAXGEN") = 10:33:1000; %30 points
%        ranges("ELITIST") = 0:0.05:1; %20 points
%        ranges("PROB.CROSS") = 0:0.05:1; %20 points
%        ranges("PROB.MUT") =  0:0.05:1; %20 points
%        ranges("LOCALLOOP") =  0:1; %1:ON and 0: OFF
%        ranges("REPLACE_WORST") =  0:1; %0 for elitism ; 1 for replace worst
%%------------------------------------------------------------------------------------
```

```
  for j = 1:size(parameters,2)%parameter = parameters
    parameter = parameters(j);
    curr_param_vals = ranges(parameter);
    dist_param = zeros(number_of_runs, size(curr_param_vals,2));
    time_min_dist = zeros(number_of_runs, size(curr_param_vals,2));
    gen_min_dist = zeros(number_of_runs, size(curr_param_vals,2));

     %Assign default values for parameters
     NIND = DEF_NIND;
     MAXGEN = DEF_MAXGEN;
     NVAR = DEF_NVAR;
     ELITIST = DEF_ELITIST;
     PR_CROSS=DEF_PR_CROSS;
     PR_MUT=DEF_PR_MUT;
     LOCALLOOP = DEF_LOCALLOOP;
     REPLACE_WORST = DEF_REPLACE_WORST;

  %Select which parameter to change, keeping rest as the default value
  switch parameter

     case "NIND"
        for i = 1:size(curr_param_vals,2)
           NIND = curr_param_vals(i);
           [best_all_gen , best_gen_time, best_gen,~] = run_experiment(x, y, NIND,
MAXGEN, NVAR, ELITIST, STOP_PERCENTAGE, PR_CROSS, PR_MUT, MUTATION,
CROSSOVER, LOCALLOOP, ah1, ah2, ah3,
STOP_CRIT,REPLACE_WORST,REPRESENTATION, number_of_runs);
           dist_param(:,i) = best_all_gen;
           time_min_dist(:,i) = best_gen_time;
           gen_min_dist(:,i) = best_gen;
        end
      case "MAXGEN"
         for i = 1:size(curr_param_vals,2)
           MAXGEN = curr_param_vals(i);
           [best_all_gen , best_gen_time, best_gen, ~] = run_experiment(x, y, NIND,
MAXGEN, NVAR, ELITIST, STOP_PERCENTAGE, PR_CROSS, PR_MUT, MUTATION,
CROSSOVER, LOCALLOOP, ah1, ah2, ah3,
STOP_CRIT,REPLACE_WORST,REPRESENTATION, number_of_runs);
           dist_param(:,i) = best_all_gen;
           time_min_dist(:,i) = best_gen_time;
           gen_min_dist(:,i) = best_gen;
        end
     case "ELITIST"
        for i = 1:size(curr_param_vals,2)
           ELITIST = curr_param_vals(i);
           [best_all_gen , best_gen_time, best_gen, ~] = run_experiment(x, y, NIND,
MAXGEN, NVAR, ELITIST, STOP_PERCENTAGE, PR_CROSS, PR_MUT, MUTATION,
CROSSOVER, LOCALLOOP, ah1, ah2, ah3,
STOP_CRIT,REPLACE_WORST,REPRESENTATION, number_of_runs);
           dist_param(:,i) = best_all_gen;
```

```
                time_min_dist(:,i) = best_gen_time;
                gen_min_dist(:,i) = best_gen;
             end
        case "PROB.CROSS"
            for i = 1:size(curr_param_vals,2)
                PR_CROSS = curr_param_vals(i);
                [best_all_gen , best_gen_time, best_gen, ~] = run_experiment(x, y, NIND,
MAXGEN, NVAR, ELITIST, STOP_PERCENTAGE, PR_CROSS, PR_MUT, MUTATION,
CROSSOVER, LOCALLOOP, ah1, ah2, ah3,
STOP_CRIT,REPLACE_WORST,REPRESENTATION, number_of_runs);
                dist_param(:,i) = best_all_gen;
                time_min_dist(:,i) = best_gen_time;
                gen_min_dist(:,i) = best_gen;
             end
        case "PROB.MUT"
            for i = 1:size(curr_param_vals,2)
                PR_MUT = curr_param_vals(i);
                [best_all_gen , best_gen_time, best_gen, ~] = run_experiment(x, y, NIND,
MAXGEN, NVAR, ELITIST, STOP_PERCENTAGE, PR_CROSS, PR_MUT, MUTATION,
CROSSOVER, LOCALLOOP, ah1, ah2, ah3,
STOP_CRIT,REPLACE_WORST,REPRESENTATION, number_of_runs);
                dist_param(:,i) = best_all_gen;
                time_min_dist(:,i) = best_gen_time;
                gen_min_dist(:,i) = best_gen;
             end
        case "LOCALLOOP"
            if(REPRESENTATION == 0) %only works for path representation (errors in adj
crossover)
                %For local heuristic
                %Matrix: indep_runs x gens x param
                best_per_gen_matx = zeros(number_of_runs ,MAXGEN,size(curr_param_vals,2) );
                for i = 1:size(curr_param_vals,2)
                    LOCALLOOP = curr_param_vals(i);
                     [~ , ~, ~, best_per_gen] = run_experiment(x, y, NIND, MAXGEN, NVAR,
ELITIST, STOP_PERCENTAGE, PR_CROSS, PR_MUT, MUTATION, CROSSOVER,
LOCALLOOP, ah1, ah2, ah3, STOP_CRIT,REPLACE_WORST,REPRESENTATION,
number_of_runs);
                     best_per_gen_matx(:,:, i) = best_per_gen;

                end
            end
        case "REPLACE_WORST"
            if(REPRESENTATION == 0) %only works for path representation
                %survivor selection strategy: REPLACE_WORST
                %Matrix: indep_runs x gens x param
                best_per_gen_matx = zeros(number_of_runs ,MAXGEN,size(curr_param_vals,2) );
                for i = 1:size(curr_param_vals,2)
                    REPLACE_WORST = curr_param_vals(i);
                     [~ , ~, ~, best_per_gen] = run_experiment(x, y, NIND, MAXGEN, NVAR,
ELITIST, STOP_PERCENTAGE, PR_CROSS, PR_MUT, MUTATION, CROSSOVER,
LOCALLOOP,   ah1,   ah2,   ah3,   STOP_CRIT,REPLACE_WORST,REPRESENTATION,
number_of_runs);
                best_per_gen_matx(:,:, i) = best_per_gen;

            end
         end

   end  %end switch



   filename = "results/" + parameter + ".mat";

   %Iterate for every parameter  (variate one parameter at a time)
   figure

   if(parameter == "LOCALLOOP" || parameter ==  "REPLACE_WORST")
      save(filename, 'curr_param_vals', 'best_per_gen_matx');
      color = ['r', 'b'];

      for  i = 1:size(curr_param_vals,2)

         p(i)   =   stdshade(best_per_gen_matx(:,:,i),0.1,color(i),0:(size(best_per_gen_matx,2)-
1));
         hold on
         grid on
         xlabel('Generation');
         ylabel('Avg. Best solution per gen. across runs');

      end
      if(parameter == "LOCALLOOP")
         legend([p(1) p(2)],{'Local loop OFF','Local loop ON'});
         title1 =  sprintf('Avg. results for %d independent runs.' ,number_of_runs);
         title({'Local heuristic comparision (ON/OFF)';title1});
      else
         %0 for elitism ; 1 for replace worst
         legend([p(1) p(2)],{'Elitism','Replace worst'});
         title1 =  sprintf('Avg. results for %d independent runs.' ,number_of_runs);
         title({'Survivor selection strategy';title1});
      end
      hold off

   else
       %Store variables
      save(filename, 'curr_param_vals', 'dist_param' , 'time_min_dist', 'gen_min_dist');

      subplot(1,2,1);
      stdshade(dist_param,0.1,'r',curr_param_vals);
      grid on
      xlabel(parameter);
```

```
        ylabel('Avg. Best solution across runs');


        subplot(1,2,2);
        stdshade(gen_min_dist,0.1,'b',curr_param_vals);
        grid on
        xlabel(parameter);
        ylabel('Avg generations to get best sol. (s)');
    end

    end

end %End of function
```

Run experiment (called from parameter variation):

```
%Average best travelled distance and time of convergence.

function [dist_param , time_min_dist , gen_min_dist, best_per_gen_matx] = run_experiment(x, y, NIND, MAXGEN, NVAR, ELITIST, STOP_PERCENTAGE, PR_CROSS, PR_MUT, MUTATION, CROSSOVER, LOCALLOOP, ah1, ah2, ah3, STOP_CRIT,REPLACE_WORST,REPRESENTATION, number_of_runs)
dist_param = zeros(number_of_runs, 1);
time_min_dist = zeros(number_of_runs, 1);
gen_min_dist = zeros(number_of_runs ,1);
best_per_gen_matx = zeros(number_of_runs ,MAXGEN);

for i=1:number_of_runs
    [best_all_gen , best_gen_time, best_gen , best_per_gen] = run_ga_customized(x, y, NIND, MAXGEN, NVAR, ELITIST, STOP_PERCENTAGE, PR_CROSS, PR_MUT, MUTATION, CROSSOVER, LOCALLOOP, ah1, ah2, ah3,STOP_CRIT,REPLACE_WORST,REPRESENTATION);
    dist_param(i) = best_all_gen;
    time_min_dist(i) = best_gen_time;
    gen_min_dist(i) = best_gen;
    best_per_gen_matx(i,:) = best_per_gen;
end

end
%end function
```

Stopping Criterion:

```
classdef stopping_criteria

 properties
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
gen_comparision_percen = 0.5 ; % Maximal improvement of solution over last N generations
(N = gen*gen_comparision_percen)
min_improvement = 0.05 ;  % minimum percentage of improvement: 30%
min_diversity = 0.1; %minimum diversity limit below which the algorithm stops
thr_gen = 50 ; %When to start checking for stopping criteria
efficiency_lim = 0.0625 ; %efficiency should not be below this limit - To be defined : Desired
min lengt? 100/(d*gen)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 end

methods

    function STOP = efficiency_limit(Obj, curr_gen_maxFitness, gen)
        if (gen > Obj.thr_gen)
            curr_gen_efficiency = curr_gen_maxFitness/(gen+1);
            if (curr_gen_efficiency < Obj.efficiency_lim )
                disp("Stopping criteria: Total efficiency should not be lower than a defined limit");
                STOP = true;
            end
        end
    end

   function STOP = max_improvement(Obj, gen, best, best_all_gen)

        STOP = false ;

        N_gens = round(Obj.gen_comparision_percen*(gen+1));

        if (N_gens > 0 && gen > Obj.thr_gen)
            last_N_gens = best((gen-N_gens+1):gen+1);

            if(all( ((last_N_gens - best_all_gen)/best_all_gen) < Obj.min_improvement))
                disp("Stopping criteria: Maximal improvement of solution over last generations");
                STOP = true;

            end
        end

   end

   function STOP = diversity_pheno(Obj, Fitness, gen)
       %Terminate if diversity in the phenotype space is lower than limit.
       %Diversity is measured in terms of fitness variance.
       STOP = false;
       if (gen > Obj.thr_gen)
           curr_gen_diversity = var(Fitness);
           if (curr_gen_diversity < Obj.min_diversity)
```

```
            disp("Stopping criteria: diversity in the phenotype space is lower than limit");
            STOP = true;
         end
      end
   end

   function STOP = choose_stopping_criteria(sc, stop_crit, curr_gen_maxFitness, gen, best,
best_all_gen, parents_Fitness)

      switch stop_crit
         case 1 %max_improvement
            STOP = sc.max_improvement(gen , best, best_all_gen);
         case 2 %diversity in phenotype/fitness function
            STOP = sc.diversity_pheno(parents_Fitness,gen);
         case 3 %efficiency
            STOP = sc.efficiency_limit(curr_gen_maxFitness,gen);
         otherwise
            %warning('Unexpected stopping criterion type.')
            STOP = false ;
      end
   end

end %end methods
end
```

Path representation:

```
Chrom=zeros(NIND,NVAR);
      for row=1:NIND
         if(REPRESENTATION==1) %1: adj repr
            Chrom(row,:)=path2adj(randperm(NVAR));
         else
            Chrom(row,:)=randperm(NVAR);
         end

      end
```

To visualize:

```
visualizeTSP(x,y,(Chrom(t,:)), minimum, ah1, gen, best, mean_fits, worst, ah2, ObjV, NIND,
ah3);
```

To calculate minimum distance travelled (cost):

```
function ObjVal = tspfun_path(Phen, Dist)

   ObjVal=zeros(size(Phen,1),1);
   for i=1:size(Phen(:,1))
      for j=1:size(Phen,2)-1
         ObjVal(i)=ObjVal(i)+Dist(Phen(i,j),Phen(i,j+1));
      end
    end

end % End of function
```

Order Crossover:

```
function NewChrom = order_crossover(OldChrom, XOVR);

if nargin < 2, XOVR = NaN; end

[rows,cols]=size(OldChrom);

   maxrows=rows;
%to test if the #rows or individuals is even
%as we take 2 parents at a time for crossover
   if rem(rows,2)~=0
       maxrows=maxrows-1;
   end

   for row=1:2:maxrows

    % crossover of the two chromosomes
    % results in 2 offsprings
    if rand<XOVR            % recombine with a given probability
        NewChrom(row,:) =cross_OX([OldChrom(row,:);OldChrom(row+1,:)]); %Parent1 cross
Parent2 = child1
        NewChrom(row+1,:)=cross_OX([OldChrom(row+1,:);OldChrom(row,:)]);       %Parent2
cross Parent1 = child2
    else
        NewChrom(row,:)=OldChrom(row,:);
        NewChrom(row+1,:)=OldChrom(row+1,:);
    end
   end

   if rem(rows,2)~=0
       NewChrom(rows,:)=OldChrom(rows,:);
   end
```

crossOX (called from order crossover):

```
function Offspring=cross_OX(Parents);
    cols=size(Parents,2);
    Offspring=zeros(1,cols);

    %select random positions
    rndi=zeros(1,2);

    while (abs(rndi(1)-rndi(2))<=1)
```

```
        rndi=rand_int(1,2,[1 cols]);
    end
    rndi = sort(rndi);
    %step 1: copy randomly selected segment from first parent into offspring
    for i=rndi(1):rndi(2)
        Offspring(i)=Parents(1,i);
    end

    %step 2: copy rest of alleles in order they appear in second parent, treating string as toroidal
    j=increment(i,cols); %j is index in parent2
    k=increment(i,cols); %k is index in offspring
    while(any(Offspring(:) == 0))
        if all(Parents(2,j)~= Offspring)
            Offspring(k)=Parents(2,j);
            k=increment(k,cols);
        else
            disp("");
        end
        j=increment(j,cols);
    end
%end function
```

Insertion Mutation:

```
function NewChrom = insertion(OldChrom,Representation);

NewChrom=OldChrom;

if Representation==1
    NewChrom=adj2path(NewChrom);
end

% select two positions in the tour randomly

rndi=zeros(1,2);

%select random values until the difference between them is grater than one
while (abs(rndi(1)-rndi(2))<=1)
    rndi=rand_int(1,2,[1 size(NewChrom,2)]);
end
rndi = sort(rndi);

%second city moved next to the first
NewChrom(rndi(1)+1) = OldChrom(rndi(2)) ;

%shuffling along the others to make room
for i = rndi(1):rndi(2)-2
    NewChrom(i+2) = OldChrom(i+1) ;
end
```

```
if Representation==1
    NewChrom=path2adj(NewChrom);
end
```

Worst Replacement Survival selection:

```
function [Chrom ObjV] =  worse_replacement(Chrom, offspring, ObjV ,offs_ObjV);
    %Calculate fitness values
    parents_Fitness = fitness_funct(ObjV);
    offs_Fitness = fitness_funct(offs_ObjV);

    %Compare fitness values
    for i = 1:size(Chrom,1)
        if(offs_Fitness(i)> parents_Fitness(i))
            Chrom(i,:) = offspring(i,:);
            ObjV(i) = offs_ObjV(i);
        end
    end
end
% End of function
```

Fitness function:

```
function Fitness = fitness_funct(ObjV);

    constant_fitness = 100; %created to avoid very small number in efficiency
    Fitness = constant_fitness./(ObjV);

end
```