**Genetic Algorithms and Evolutionary Computing Project**

**2018-2019**

Authors:

Swarnalata Patra (r0729319)

Maria Camila Alvarez Trivino

Evaluator:

Prof Dirk Roose

**INTRODUCTION**

The **travelling salesman problem**(**TSP**) is based 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**

Table1 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 keeping the rest untouched. The predefined 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. It shows the effect of the change in 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. For plotting, we considered the number of the generations as a measure of time (how long the algorithm runs).

Then we implemented few stopping criteria mentioned in Table1. These 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 the performance. Then we can decide which one works best for a particular problem. So, we implemented a seed function to have the same values for a better comparison.

“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 used the basic crossover and mutation methods suitable for this representation as specified in Table1. The same parameter variation function was used here, to tune the parameter values and get an optimal combination 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 two of the benchmark problems provided. We modified the no. of individuals and maximum generation 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 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 | - |

**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:**

**No. of individuals**: In Fig1.a, we observe that, with the increase in the no. of individuals, we get better results as the min distance travelled across all generations reduces further. 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.

**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 desired 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).

**Probability of Crossover**: In Fig1.c, we can see that the best solution improves till a certain value of the crossover probability and after that they don’t improve any further. In adjacency, 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. Also, with higher value of crossover probability, the no. of generations needed to converge to a better solution is less. Also, for adjacency, the no. of generations needed to converge reduces with increased crossover probability, but it doesn’t give a good result.

**Probability of Mutation**: In Fig1.d, we observe that the best solution is achieved with a lower value of mutation probability and the results deteriorate with increasing rate. For adjacency, 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 convergence to best solution.

**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 0.1 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 (fitness is inversely proportional to distance travelled), and efficiency is fitness/generations (fitness achieved at the cost of 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. Fig2. shows the comparison of all these stopping criteria along with the default one.

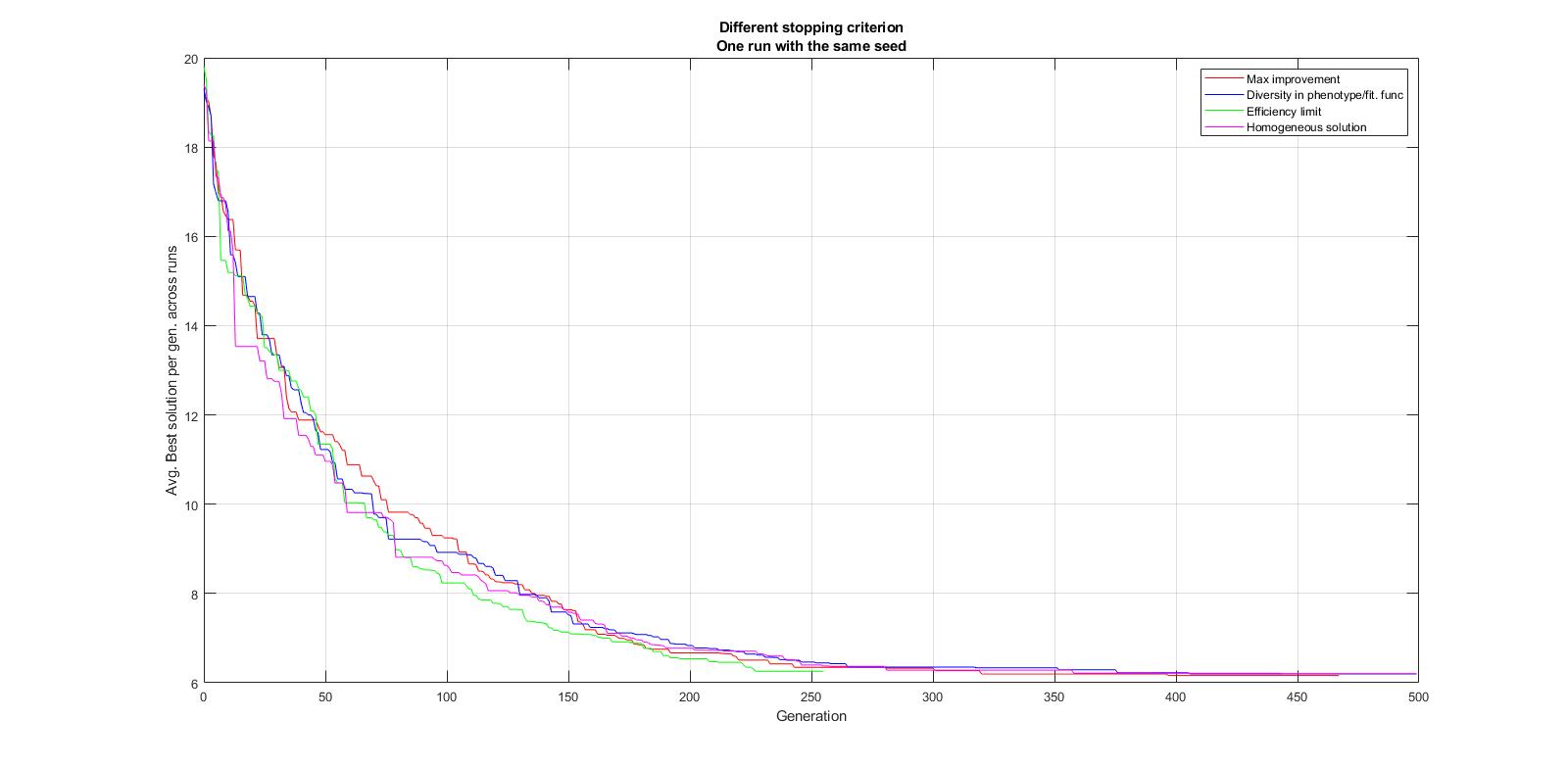


Fig2. Different stopping criterion run with same seed value for comparison.

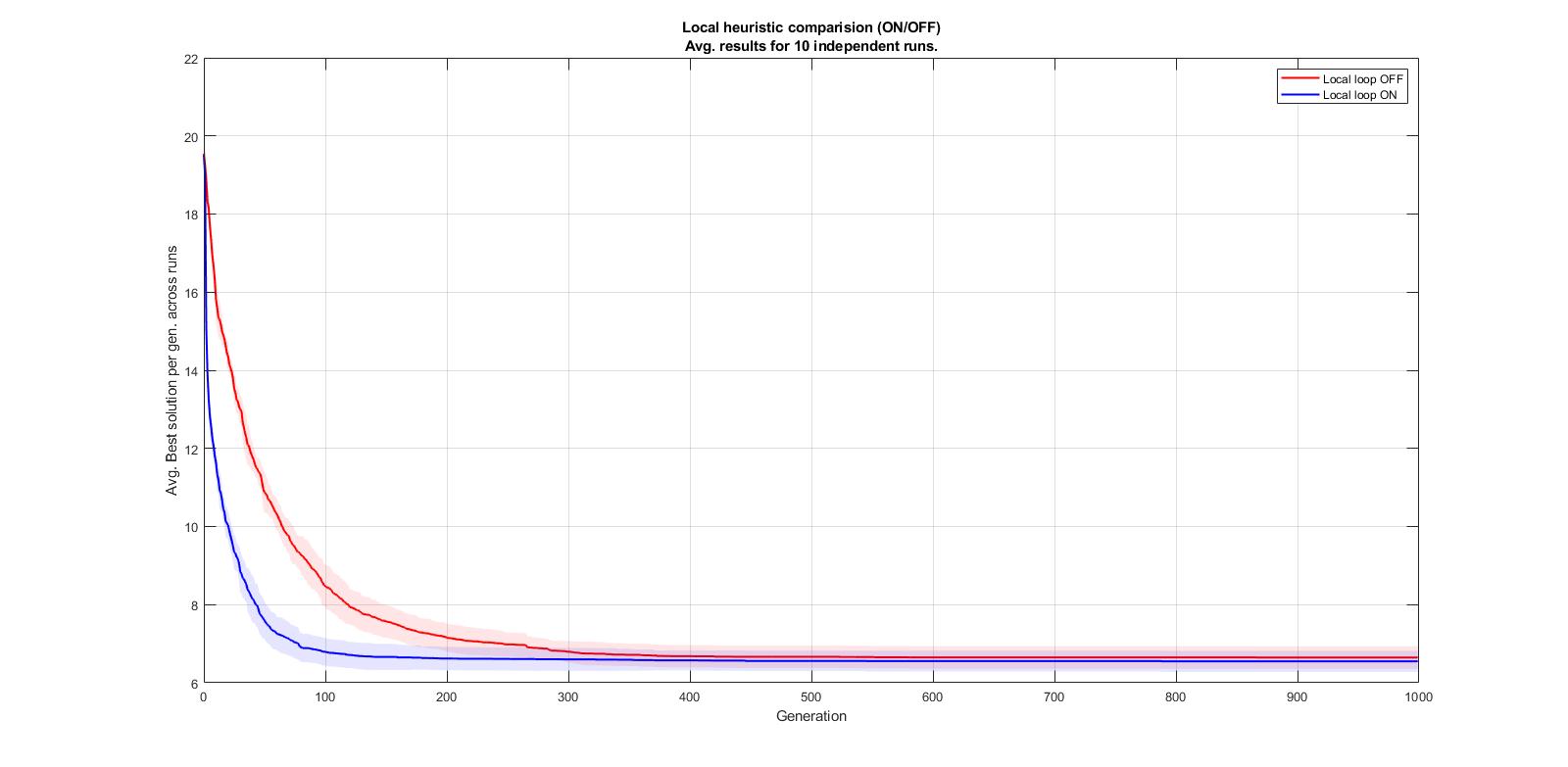
**Parameter Tuning for Path representation:**

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 maximum no. of generations to higher values for better results. So, we decided to use experiment with different combination of parameters for separate problems.

**Impact of Local Optimal Heuristic:**

We used the optimal parameter values from Table3 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, 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) 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 |
| Crossover Type | Order crossover |

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 |
| Crossover Type | Order crossover |

Table4. Optimal Parameter values for 380 cities Benchmark Problem.

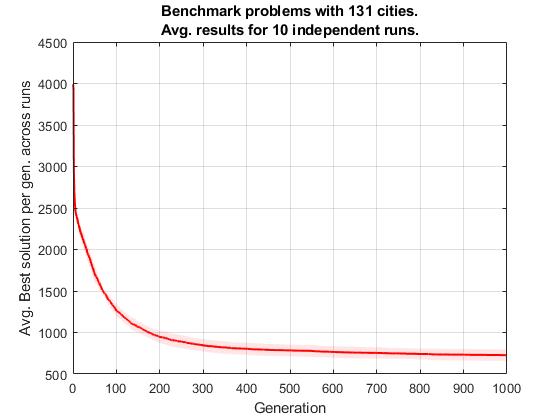


Fig4a. Benchmark Problem with 131 cities. Avg. results for 5 independent runs.

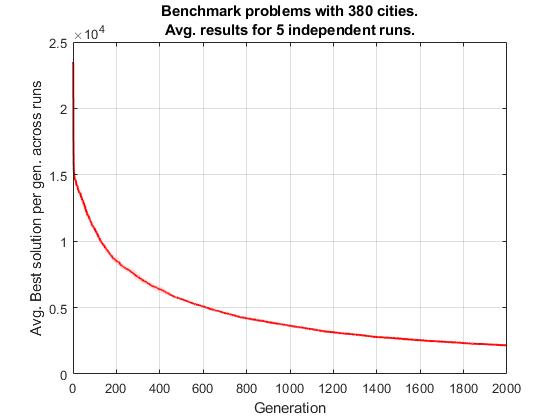


Fig4b. Benchmark Problem with 380 cities. Avg. results for 5 independent runs.

**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. 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).

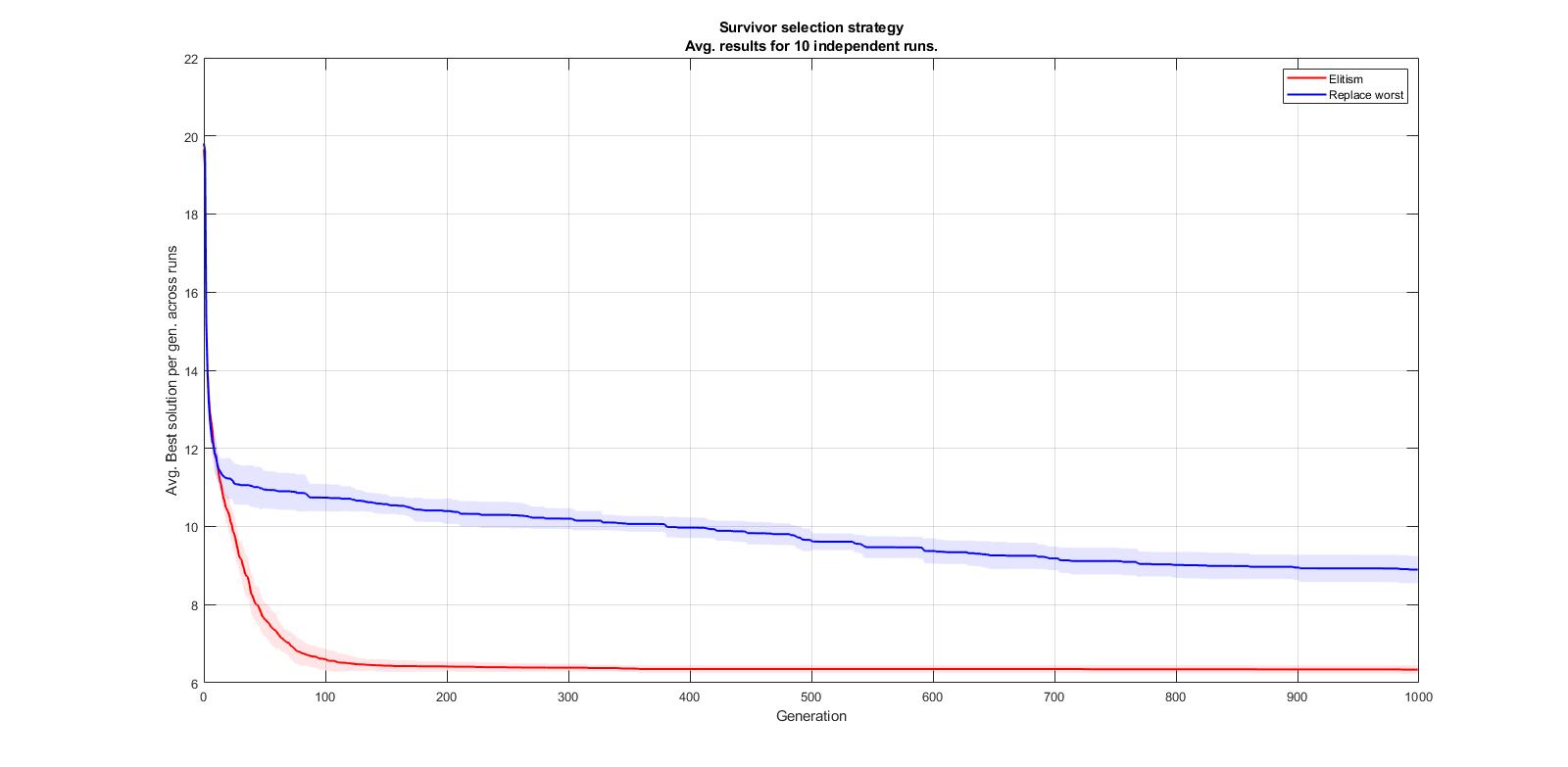


Fig5. Replace Worst vs Elitism Survival Selection strategy.   
Avg. results for 10 independent runs.

**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 probability of losing the 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. Also, the no. of generations for convergence reduces. 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. We observe that no. of generations reduces with 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. However, 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 level of convergence is achieved. We observed from the results that after 250 generations, we get the best solution. So, we want to stop the 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. 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. 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.

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 Fig1.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. Hence, 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. Table5 shows the combination for 51 cities problem.

|  |  |
| --- | --- |
| **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 |

Table5. Optimal Parameter values for 51 cities

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.

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, hence not lost. Hence, this strategy converges to a better solution compared to the other one. Replace worst method, on the other hand, 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.

**CONCLUSION**

# Bibliography

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

**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