CS 776: Evolutionary Computing -Assignment 4 Fall 2018

Full Name: Batyr Charyyev
Email address: bcharyyev@nevada.unr.edu

For this assignment I used GA code [1] (implemented in C) of professor Sushil. I modified selection procedure, crossover and mutation procedure of this GA code.

1) Selection procedure

I used Elitist selection ((μ + λ) selection) as it was recommended in lecture. By creating *n* children from *n* parents, sort and select best *n* individuals from *2n(parents+children)* individuals. Different from canonical GA, selection procedure keeps best individuals from previous population, by doing this we are able to keep best individuals that we encountered so far.

2) Crossover

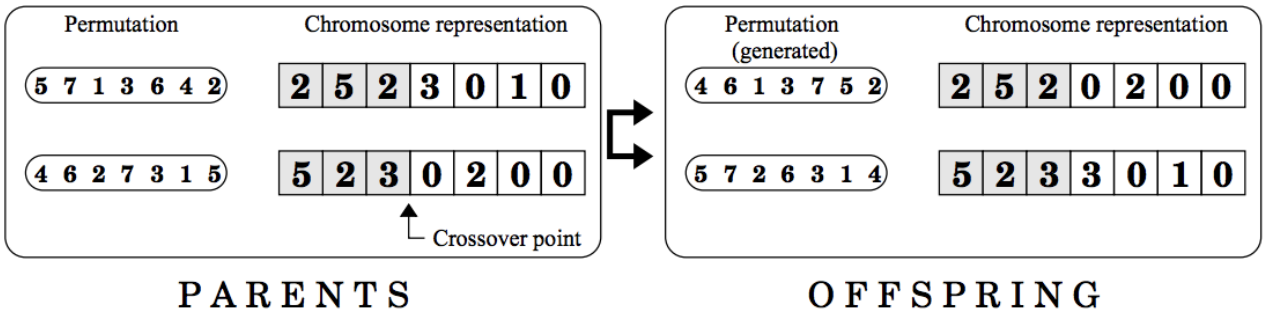
For cross over I used technique proposed by Gokturk [2]. In this paper it first converts permutation of cities into inversion sequence. Then makes crossoever on that inversion sequence and converts it back to permutation of cities. Below I provided algorithm to convert permutation of cities into inversion sequence and inversion sequence back to permutation of cities, also I provided illustration of obtaining offsprings from parents on simple example. All these figures were taken from the article [2].

Input *perm* : array holding the permutation
Output *inv* : array holding the inversion sequence

```
for i ← 1..N do
    { inv_i ← 0
      m ← 1
      while perm_m ≠ i do
            { if perm_m > i then inv_i ← inv_i + 1
              m ← m + 1 } }
```

Input *inv* : array holding the inversion sequence
Output *perm* : array holding the permutation
Uses *pos* : dummy array for intermediate result[1]

```
for i ← N..1 do
    { for m ← i + 1..N do
          if pos_m ≥ inv_i + 1 then pos_m ← pos_m + 1
      pos_i ← inv_i + 1 }
for i ← 1..N do perm_pos_i = i
```



PARENTS OFFSPRING

3) Mutation

For mutation I used Inversion Mutation [3].

0 **1 2 3 4** 5 6 7

becomes

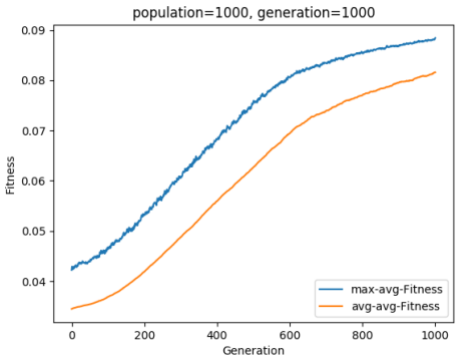
0 **4 3 2 1** 5 6 7

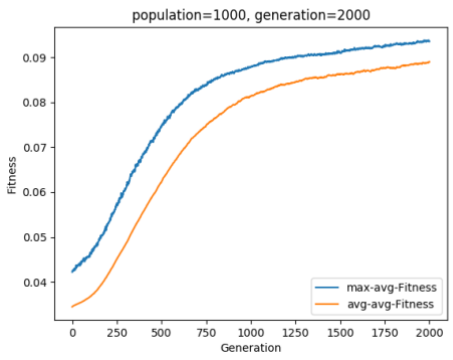
For GA, I took inspiration from [4], this tutorial contains code written in java but I did not use it. Basically each gene represents one city and chromosome represents route that we should take. So in eval function we compute sum of distances between each consecutive cities and try to minimize overall distance.

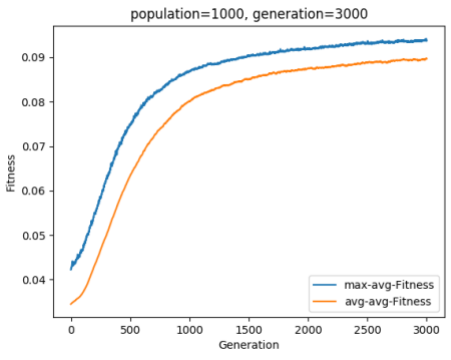
RESULTS

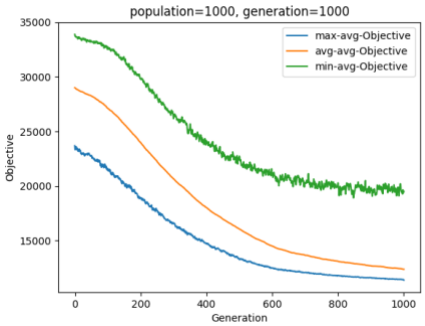
Berlin52

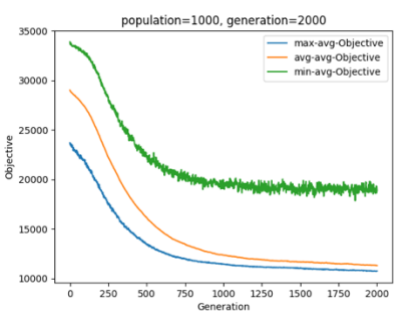
Quality	Reliability	Speed
35%	93%	598250
50%	90%	1047555
51%	93%	1152035
52%	90%	1215037
53%	93%	1951285
65%	93%	2069428
68%	90%	2978888
69%	86%	2349000
73%	96%	3754655

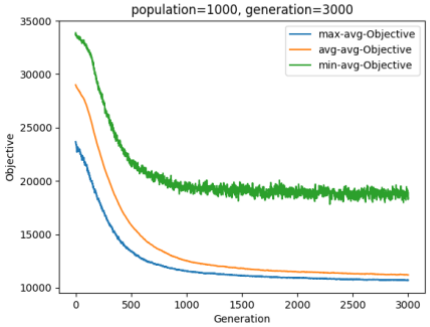


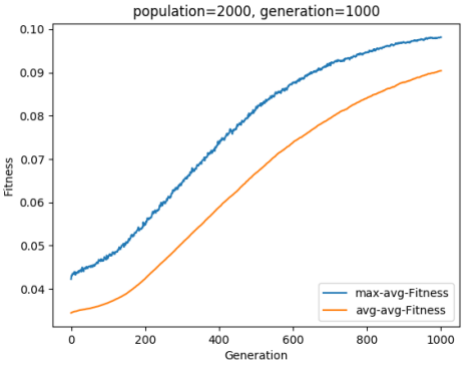


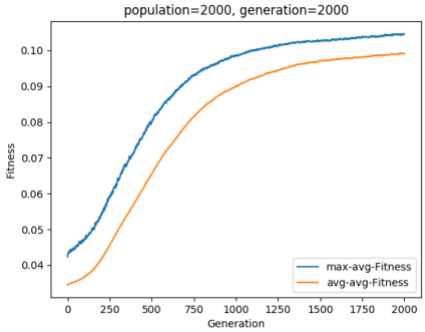


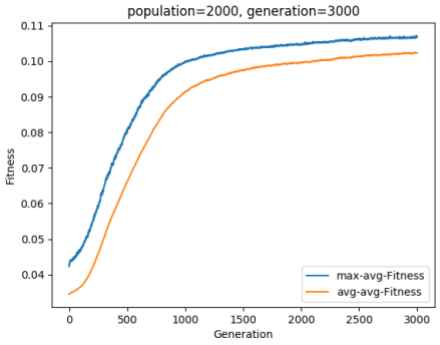


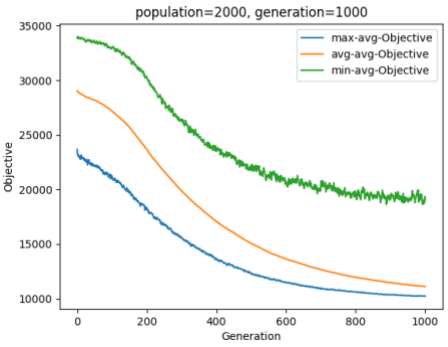


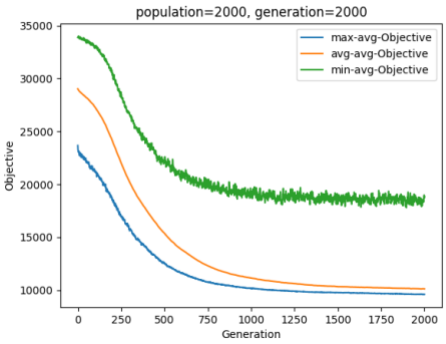


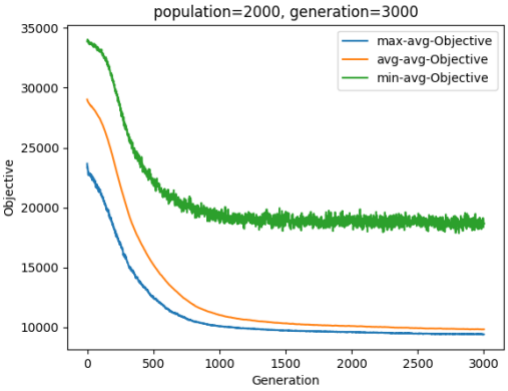

population=2000, generation=1000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

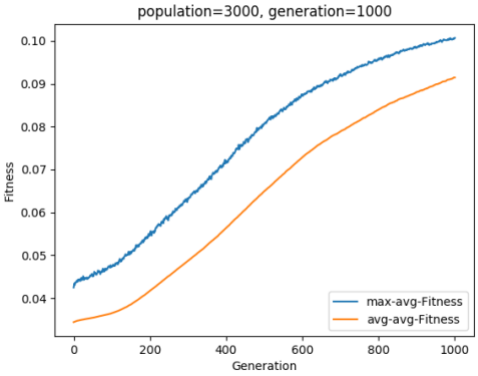

population=2000, generation=2000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

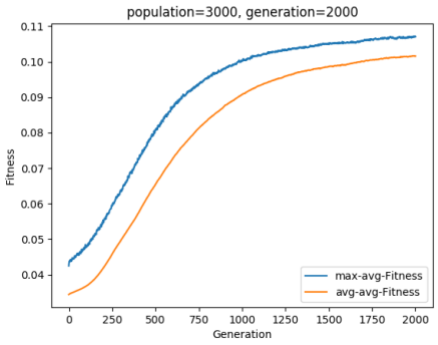

population=2000, generation=3000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

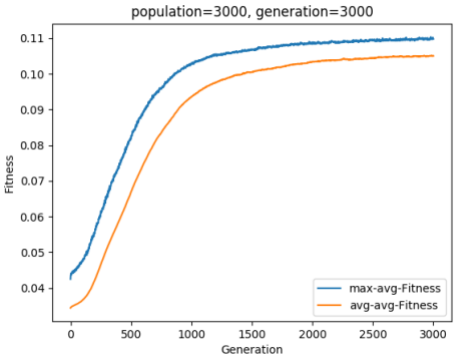

population=2000, generation=1000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

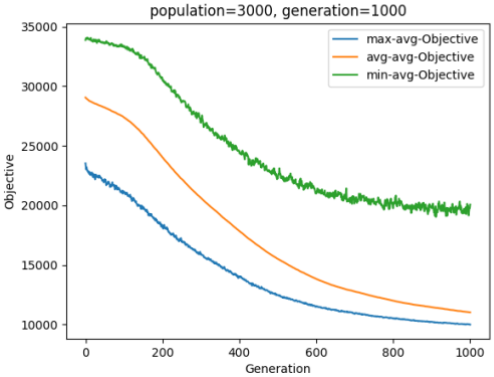

population=2000, generation=2000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

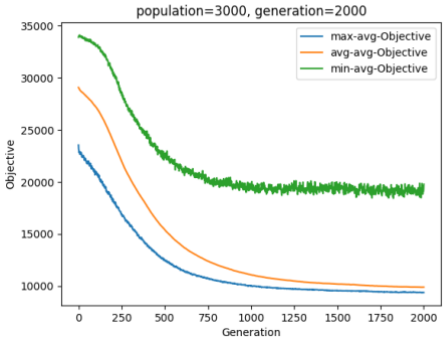

population=2000, generation=3000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

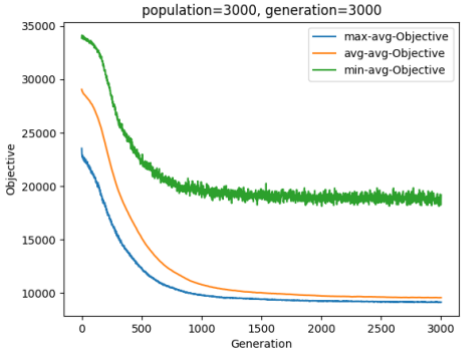

population=3000, generation=1000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=3000, generation=2000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=3000, generation=3000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

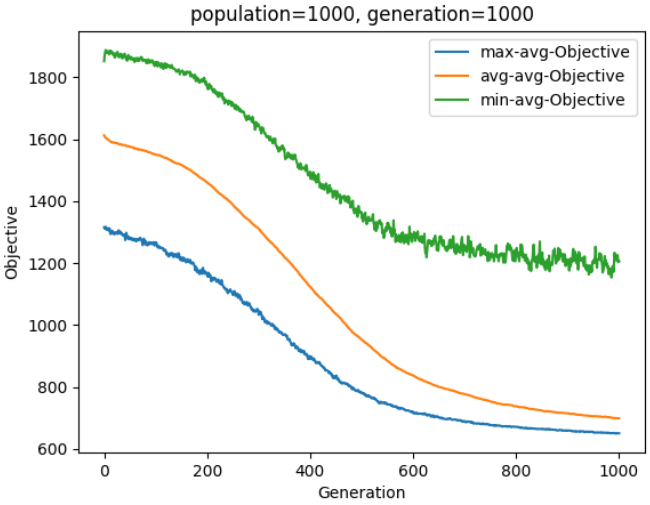

population=3000, generation=1000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

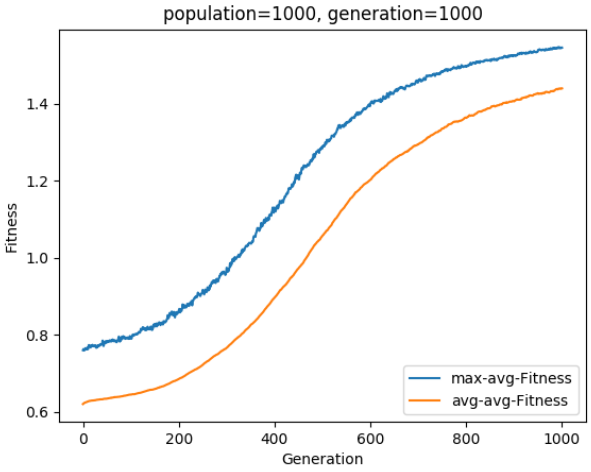

population=3000, generation=2000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

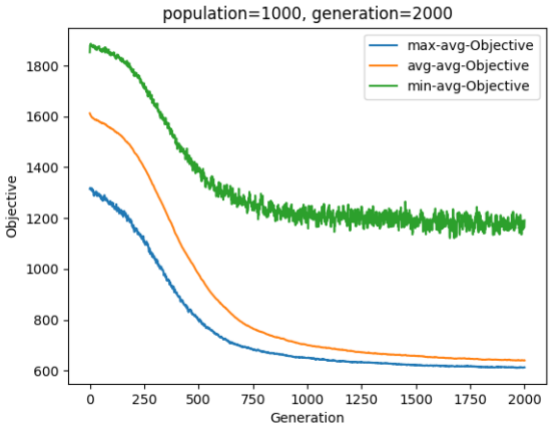

population=3000, generation=3000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

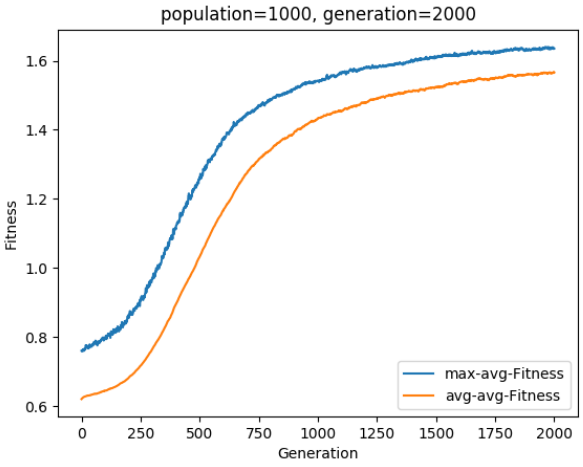
Eil51

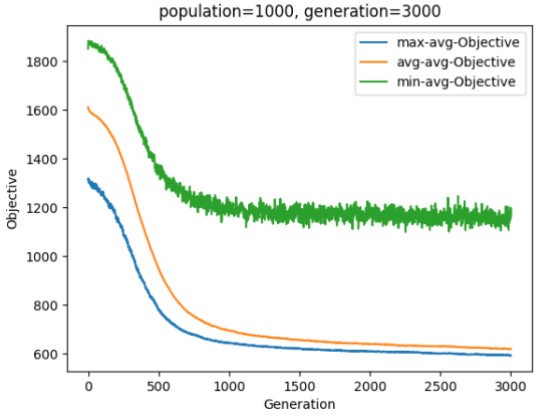
Quality	Reliability	Speed
36%	86%	638500
49%	96%	1024517
53%	100%	1391266
54%	90%	1423444
57%	100%	2166600
64%	90%	2018888
67%	90%	2314074
73%	90%	3655666
74%	86%	3089769

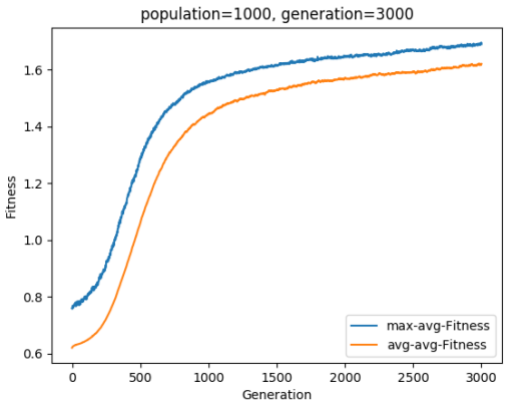


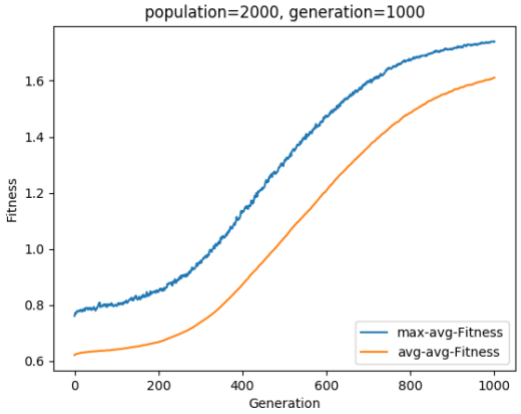


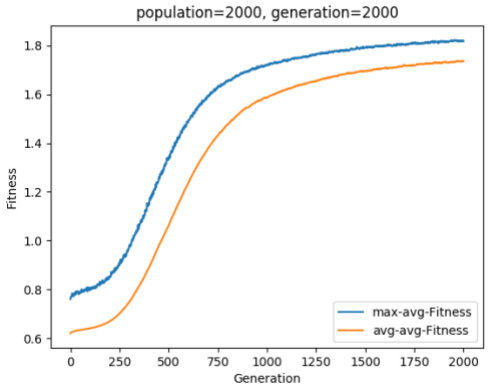


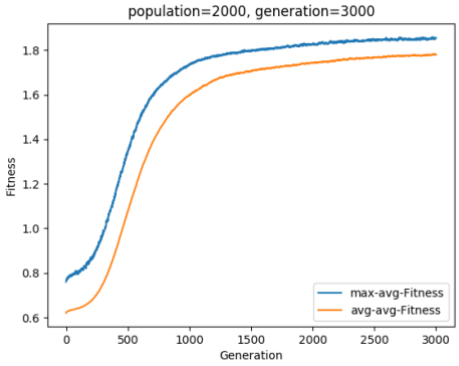


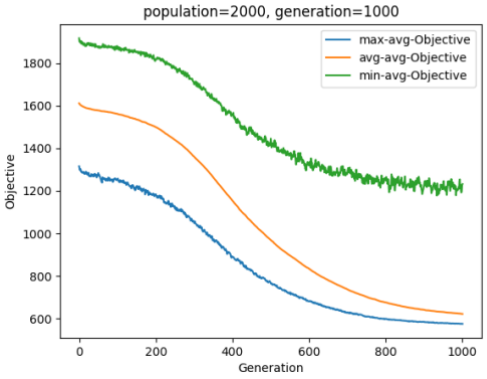


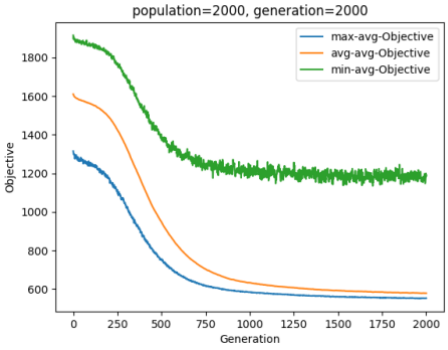


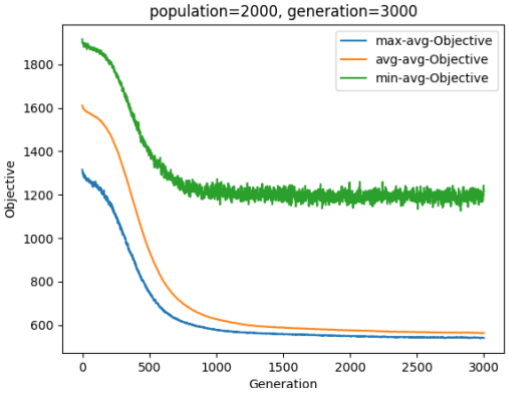

population=2000, generation=1000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

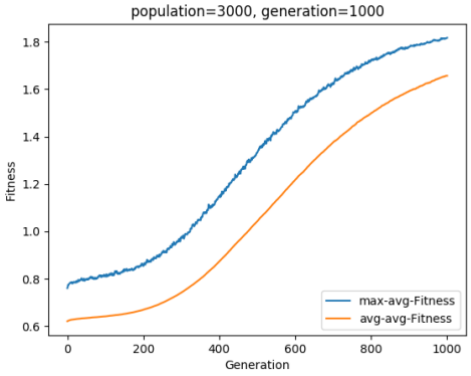

population=2000, generation=2000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

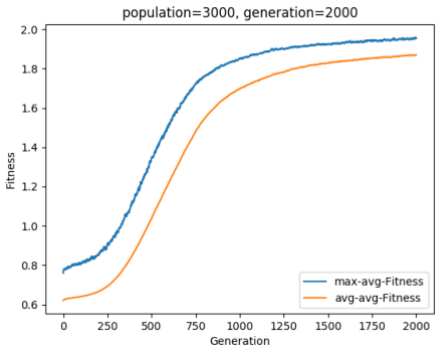

population=2000, generation=3000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

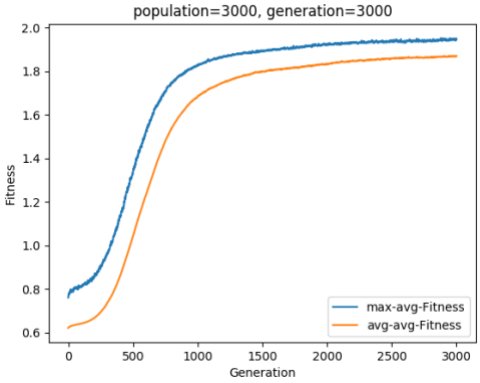

population=2000, generation=1000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

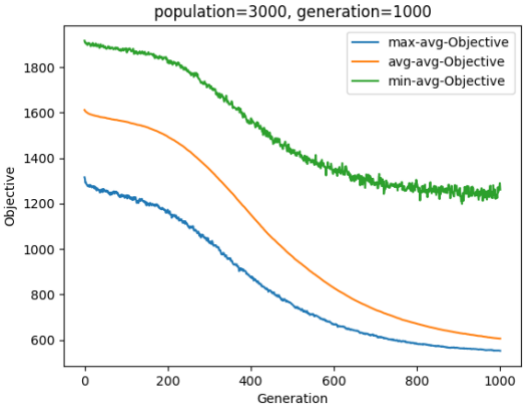

population=2000, generation=2000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

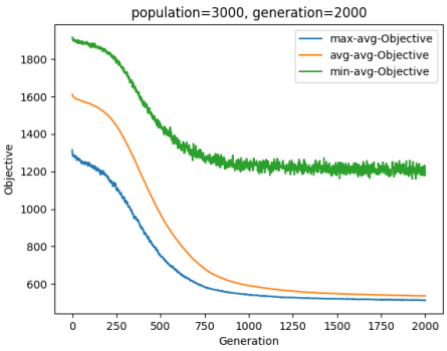

population=2000, generation=3000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

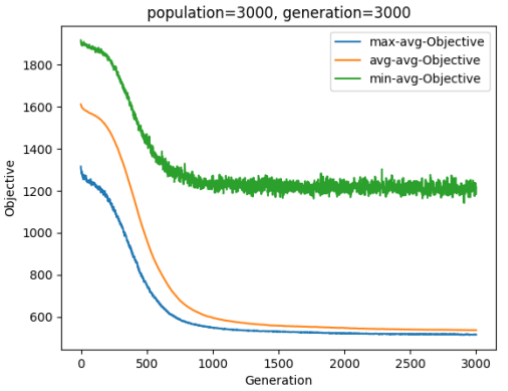

population=3000, generation=1000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=3000, generation=2000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=3000, generation=3000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

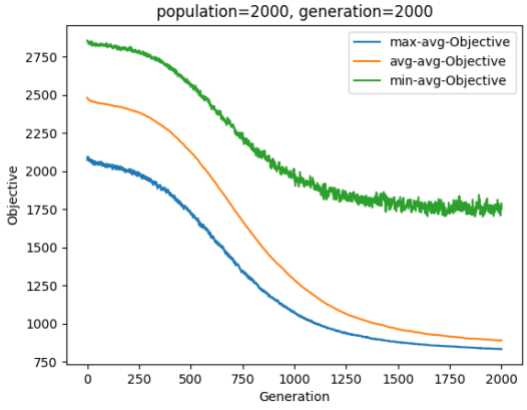

population=3000, generation=1000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

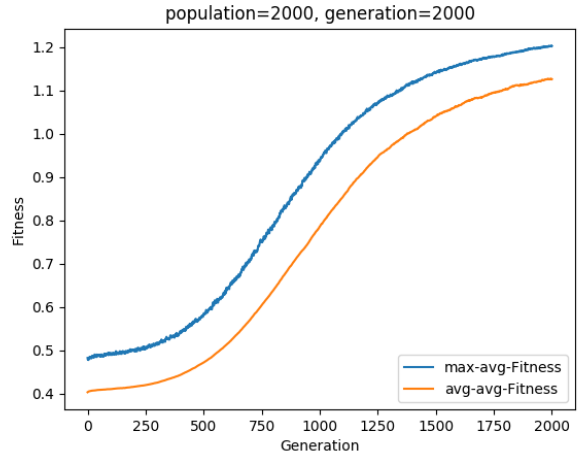

population=3000, generation=2000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

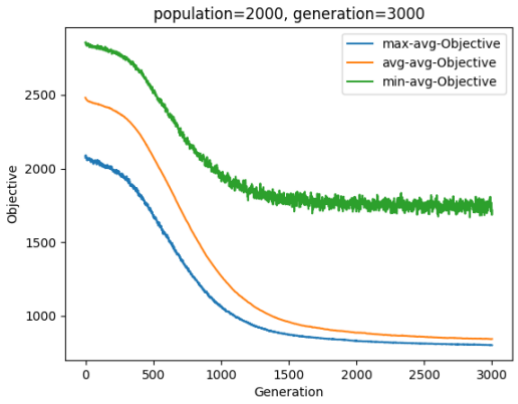

population=3000, generation=3000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

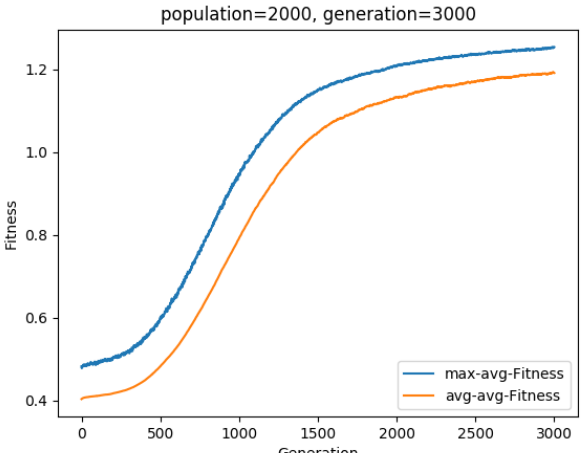
Eil76

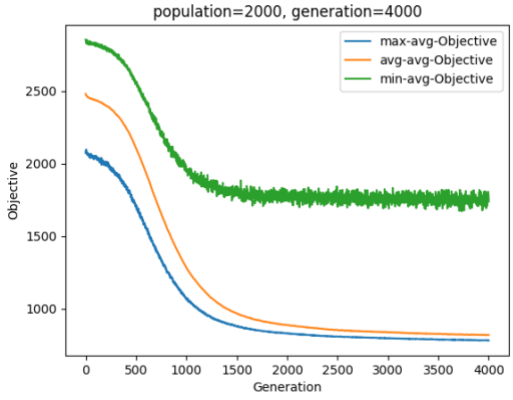
Quality	Reliability	Speed
34%	96%	2781793
43%	83%	3373600
46%	96%	4352379
47%	93%	4219214
51%	96%	5937379
54%	80%	5431625
58%	83%	5924520
59%	80%	7092500
61%	76%	7423652

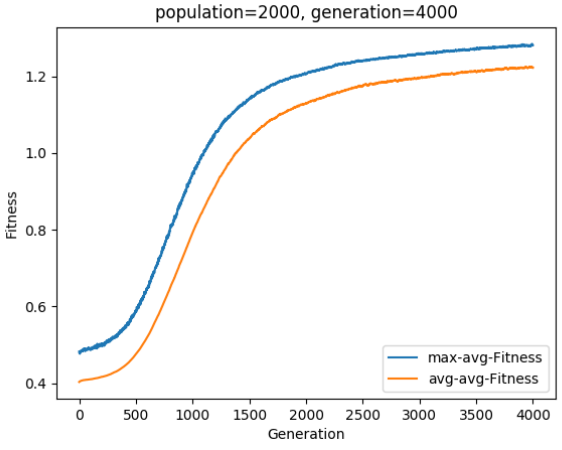


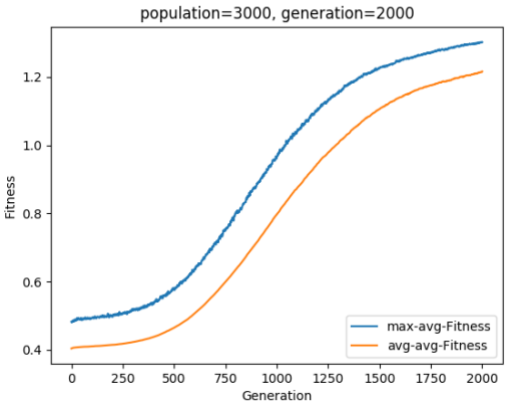


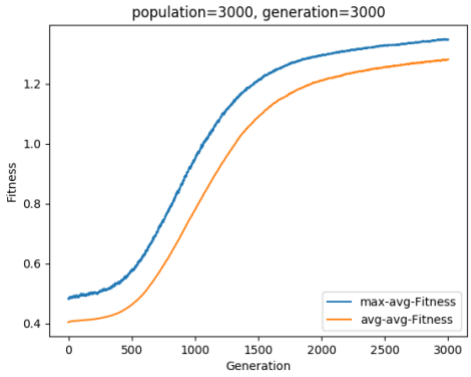


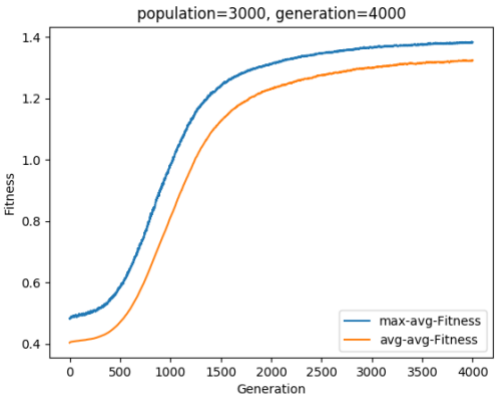


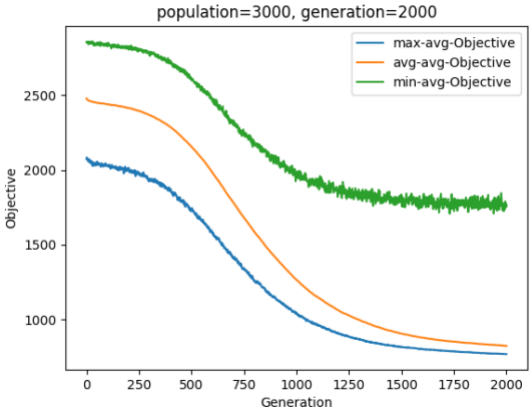


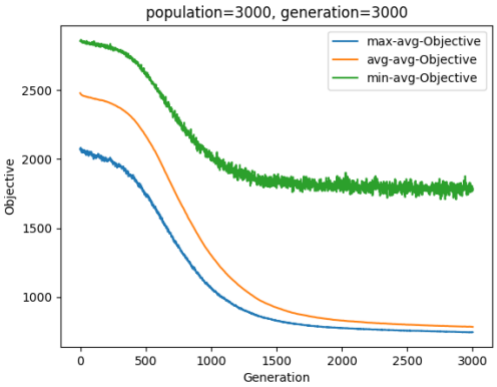


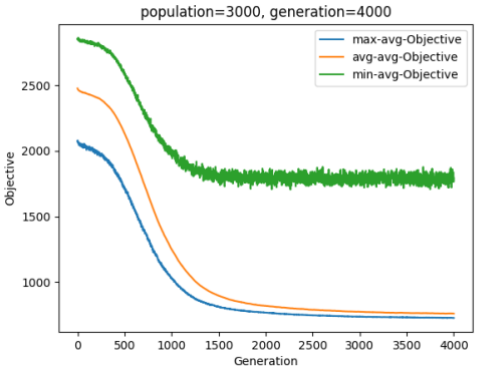

population=3000, generation=2000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

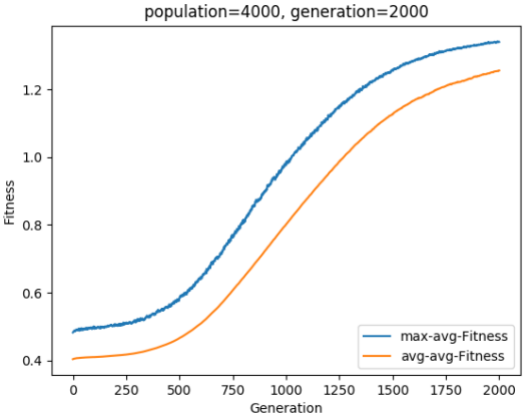

population=3000, generation=3000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

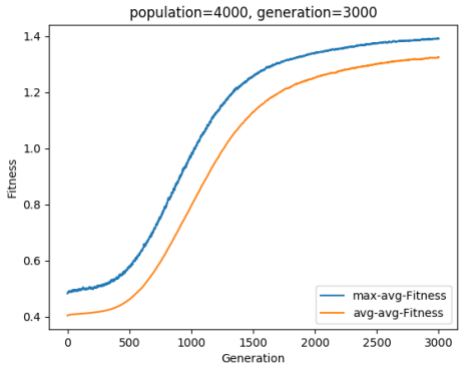

population=3000, generation=4000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

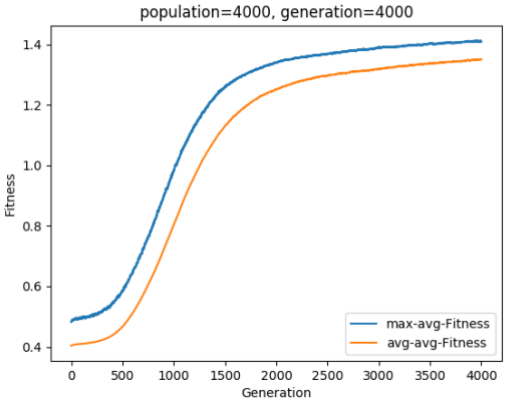

population=3000, generation=2000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

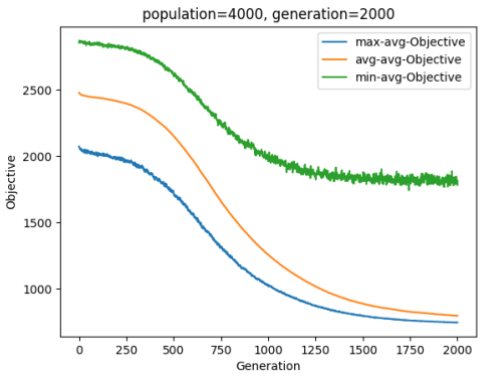

population=3000, generation=3000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

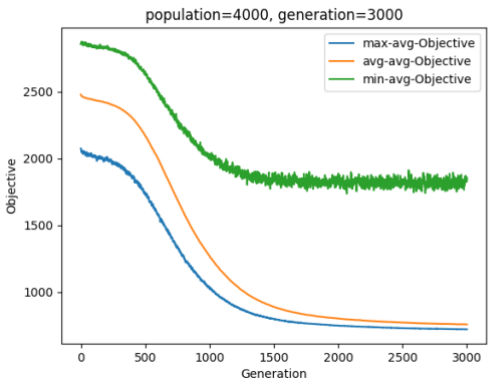

population=3000, generation=4000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

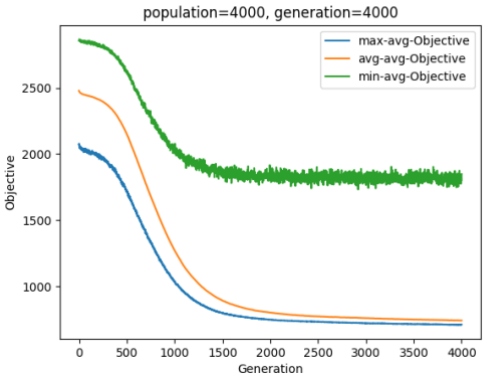

population=4000, generation=2000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=4000, generation=3000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=4000, generation=4000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

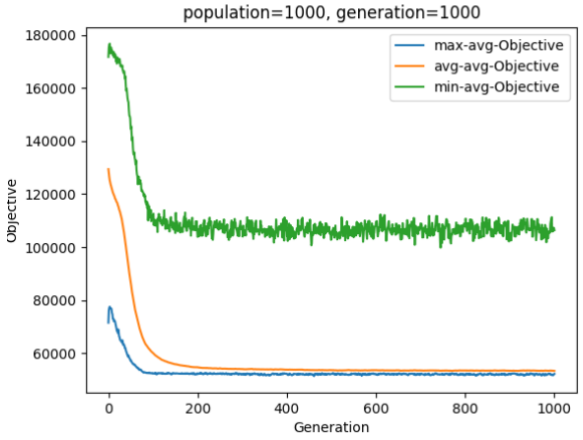

population=4000, generation=2000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

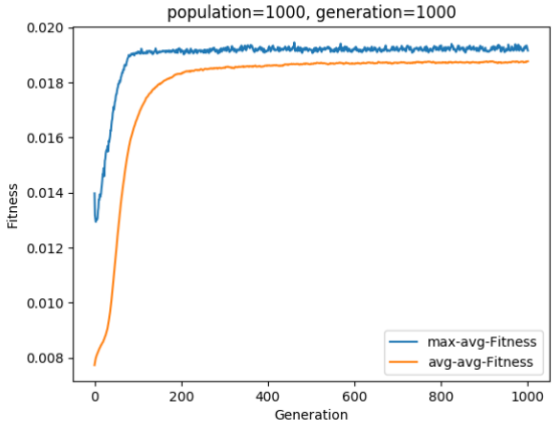

population=4000, generation=3000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

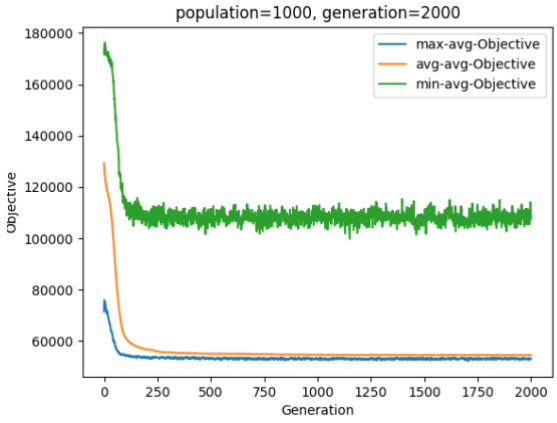

population=4000, generation=4000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

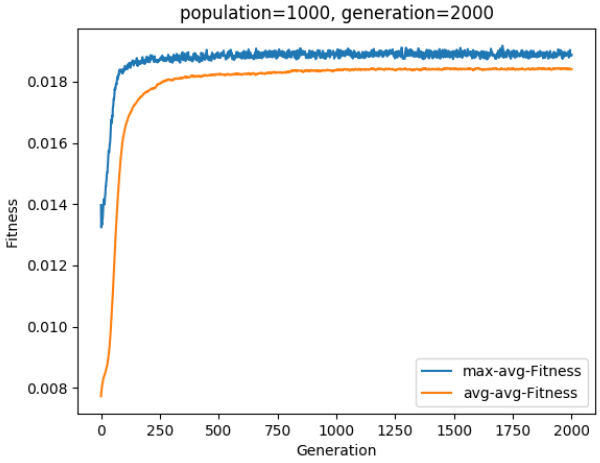
Burma14

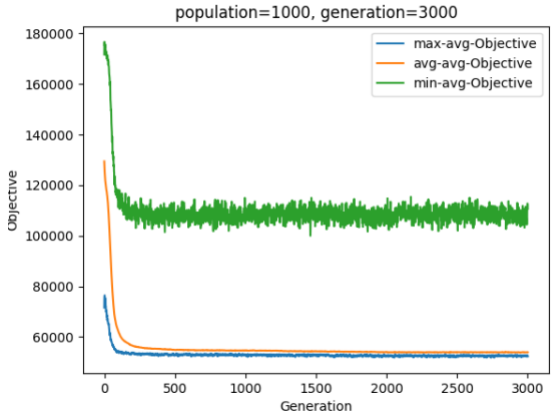
Quality	Reliability	Speed
66%	100%	84466
63%	93%	144964
65%	93%	95714
67%	96%	131724
68%	100%	155600
69%	93%	131357
72%	93%	204321
70%	100%	205100
70%	100%	211300

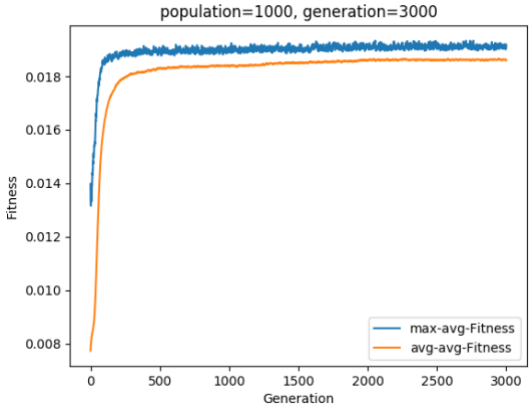


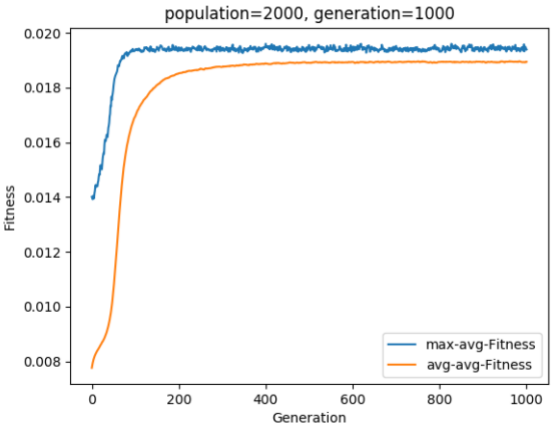


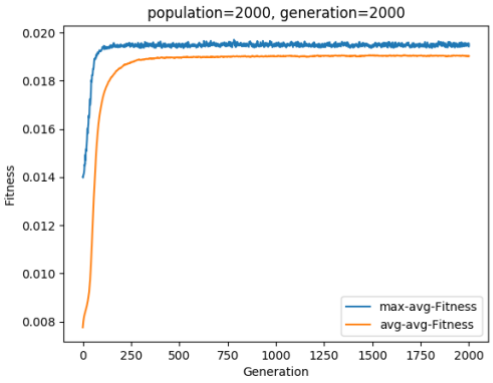


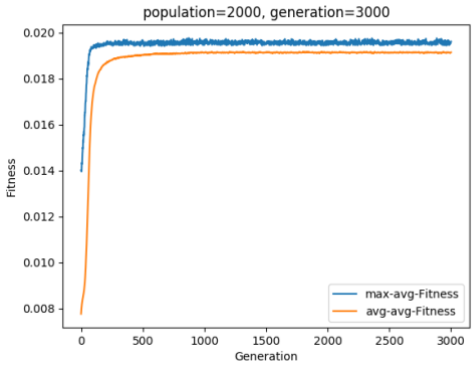


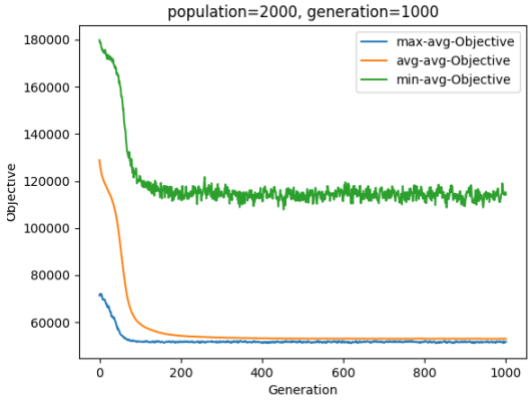


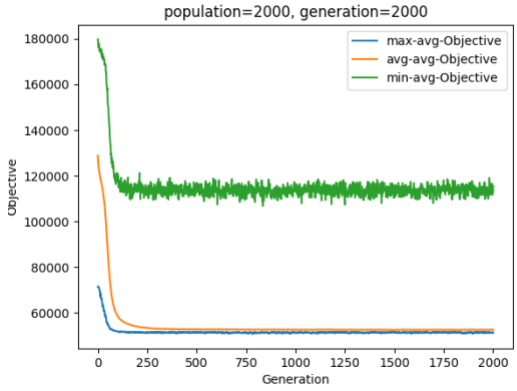


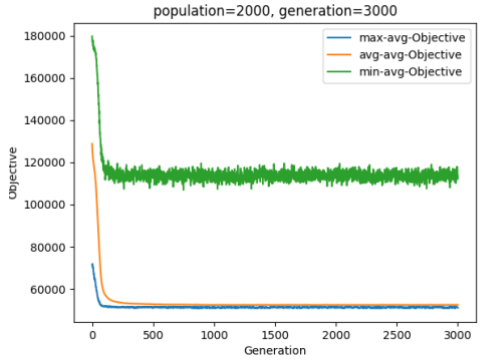

population=2000, generation=1000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

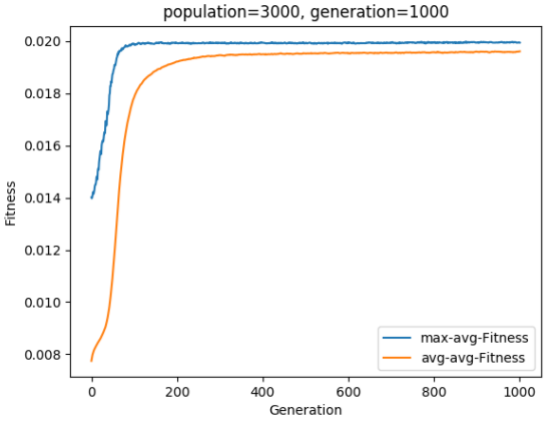

population=2000, generation=2000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

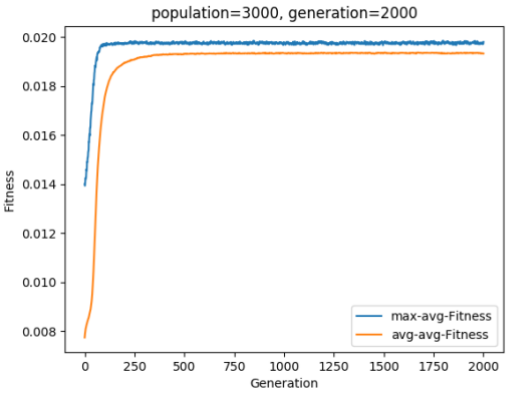

population=2000, generation=3000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

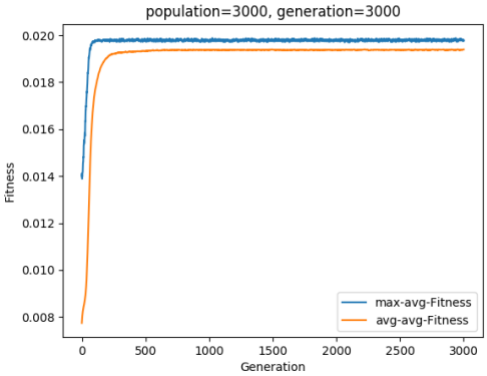

population=2000, generation=1000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

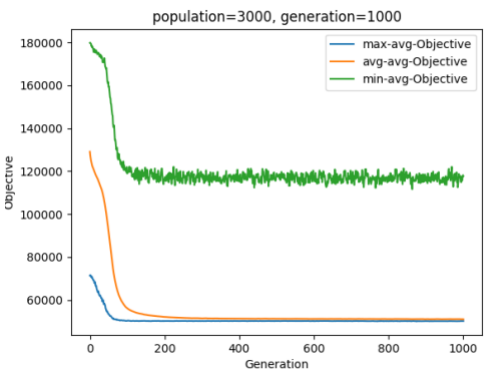

population=2000, generation=2000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

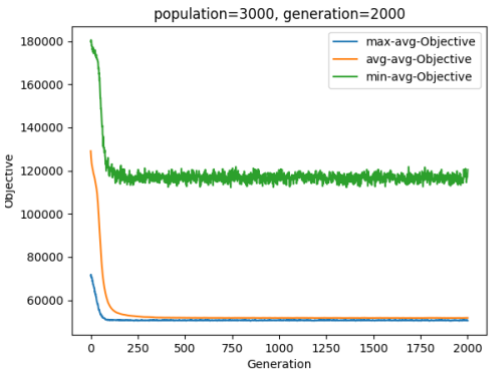

population=2000, generation=3000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

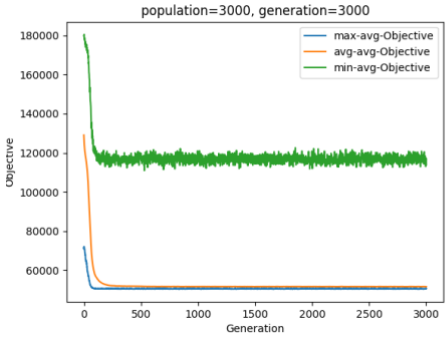

population=3000, generation=1000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=3000, generation=2000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=3000, generation=3000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

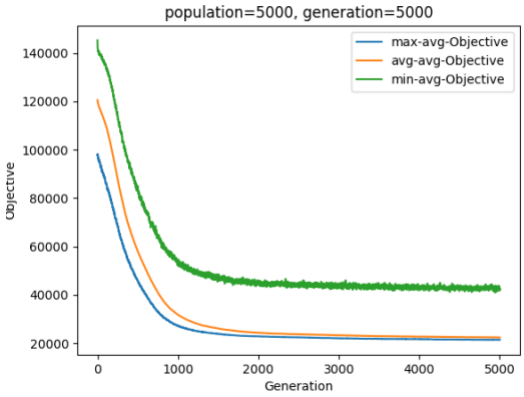

population=3000, generation=1000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

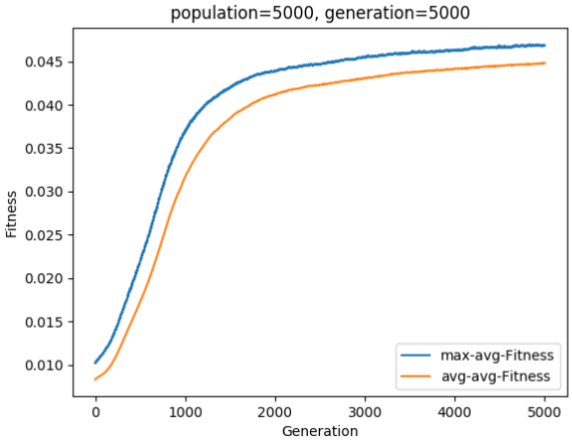

population=3000, generation=2000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

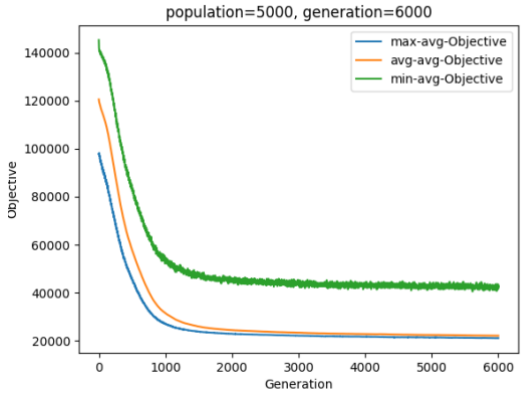

population=3000, generation=3000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

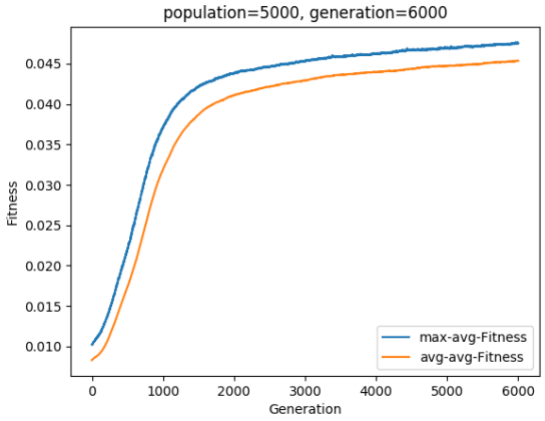
Lin105

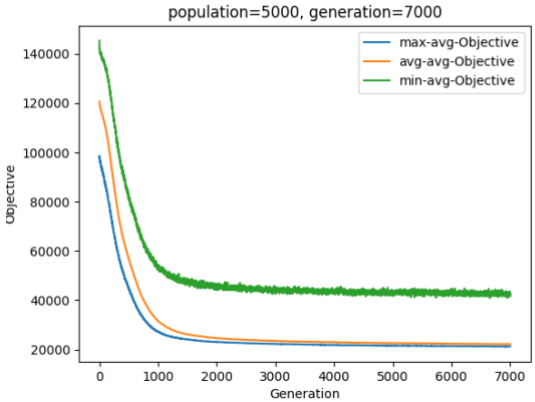
Quality	Reliability	Speed
44%	86%	11192692
45%	100%	14753000
45%	83%	12362400
46%	90%	12812962
47%	93%	14885357
50%	86%	13449692
51%	83%	15484080
51%	93%	20630500
52%	83%	15112160

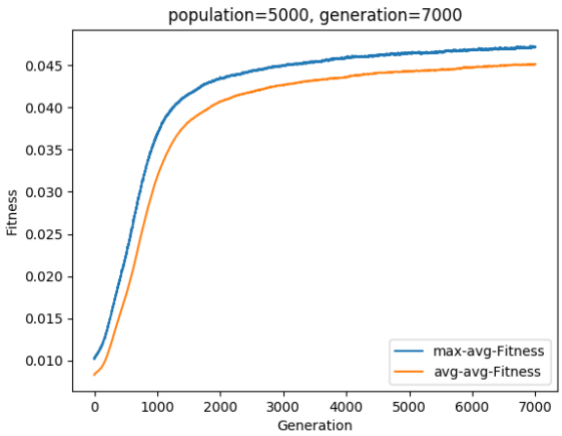


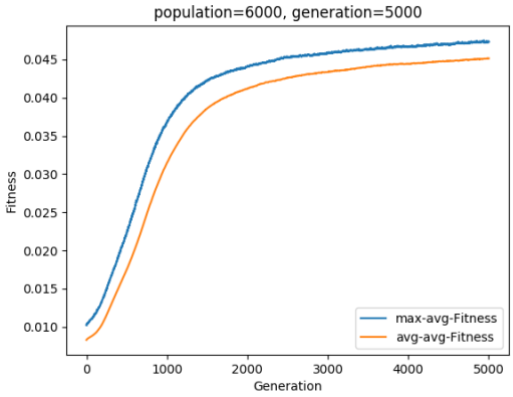


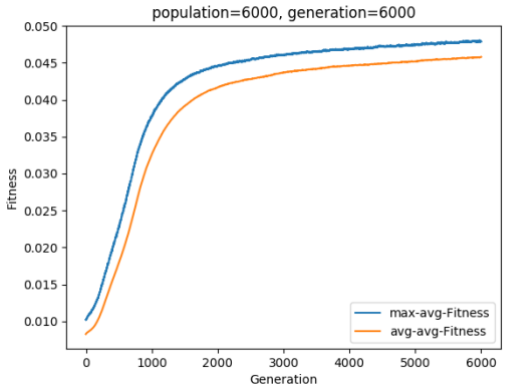


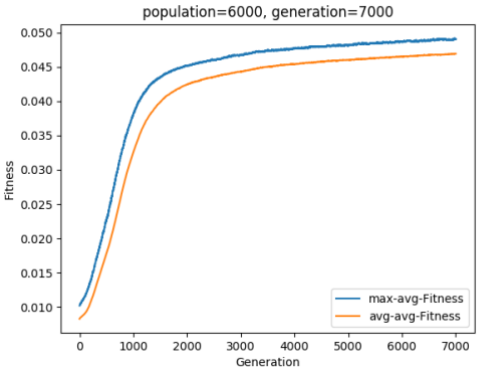


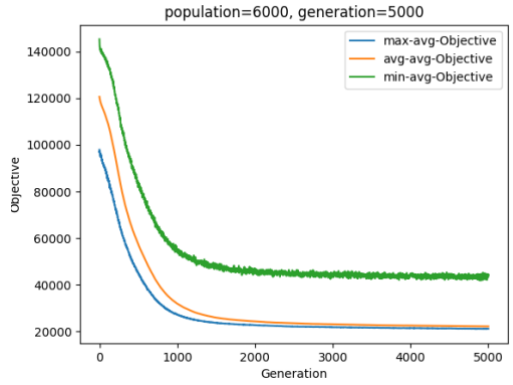


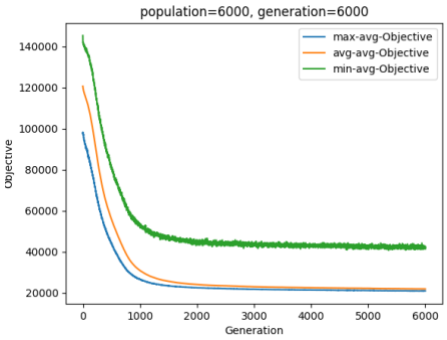


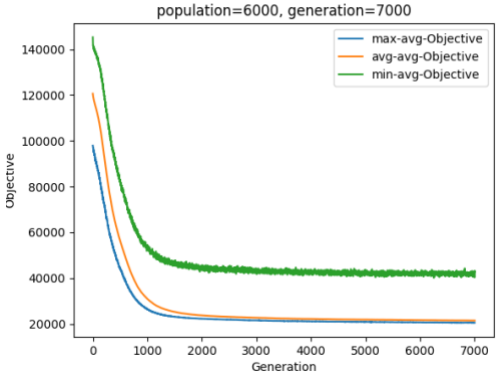

population=6000, generation=5000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

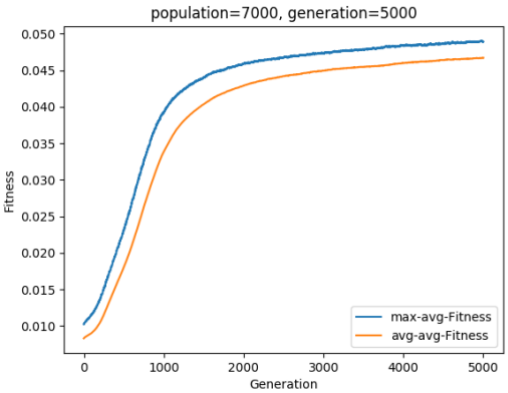

population=6000, generation=6000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

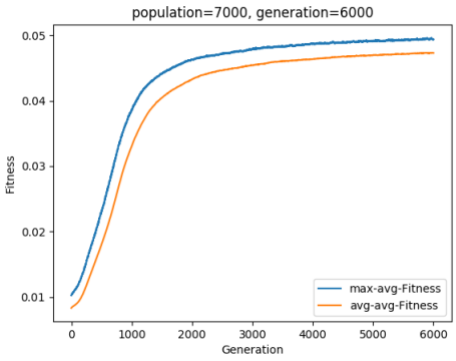

population=6000, generation=7000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

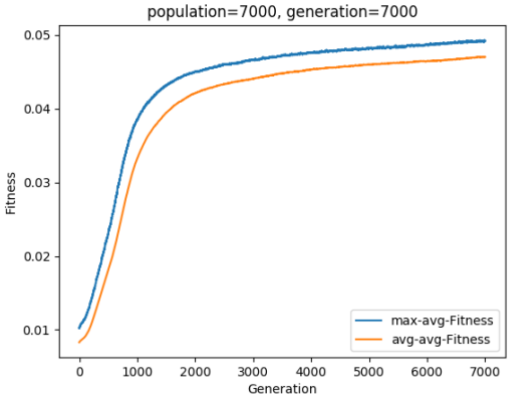

population=6000, generation=5000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

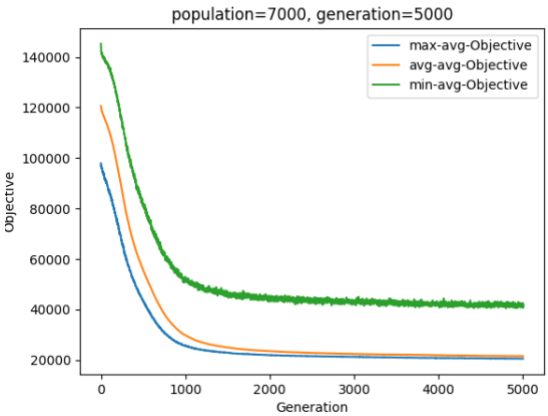

population=6000, generation=6000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

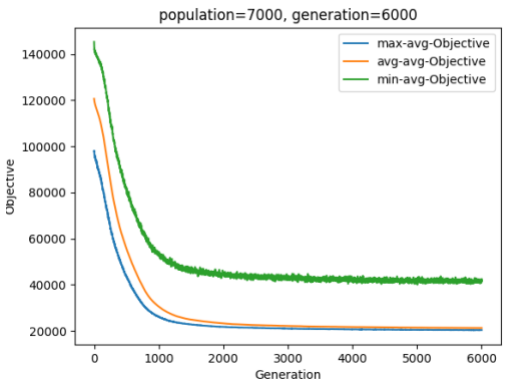

population=6000, generation=7000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

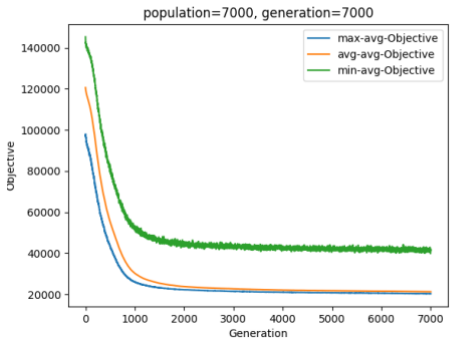

population=7000, generation=5000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=7000, generation=6000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=7000, generation=7000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=7000, generation=5000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

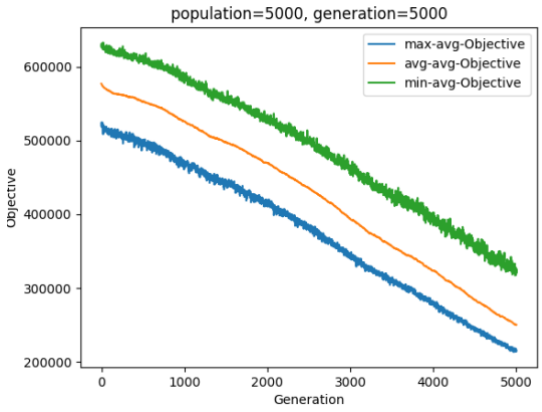

population=7000, generation=6000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

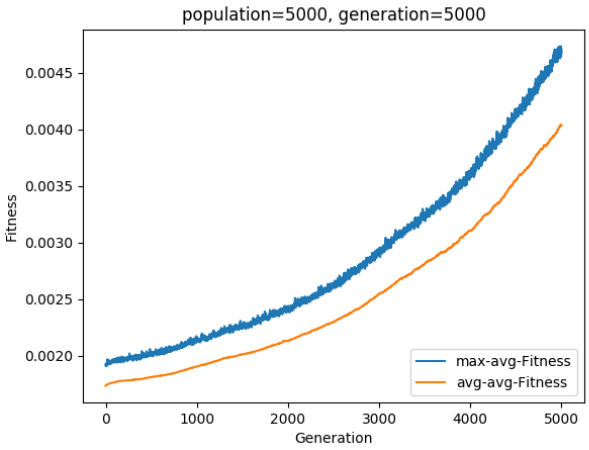

population=7000, generation=7000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

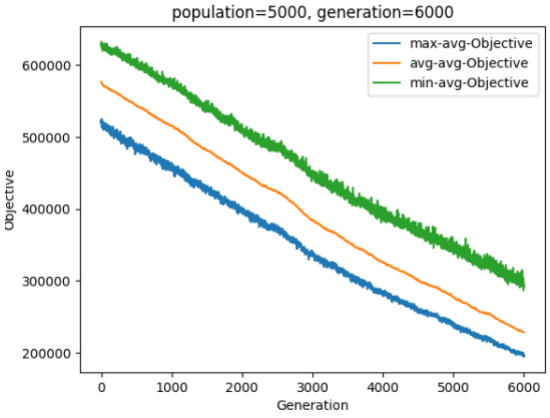
Lin318

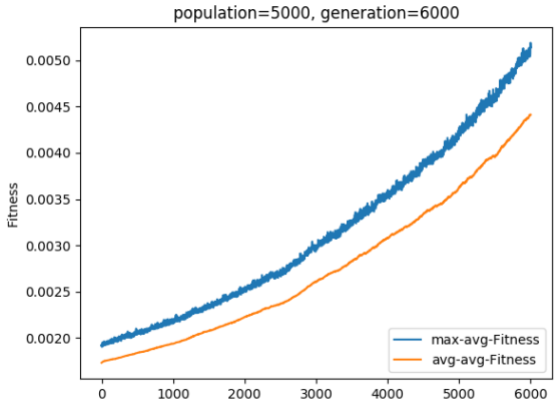
For this problem I did not put quality because it was really low. Instead I showed Tour that I obtained and optimal tour from website. As you can see my tour is considerably huge compared to known optimal value. The reason is I needed some more time as you can see from graphs they are not converging. If I hadesome more time maybe my function would converge and get optimal value.

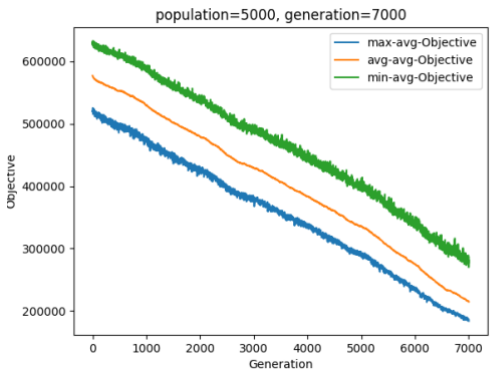
Tour	OptimalTour	Reliability	Speed
250192	42029	73%	22001846
228450	42029	64%	26847009
214694	42029	75%	33090383
236815	42029	70%	27000282
209021	42029	70%	35000657
157767	42029	77%	40006219
264653	42029	80%	33036180
169558	42029	60%	41102720
136034	42029	72%	48030009

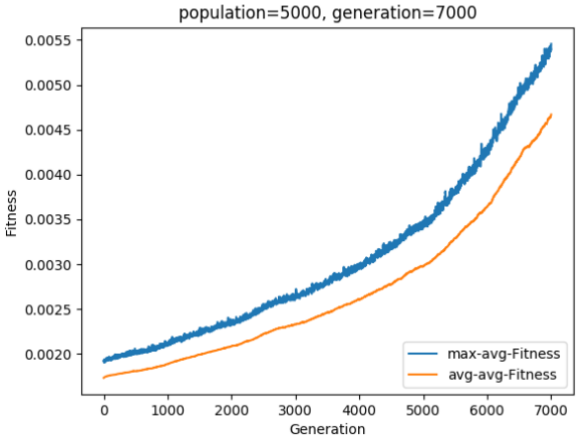


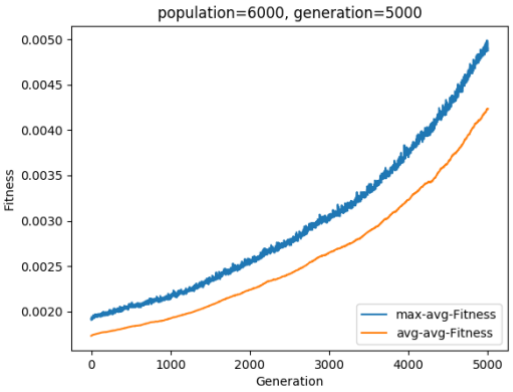


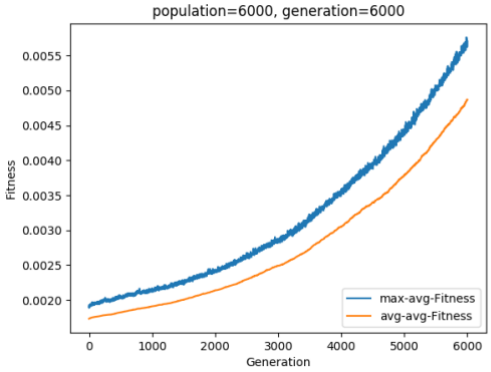


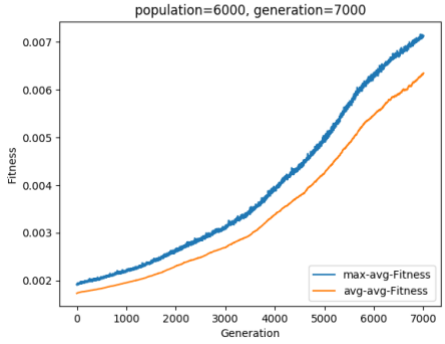


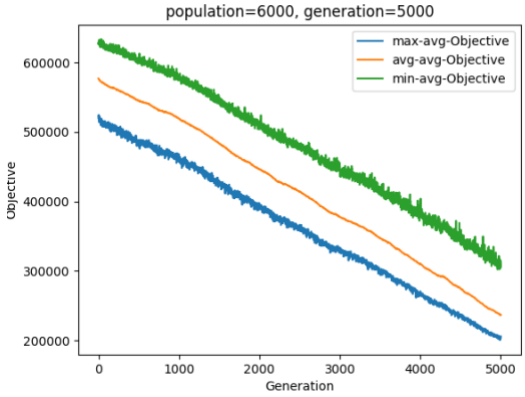


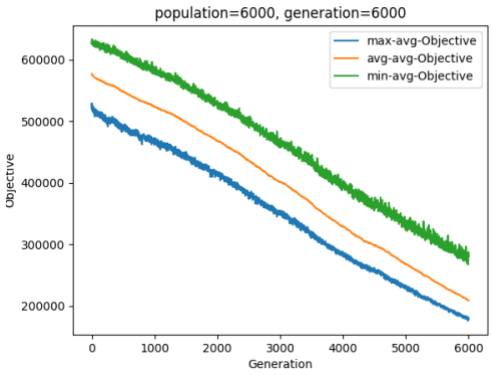


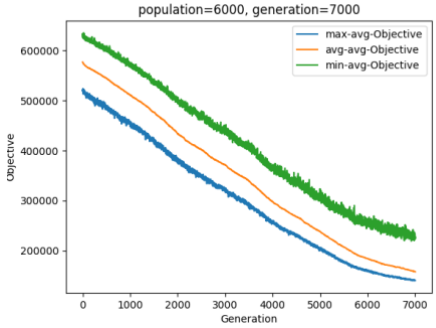

population=6000, generation=5000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

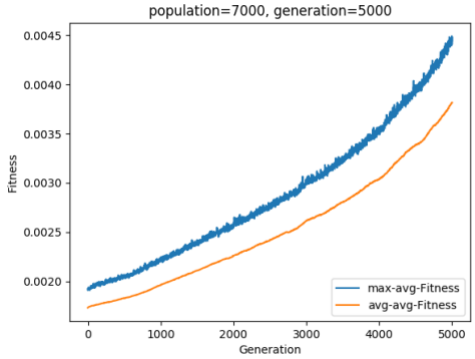

population=6000, generation=6000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

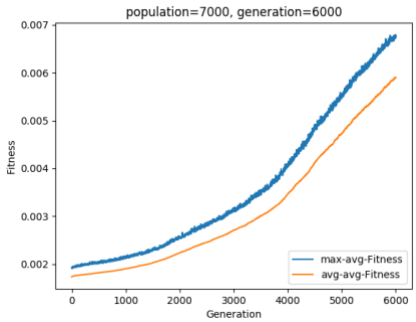

population=6000, generation=7000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness

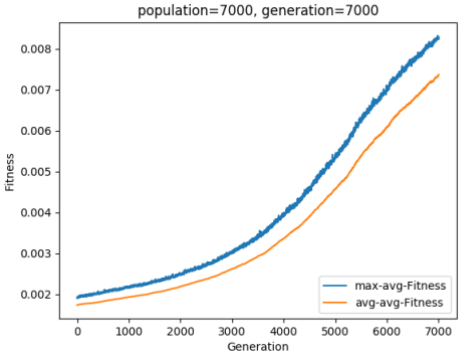

population=6000, generation=5000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

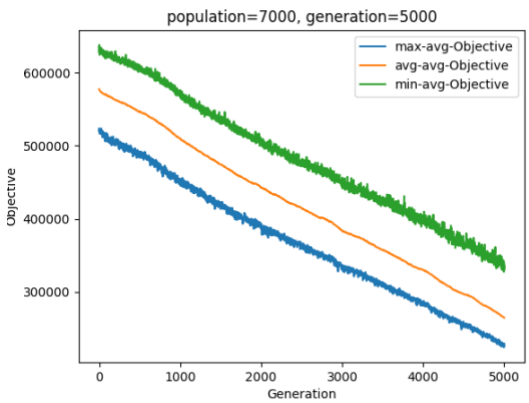

population=6000, generation=6000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

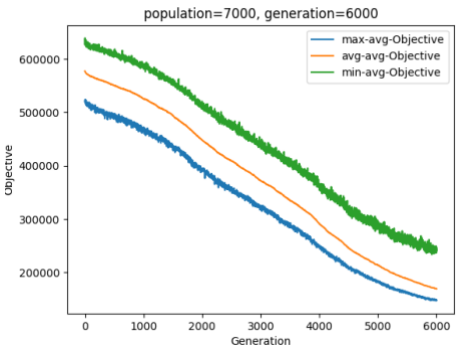

population=6000, generation=7000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

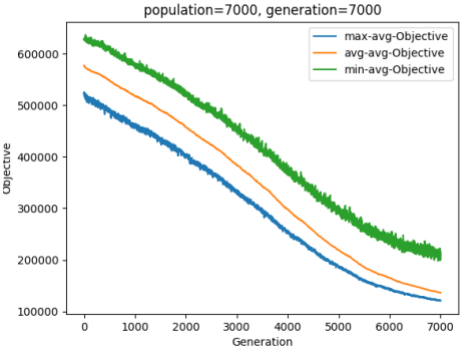

population=7000, generation=5000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=7000, generation=6000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=7000, generation=7000
Fitness
Generation
max-avg-Fitness
avg-avg-Fitness


population=7000, generation=5000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective


population=7000, generation=6000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective


population=7000, generation=7000
Objective
Generation
max-avg-Objective
avg-avg-Objective
min-avg-Objective

References:

[1] “Genetic Algorithms Repository” available at https://www.cse.unr.edu/~sushil/class/gas/code/index.html

[2] Gokturk Ucoluk “Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation”

[3] John McCullock “Genetic Algorithms, Mutation, Inversion mutation” available at http://mnemstudio.org/genetic-algorithms-mutation.htm

[4] Lee Jacobson “Applying a genetic algorithm to the traveling salesman problem” available at http://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5