Universidade de São Paulo
Instituto de Física de São Carlos

Mathematical-Computational Modeling

Genetic Algorithm

Éverton Luís Mendes da Silva (10728171)

1 Introduction

This project aims to solve the traveling salesman problem using the genetic algorithm. For this, several operators of genetic modification were considered, aiming thus, with the intuition of understanding its importance for minimizing the distance covered.

2 Cities

For analysis, points randomly distributed within a circle of radius 1 were chosen. Thus, each point represents a city to be covered by the traveler.

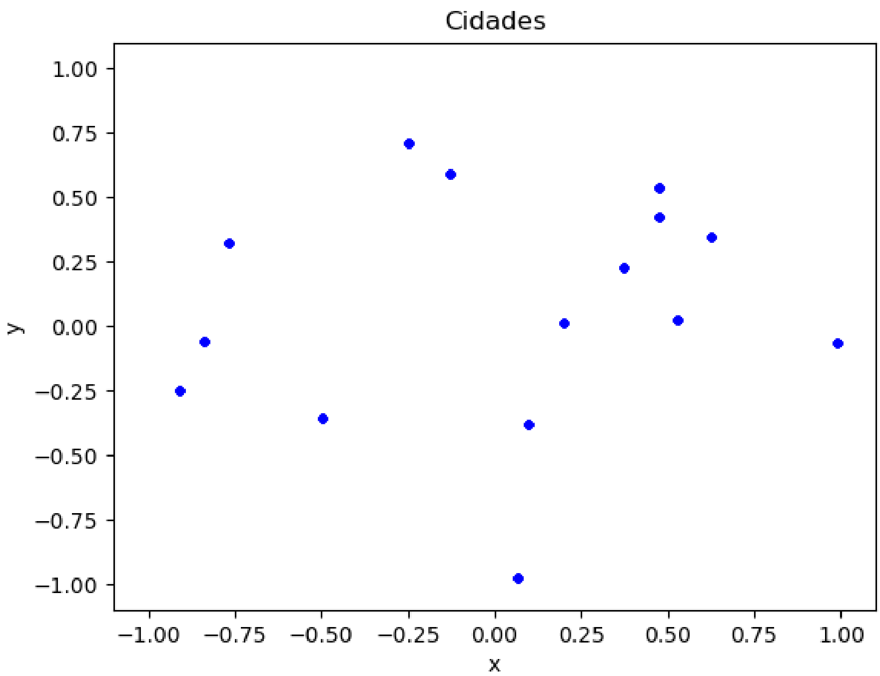
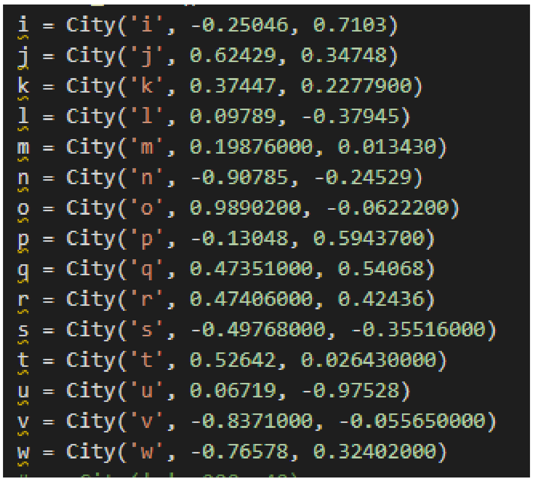


Figure 1

For each city a letter of the alphabet was assigned for identification.



```
i = City('i', -0.25046, 0.7103)
j = City('j', 0.62429, 0.34748)
k = City('k', 0.37447, 0.2277900)
l = City('l', 0.09789, -0.37945)
m = City('m', 0.19876000, 0.013430)
n = City('n', -0.90785, -0.24529)
o = City('o', 0.9890200, -0.0622200)
p = City('p', -0.13048, 0.5943700)
q = City('q', 0.47351000, 0.54068)
r = City('r', 0.47406000, 0.42436)
s = City('s', -0.49768000, -0.35516000)
t = City('t', 0.52642, 0.026430000)
u = City('u', 0.06719, -0.97528)
v = City('v', -0.8371000, -0.055650000)
w = City('w', -0.76578, 0.32402000)
```

Figure 2

3 Types of genetic modification

The genes were modified from some operators explained in the CDT-38. These operators are 'mutation', 'inversion', 'transposition', 'crossing-over'. Such operators are presented in the codes below, respectively:

```python
def mutate(self, route_to_mut):
    '''
    Route() --> Route()
    Swaps two random indexes in route_to_mut.route. Runs k_mut_prob of the time
    '''
    # k_mut_prob %
    if random.random() < m_mut_prob:

        # two random indices:
        mut_pos1 = random.randint(0,len(route_to_mut.route)-1)
        mut_pos2 = random.randint(0,len(route_to_mut.route)-1)

        # if they're the same, skip to the chase
        if mut_pos1 == mut_pos2:
            return route_to_mut

        # Otherwise swap them:
        city1 = route_to_mut.route[mut_pos1]
        city2 = route_to_mut.route[mut_pos2]

        route_to_mut.route[mut_pos2] = city1
        route_to_mut.route[mut_pos1] = city2

    # Recalculate the length of the route (updates it's .length)
    route_to_mut.recalc_rt_len()

    return route_to_mut
```

Figure 3: Mutation

```python
def inversion(self, route_to_mut):


    if random.random() < i_mut_prob:

        # two random indices:
        mut_pos1 = random.randint(0,len(route_to_mut.route)-1)
        mut_pos2 = random.randint(0,len(route_to_mut.route)-1)

        # if they're the same, skip to the chase
        if mut_pos1 == mut_pos2:
            return route_to_mut

        route_to_mut.route[mut_pos1:mut_pos2] = route_to_mut.route[mut_pos1:mut_pos2][::-1]

    return route_to_mut
```

Figure 4: inversion

```python
def transposition(self, route_to_mut):

    if random.random() < t_mut_prob:

        n= random.randint(0,(len(route_to_mut.route))//2)

        # two random indices:
        mut_pos1 = random.randint(0,(len(route_to_mut.route)-1)//2-n)
        mut_pos2 = random.randint((len(route_to_mut.route)-1)//2,len(route_to_mut.route)-1-n)

        # if they're the same, skip to the chase
        if mut_pos1 == mut_pos2:
            return route_to_mut


        slice1 = route_to_mut.route[mut_pos1:mut_pos1+n]
        slice2 = route_to_mut.route[mut_pos2: mut_pos2+n]


        route_to_mut.route[mut_pos1:mut_pos1+n]= slice2[:]
        route_to_mut.route[mut_pos2:mut_pos2+n]= slice1[:]


        return route_to_mut
```

Figure 5: transposition

```python
def crossover(self, parent1, parent2):
    '''
    Route(), Route() --> Route()
    Returns a child route Route() after breeding the two parent routes.
    Routes must be of same length.
    Breeding is done by selecting a random range of parent1, and placing it into the empty child route (in th
    Gaps are then filled in, without duplicates, in the order they appear in parent2.

    '''
    # new child Route()
    child_rt = Route()

    for x in range(0,len(child_rt.route)):
        child_rt.route[x] = None

    # Two random integer indices of the parent1:
    start_pos = random.randint(0,len(parent1.route))
    end_pos = random.randint(0,len(parent1.route))

    #### takes the sub-route from parent one and sticks it in itself:
    # if the start position is before the end:
    if start_pos < end_pos:
        # do it in the start-->end order
        for x in range(start_pos,end_pos):
            child_rt.route[x] = parent1.route[x] # set the values to eachother
    # if the start position is after the end:
    elif start_pos > end_pos:
        # do it in the end-->start order
        for i in range(end_pos,start_pos):
            child_rt.route[i] = parent1.route[i] # set the values to eachother
```

Figure 6: Crossing-over 1

```python
    # Cycles through the parent2. And fills in the child_rt
    # cycles through length of parent2:
    for i in range(len(parent2.route)):
        # if parent2 has a city that the child doesn't have yet:
        if not parent2.route[i] in child_rt.route:
            # it puts it in the first 'None' spot and breaks out of the loop.
            for x in range(len(child_rt.route)):
                if child_rt.route[x] == None:
                    child_rt.route[x] = parent2.route[i]
                    break
    # repeated until all the cities are in the child route

    # returns the child route (of type Route())
    child_rt.recalc_rt_len()
    return child_rt
```

Figure 7: Crossing-over 2

4 Genetic Algorithm

In order to find the best parameters for the traveling salesman problem, several values were tested for this problem. These sets will be handled by the representations below.

$$\text{Parameters} \rightarrow \Big\{ \phi_n,\ n = 1, 2, ..., 8$$

4.1 ϕ_1

$$\rightarrow \begin{cases} Number\ of\ generations = 100 \\ Number\ of\ population = 100 \\ Prob.\ of\ Mutation = 0.5 \\ Prob.\ of\ Inversion = 0.5 \\ Prob.\ of\ Transposition = 0.1 \\ Prob.\ of\ Crossing - over = 0.5 \end{cases}$$

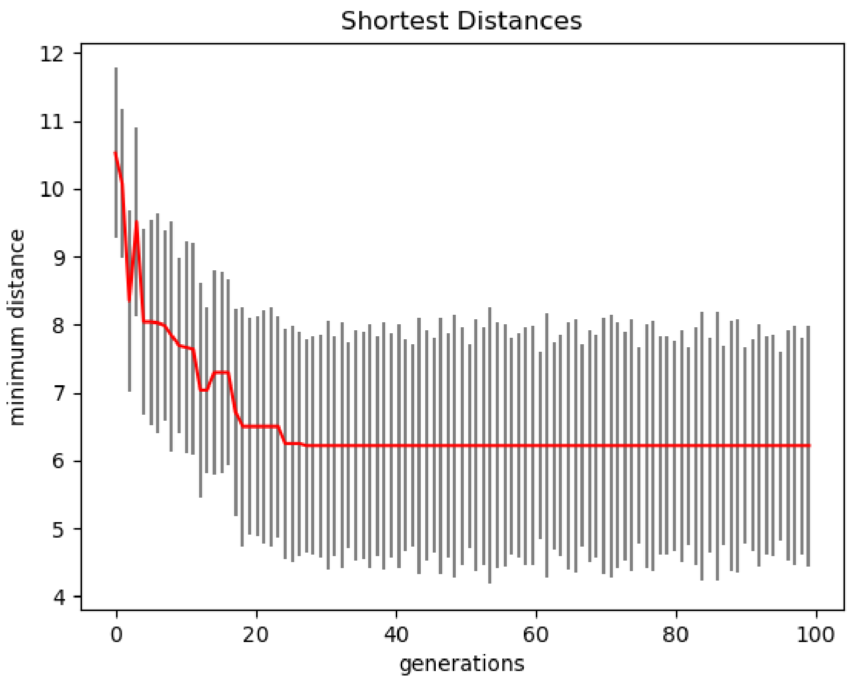


Figure 8

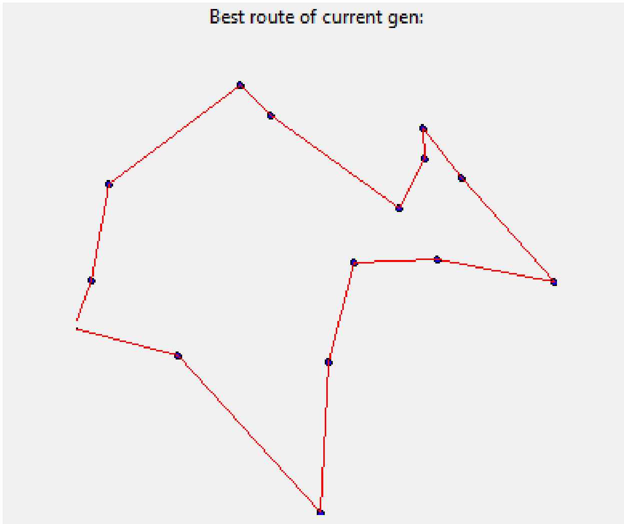


Figure 9

4.2 ϕ_2

$$\rightarrow \begin{cases} Number\ of\ generations = 100 \\ Number\ of\ population = 50 \\ Prob.\ of\ Mutation = 0.3 \\ Prob.\ of\ Inversion = 0.5 \\ Prob.\ of\ Transposition = 0.4 \\ Prob.\ of\ Crossing-over = 0.1 \end{cases}$$

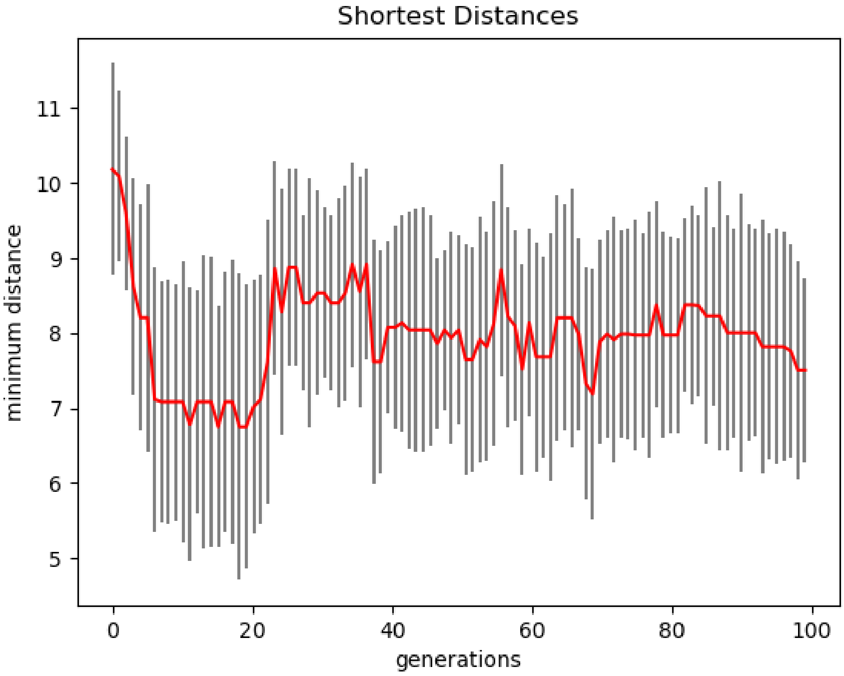


Figure 10

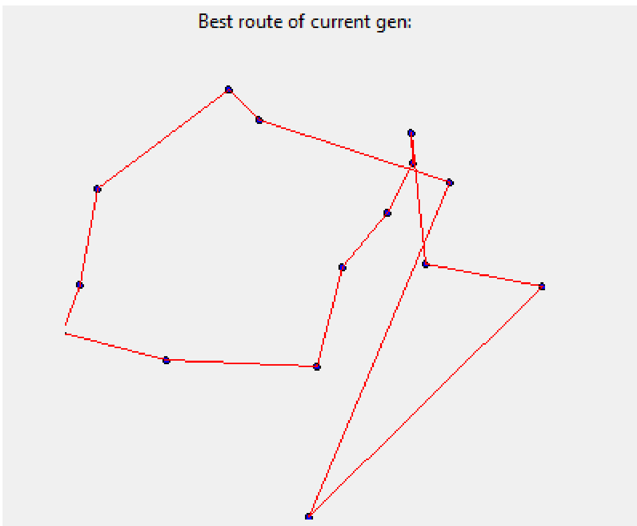


Figure 11

4.3 ϕ_3

$$\rightarrow \begin{cases} Number\ of\ generations = 100 \\ Number\ of\ population = 200 \\ Prob.\ of\ Mutation = 0.3 \\ Prob.\ of\ Inversion = 0.5 \\ Prob.\ of\ Transposition = 0.4 \\ Prob.\ of\ Crossing - over = 0.1 \end{cases}$$

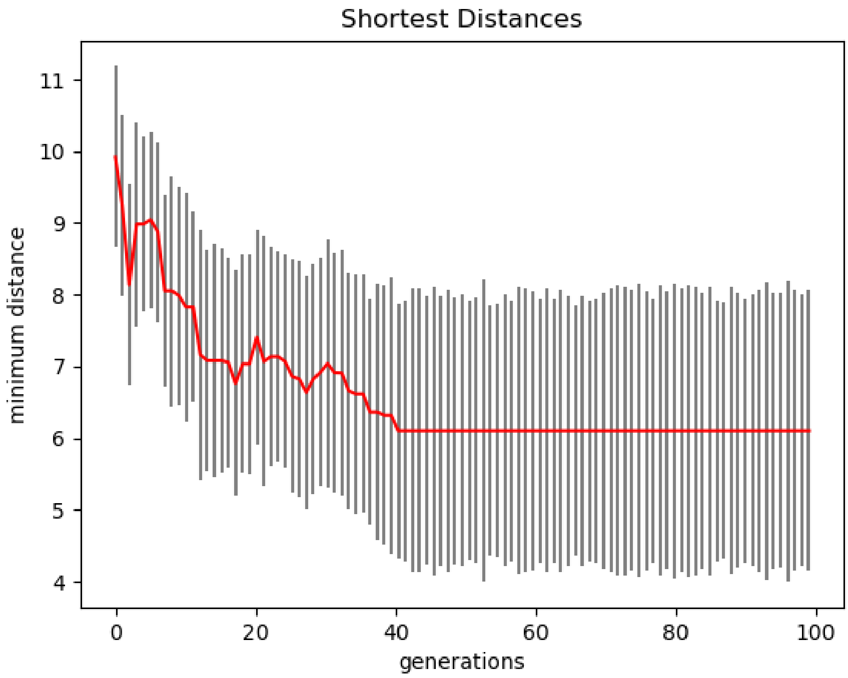


Figure 12

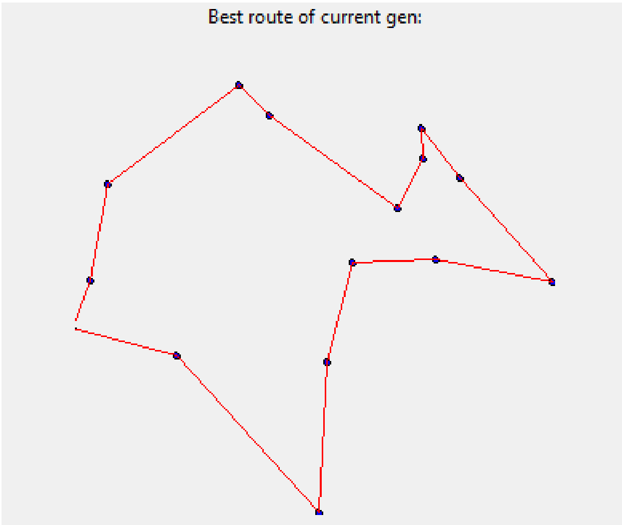


Figure 13

4.4 ϕ_4

$$\rightarrow \begin{cases} Number\ of\ generations = 50 \\ Number\ of\ population = 200 \\ Prob.\ of\ Mutation = 0.3 \\ Prob.\ of\ Inversion = 0.3 \\ Prob.\ of\ Transposition = 0.2 \\ Prob.\ of\ Crossing - over = 0.3 \end{cases}$$

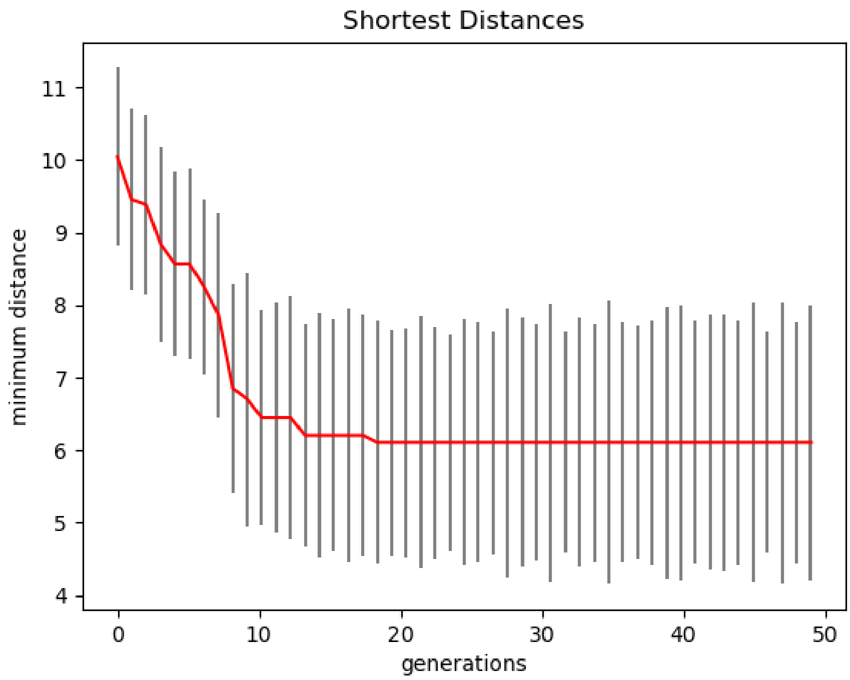


Figure 14

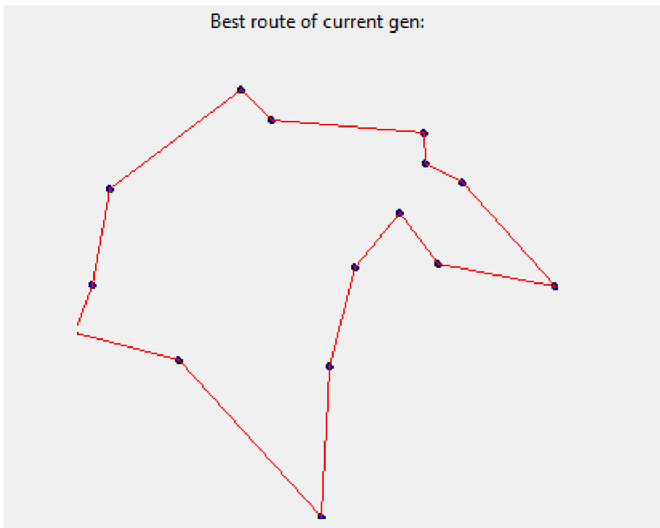


Figure 15

4.5 ϕ_5

$$\rightarrow \begin{cases} Number\ of\ generations = 100 \\ Number\ of\ population = 200 \\ Prob.\ of\ Mutation = 0.1 \\ Prob.\ of\ Inversion = 0.7 \\ Prob.\ of\ Transposition = 0.1 \\ Prob.\ of\ Crossing - over = 0.3 \end{cases}$$

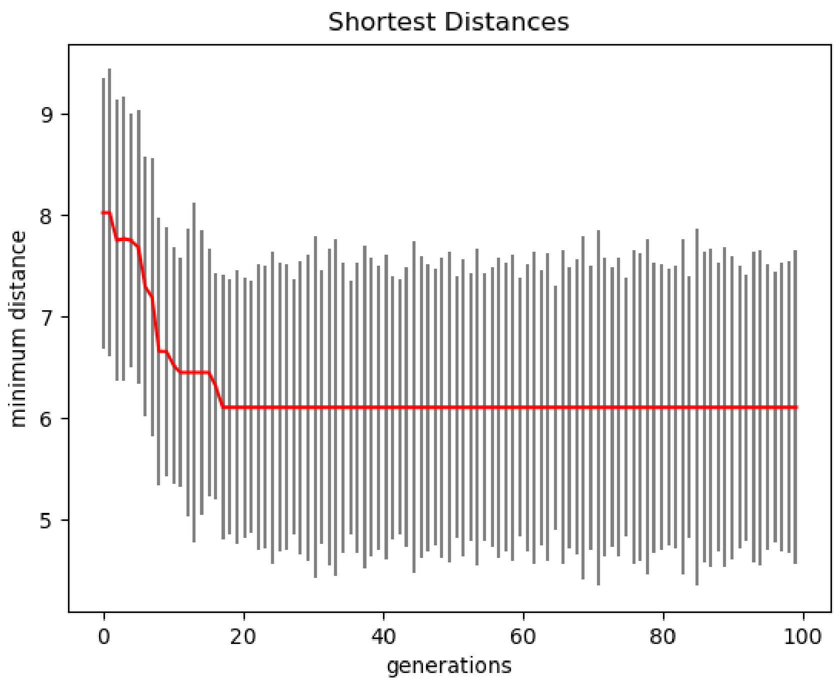


Figure 16

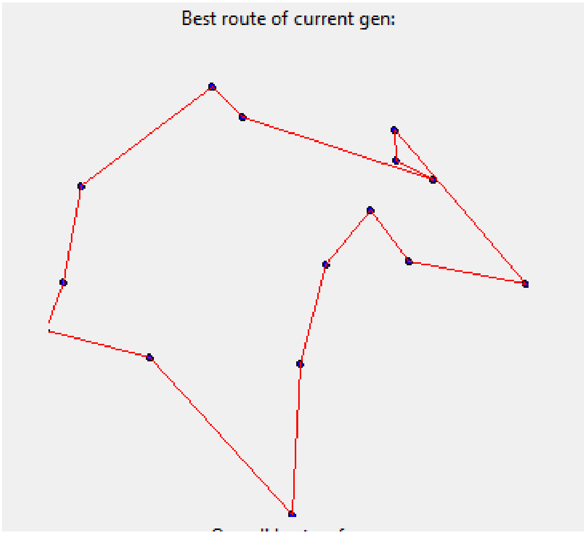


Figure 17

4.6 ϕ_6

$$\rightarrow \begin{cases} Number\ of\ generations = 100 \\ Number\ of\ population = 300 \\ Prob.\ of\ Mutation = 0.1 \\ Prob.\ of\ Inversion = 0.2 \\ Prob.\ of\ Transposition = 0.8 \\ Prob.\ of\ Crossing-over = 0.1 \end{cases}$$

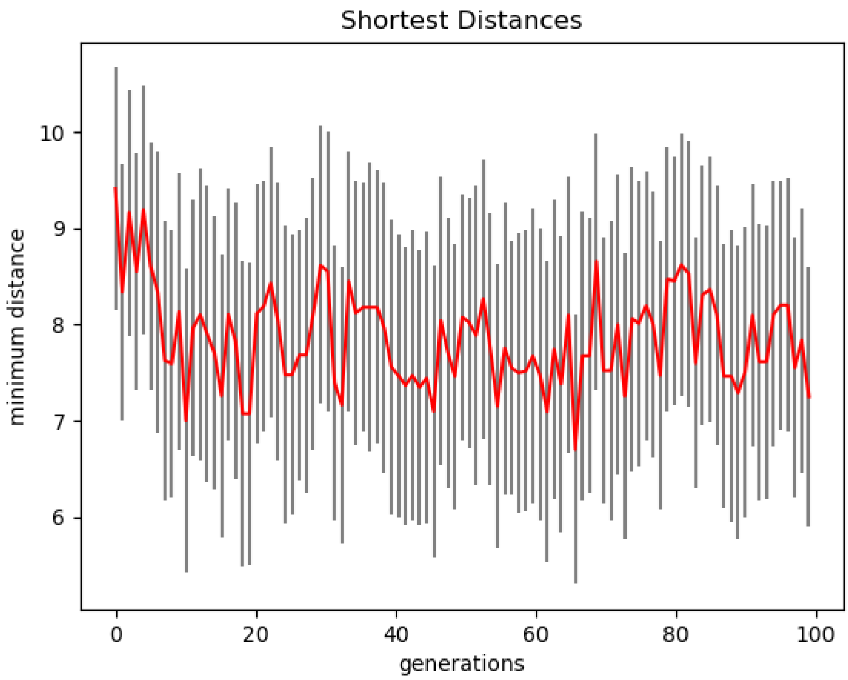


Figure 18

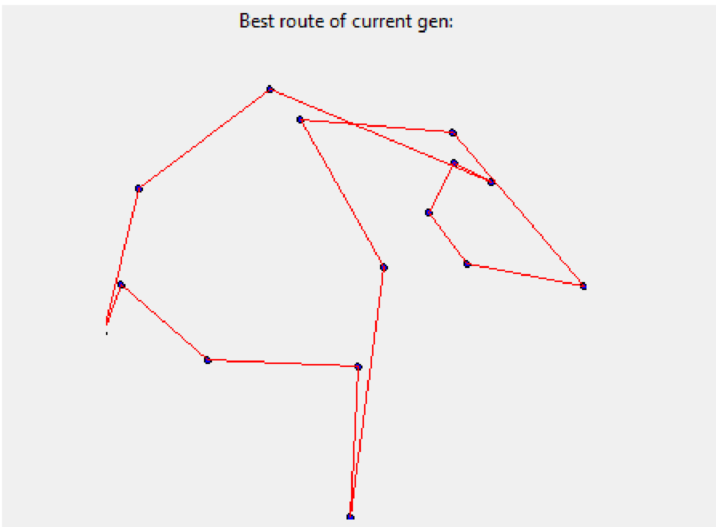


Figure 19

4.7 ϕ_7

$$\rightarrow \begin{cases} Number\ of\ generations = 100 \\ Number\ of\ population = 300 \\ Prob.\ of\ Mutation = 0.1 \\ Prob.\ of\ Inversion = 0.2 \\ Prob.\ of\ Transposition = 0.1 \\ Prob.\ of\ Crossing - over = 0.8 \end{cases}$$

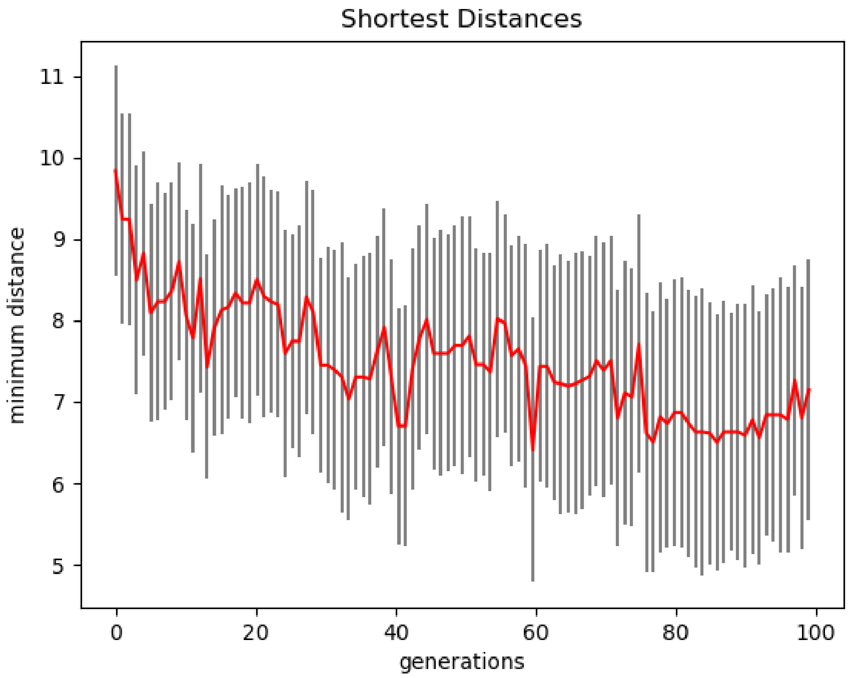


Figure 20

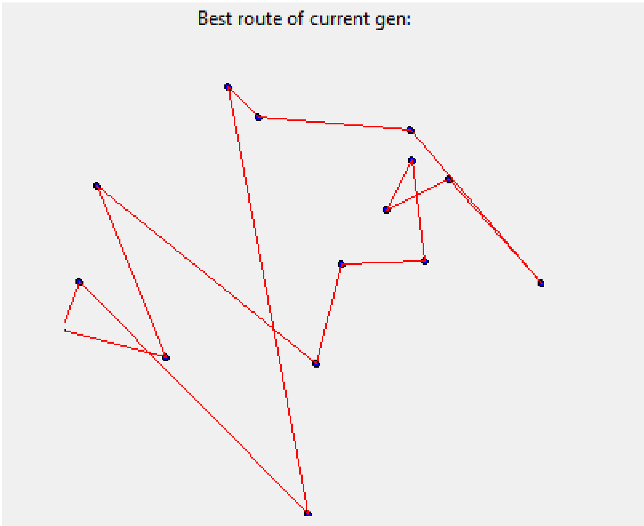


Figure 21

4.8 ϕ_8

$$\rightarrow \begin{cases} Number\ of\ generations = 100 \\ Number\ of\ population = 300 \\ Prob.\ of\ Mutation = 0.8 \\ Prob.\ of\ Inversion = 0.8 \\ Prob.\ of\ Transposition = 0.1 \\ Prob.\ of\ Crossing - over = 0.1 \end{cases}$$

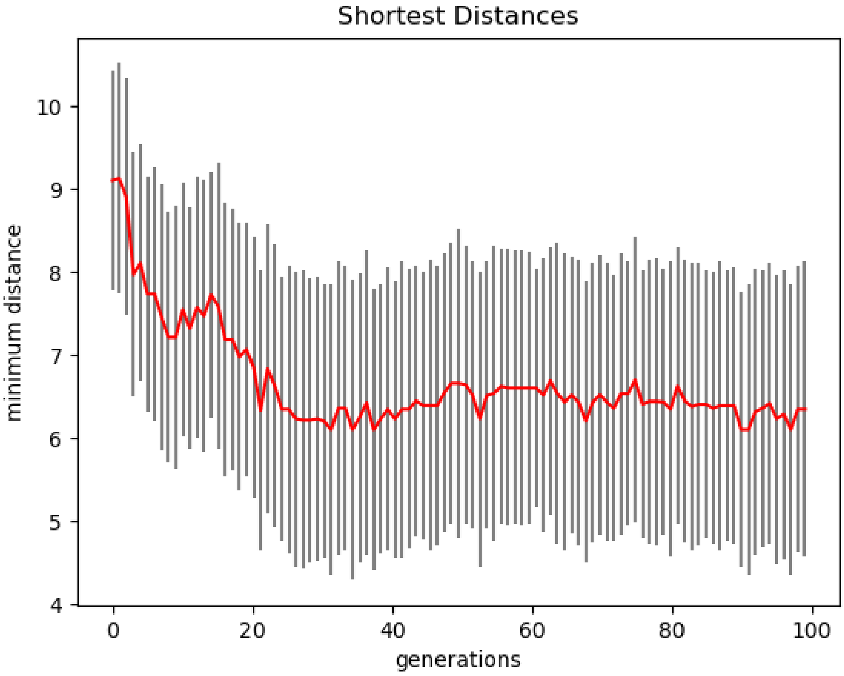


Figure 22

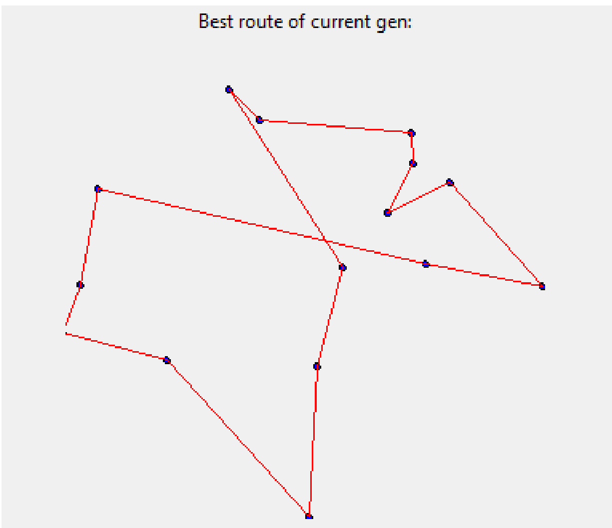


Figure 23

5 Conclusion

From the computational experiments performed, it becomes evident that operators such as 'transposition' and 'crossing-over' cannot be highly probable, since they involve the change of many crossomes. On the other hand, 'inversion' and 'mutation' even with their high probabilities over time tend to be the shortest path (as if they were a noise).

6 References

[1] da Silva, Éverton Luís Mendes. Codes and Images used in this project can be founded in my GItHub. ¡https://github.com/evertonmendes/Mathematical-Computational-Modeling¿