NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Computational Intelligence for Optimization

Final Project

Alex Anthony Panchot (M20190546)
Bruno de Lima Vieira (M20190922)
Hugo Saisse Mentzingen da Silva (M20190215)
Leonardo Motta Perazzo Lannes (M20180036)

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Table of Contents

1. Introduction

This report addresses the final project in the Computational Intelligence for Optimization course of Master Degree Program in Data Science and Advanced Analytics of Nova IMS. The groups of students received a genetic algorithm framework and were supposed to develop the algorithms needed to solve Travel Salesman Problem (TSP) and Portfolio Investment Problem (PIP). The detailed guidelines are in the Project Description document.

Therefore, we present in the following pages the rational and the algorithms applied to achieve the best possible fitness for these problems, respecting the established constraints. This report is accompanied by the files on our Github page.

2. Travel Salesman Problem (TSP)

Given a distance matrix containing the distances, taken pairwise, between n cities, the objective of this problem is to construct the shortest tour that visits each destination exactly once.

2.1. Encoding

We will be using the most natural and commonly used way to represent the TSP problem: the i-th element of the representation denotes the i-th destination visited.

For instance, suppose that the following distance matrix has been provided:

```
Matrix = [[0, 15, 3, 4],
          [15, 0, 8, 7],
          [3, 8, 0, 12],
          [4, 7, 12, 0]]
```

This example shows us the distances between 4 cities. The first city will be represented by “0”, the second one by “1”, and so on. Thus, the last city will be called “n-1”.

Moreover, the encoding is defined by an n-size list in Python, in which the tour City 2 → City 4 → City 3 → City 1 is given by [1,3,2,0].

2.2. Initializations

We have implemented 3 different sorts of initialization:

- **Randomly**: this kind of initialization only generates a permutation from [0,1,2,3,…,n-2, n-1], e.g, [1,0,2,3,…,n-2, n-1]. Since we have been using a population with 20 members, the initial population is given by 20 permutations randomly generated.

- **Using Hill Climbing**: the first stage of this method is to generate a population randomly as with the previous one. The main difference is that the first generation will be the outcome which has been optimized by the hill-climbing algorithm.

- **N-Queens**: This initialization method works by the same principle as the N-Queen problem. The departure city is imagined as the row of a matrix and the arrival city as the column of the matrix. The algorithm then “places” a “queen” on a selected row and column and blocks out all other points in that particular row and column such that there cannot be two “queens” on

the same row or column. The algorithm always starts in the first city and selects the second by choosing an available city with the lowest distance. Then the algorithm moves to the row of the chosen city. So if the second column is the lowest in the first row, the algorithm moves to the second row to pick the next city. This ensures that the departure city is the same as the arrival city. As cities are selected, their column (and row) is made unavailable to the algorithm such that it can only pick cities that have not been previously selected.
In order to pick different combinations for an initial population, the algorithm will sometimes select the second best city in order to leave the best city available for selection later.

2.3. Mutations

We have implemented 4 mutation operators, which are:

- **Swap mutation**: Two positions (genes) in the chromosome are selected at random and their allele values swapped.

- **Insert mutation**: Two alleles are selected at random and the second moved next to the first, shuffling along the others to make room.

- **Scramble mutation**: Here the entire chromosome, or some randomly chosen subset of values within it, have their positions scrambled.

- **Inversion mutation**: this works by randomly selecting two positions in the chromosome and reversing the order in which the values appear between those positions.

2.4. Crossovers

We have implemented 5 recombination operators, which are:

- **Partially Mapped Crossover (PMX)**: the same explained in the theoretical class.

- **Order Crossover**: the same explained in the theoretical class.

- **Cycle Crossover:** the same explained in the theoretical class.

- **Edge Crossover**: this algorithm can be described by the following steps:

    - Pick two parents at random

    - Construct the edge table: this table contains, for each element, all the other elements that are linked to it in the two parents. A '+' in the table indicates that the edge is present in both parents (edges that have a '+' are called 'common edges').

Parent 1:	1	2	3	4	5	6	7	8	9
Parent 2:	9	3	7	8	2	6	5	1	4

Element	Edges	Element	Edges
1	2,4,5,9	6	2,5+,7
2	1,4,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+,9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

- Pick an initial element at random and put it in the offspring. For example, we can choose randomly element 1.

  Offspring:

1								

- Remove all references to the current element from the table.

Element	Edges	Element	Edges
1	2,4,5,9	6	2,5+,7
2	4,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+,9
4	3,5,9	9	3,4,8
5	4,6+		

- Examine the list for the current element.
  - If there is a common edge, pick that to be the next element.
  - Otherwise, pick the entry in the list which itself has the shortest list.
  - Ties are split at random.

  In this case, since there is no common edge, we will use the shortest list criteria. Then, the selected element is 5, because it has length 2 in opposition to the others, which have length 3.

  Offspring:

1	5							

- Remove all references to 5 from the table.

Element	Edges	Element	Edges
		6	2,7
2	4,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+,9
4	3,9	9	3,4,8
5	4,6+		

- 6 is a common edge.

  Offspring:

1	5	6						

- Repeat in the same manner until the offspring is completed….
- At the end, the remaining values will look like this:

Element	Edges	Element	Edges
3	4,9		
4	9	9	4

- The options are 4 and 9 and both have 1 item on the list. Then, we will choose randomly. The selected element is 9.

  Offspring:

1	5	6	2	8	7	3	9	

- Remove all references to 9 from the table.

Element	Edges	Element	Edges
4		9	4

- 4 is the only and last option.

Offspring:

1	5	6	2	8	7	3	9	4

- As the algorithm only returns one offspring, it is run again with the same parents to produce a second offspring, which can be different from the first one since the code contains some random choices.

2.5. Admissibility

Since all recombination and mutation operators implemented for this problem generate feasible solutions (permutations), the admissibility function always returns True.

2.6. Elitisms

- Elitism 1 is keeping the single best value in the population. The best will replace the worst if none of the values are better than the current best value.
- Elitism 2 is keeping all of the current population's solutions that are better than the new population's best solution.
- Elitism 3 replaces the worst half of the new population with the best half of the current population.
- Elitism 4 iteratively compares n-th best solution of each population and replaces the new population's worst element with the current population's n-th. It stops when the n-th element of the current population is worse than the n-th element of the new population.

2.7. TSP Results

2.7.1. Round 1

In order to find the best set of parameters for each configuration, the other parameters are fixed with a constant value. The following parameters are fixed for all tests:

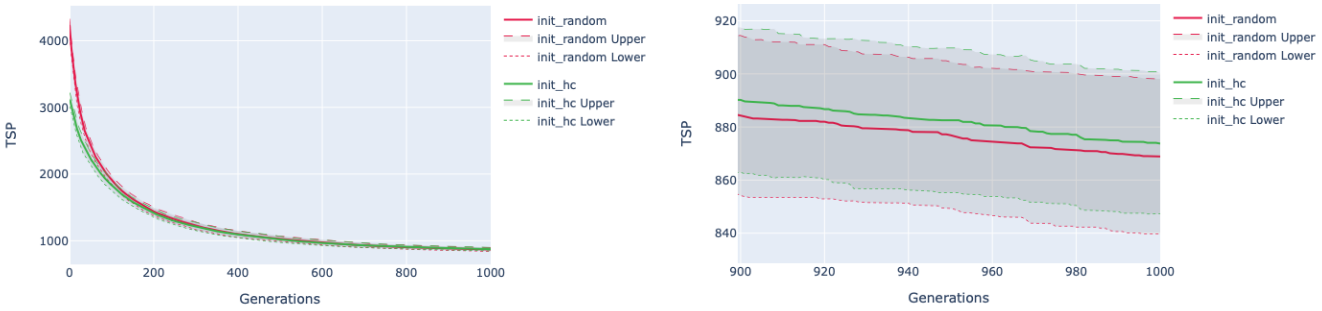
Crossover Rate: 0.6
Mutation Rate: 0.6
Tournament Size: 10

For the tournament selection, a size of 10 means that there are 10 participants in each “game”. Instead of running every possible combination of groups of participants, the algorithm selects 10 random parents (these parents can repeat) and creates a number of games equal to the population size. Therefore, the winner of each game automatically becomes a child. The tests are performed with the parameters outlined in Table 1, with the parameters being tested in **red** and the best parameter in **green**.

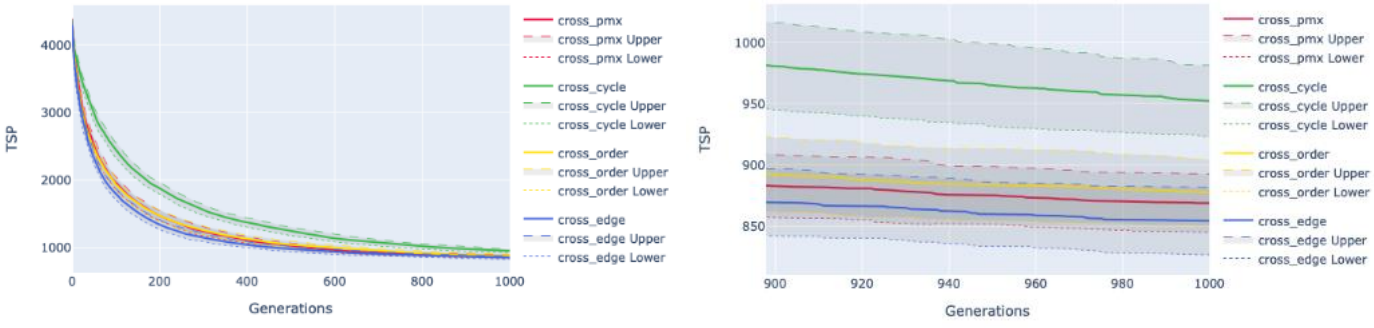
Table 1 - Round 1 Tests

Variation Number	Parameter Test	Initialization	Crossover	Mutation	Parente Selection	Replacement	Folder Name
1	Initialization	Random	Order	Inversion	Tournament (10)	Elitism	init_random
2	Initialization	Hill Climbing	Order	Inversion	Tournament (10)	Elitism	init_hc
3	Crossover	Random	PMX	Inversion	Tournament (10)	Elitism	cross_pmx
4	Crossover	Random	Cycle	Inversion	Tournament (10)	Elitism	cross_cycle
5	Crossover	Random	Order	Inversion	Tournament (10)	Elitism	cross_order
6	Crossover	Random	Edge	Inversion	Tournament (10)	Elitism	cross_edge
7	Mutation	Random	Order	Swap	Tournament (10)	Elitism	mut_swap
8	Mutation	Random	Order	Insert	Tournament (10)	Elitism	mut_insert
9	Mutation	Random	Order	Inversion	Tournament (10)	Elitism	mut_inversion
10	Mutation	Random	Order	Scramble	Tournament (10)	Elitism	mut_scramble
11	Parent Selection	Random	Order	Inversion	Roulette Wheel	Elitism	parent_roulette
12	Parent Selection	Random	Order	Inversion	Tournament (10)	Elitism	parent_tournament
13	Replacement	Random	Order	Inversion	Tournament (10)	Elitism	replace_elitism
14	Replacement	Random	Order	Inversion	Tournament (10)	Standard	replace_standard

Plotting these 5 tests shows the change in the performance of tuning a single parameter. For initialization, the hill-climbing performs better than random but as the number of generations increases the distance between the two is diminishes.

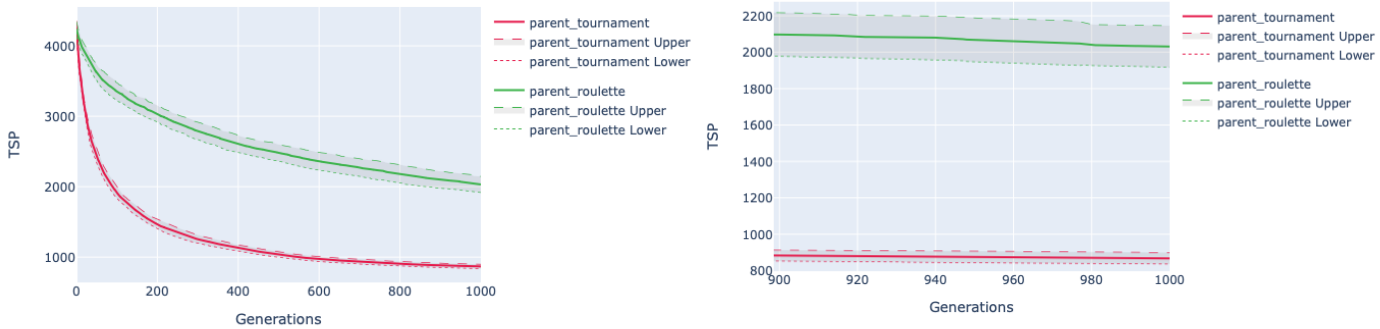


Next, the crossover (xover) types are compared with each other. Cycle xover is much worse than the other ones with edge xover being the best. One problem with edge is its runtime, with it performing only marginally better with a significant increase in runtime. This is a key tradeoff as one must determine whether time or performance is more important. If one has enough time for edge then running more generations could also be another possibility.

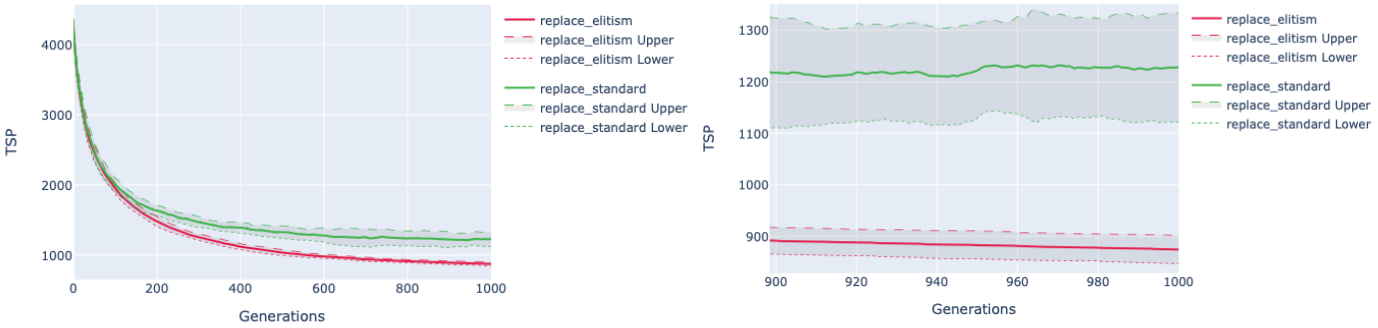


The mutations are a good example of how big a difference one parameter can make. The inversion and insert mutations are nearly identical, but swap and scramble are significantly worse. Scramble breaks too many points at once and therefore acts more as a xover than a mutation. Breaking the fewest number of existing connections is the most effective and the results for inversion show this.

The choice of parent selection is quite important as it picks which parents are of high enough quality to xover and mutate. The choice here is quite obvious as tournament converges much quicker than roulette. The range of final value is also minimized which means that any single run will be closer to the average.



The idea of elitism is also important to test as it decides whether or not to keep the best values between runs. As can be seen, not using elitism (standard) not only is worse than elitism but it also can increase (worsen) the fitness over time. The range of values produced by using it is also much larger.



2.7.2. Round 2

After the first round of selecting parameters, the values that were fixed before can now be tested. The best of each parameter as determined in the first round will be fixed here:

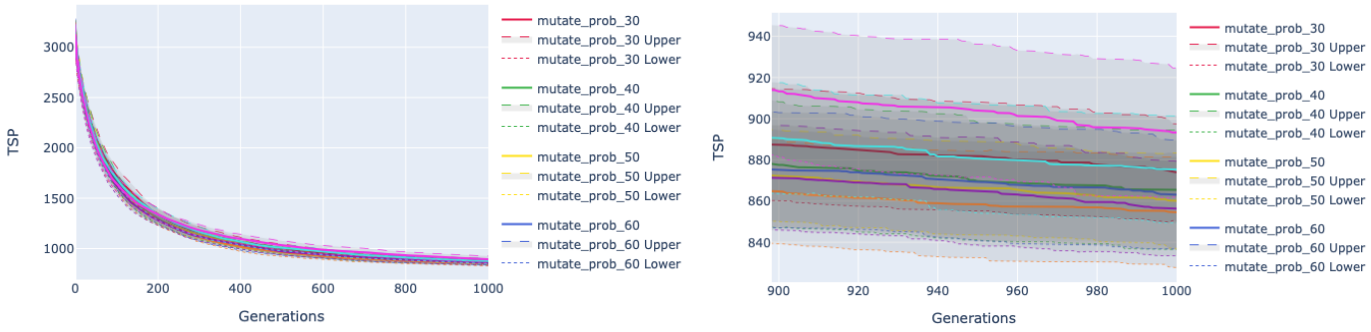
- Initialization: Hill Climbing
- Crossover: Edge
- Mutation: Inversion
- Parent Selection: Tournament Selection
- Replacement: Elitism

The performed tests are presented in Table 2.

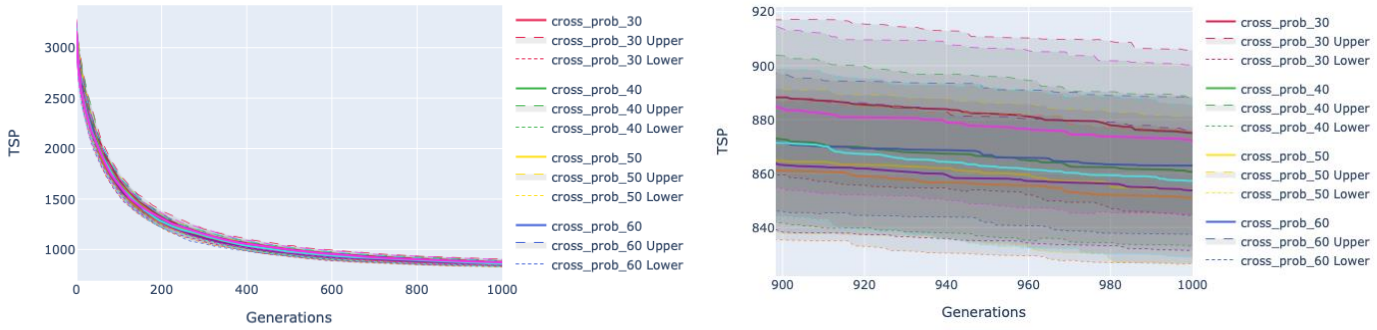
Table 2 - Round 2 Tests

Variation Number	Crossover %	Mutation %	Tournament Size	Folder Name
15	0.3	0.6	10	mutate_prob_30
16	0.4	0.6	10	mutate_prob_40
17	0.5	0.6	10	mutate_prob_50
18	0.6	0.6	10	mutate_prob_60
19	0.7	0.6	10	mutate_prob_70
20	0.8	0.6	10	mutate_prob_80
21	0.9	0.6	10	mutate_prob_90
22	1.0	0.6	10	mutate_prob_100
23	0.6	0.3	10	cross_prob_30
24	0.6	0.4	10	cross_prob_40
25	0.6	0.5	10	cross_prob_50
26	0.6	0.6	10	cross_prob_60
27	0.6	0.7	10	cross_prob_70
28	0.6	0.8	10	cross_prob_80
29	0.6	0.9	10	cross_prob_90
30	0.6	1.0	10	cross_prob_100
31	0.6	0.6	5	tournament_size_05
32	0.6	0.6	10	tournament_size_10
33	0.6	0.6	15	tournament_size_15
34	0.6	0.6	20	tournament_size_20
35	0.6	0.6	30	tournament_size_30

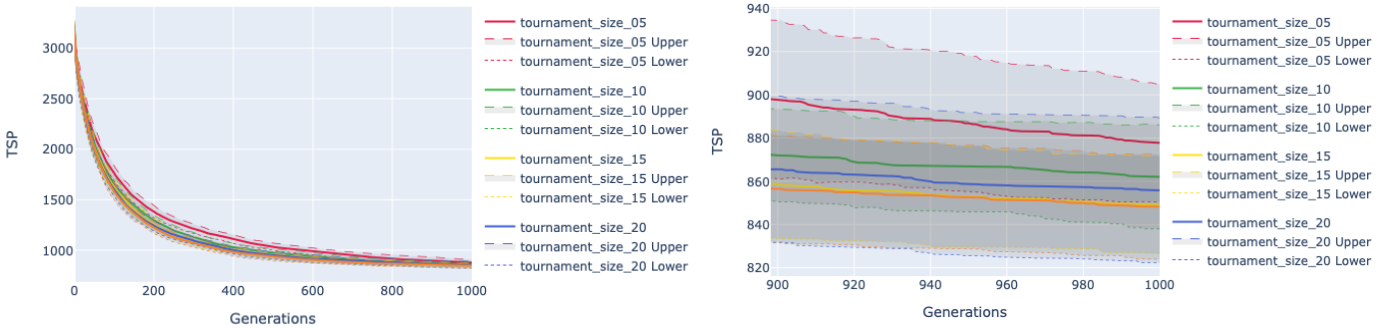
Increasing the mutation probability should allow the algorithm to search more of the neighborhood of the population. Therefore, it is interesting that the results show that a 100% probability of mutation is not the best. 70% is the best here with 100% being the worst.



The xover probability is interesting to look at as it shows that using too low or too high of value is not good. Too low and not enough of the search space is explored. Too high and the jumps between points in the search space are too large and the chance of finding the global optimum is lowered. The best value is 0.7 (70%) here.



The size of the tournaments and their results show that as more participants take part in a “game”, the better the final outcome of the fitness. More participants mean that there is a high chance that the parents are just the best parents as the selection of participants is random. At a tournament size of 30, almost certainly at least one of the participants in each game is the best parent. This means that this acts as a very strict selection, only picking non-best parents a small percentage. The best size is 30 participants.



2.7.3. Round 3

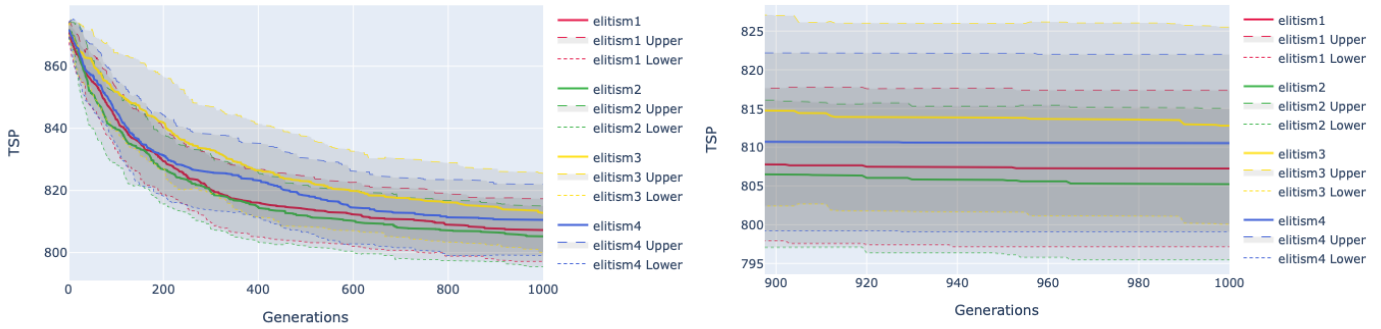
For the third round, a few experimental operators are tried out. The n-queens initialization method and several different elitisms are tested. For the n-queens the following parameters are kept constant:

- Crossover: Edge
- Mutation: Inversion
- Parent Selection: Tournament Selection
- Crossover Rate: 0.7
- Mutation Rate: 0.7
- Tournament Size: 30

Variation Number	Initialization	Elitism	Folder Name
36	Random	elitism1	init_rand1
37	HC	elitism1	init_hc1
38	Nqueens	elitism1	init_nqueen1
39	Nqueens	elitism1	elitism1
40	Nqueens	elitism2	elitism2
41	Nqueens	elitism3	elitism3
42	Nqueens	elitism4	elitism4

From the results, it is very clear that n-queens is much more effective at finding a near-optimal solution from the beginning. With better initial values the algorithm is able to find better final results as well. The zoomed-in graph also shows that at 1000 generations, the hill-climbing algorithm is actually slightly worse than the random initialization.

Testing different elitisms is interesting as it gives more insights on how many of the parents should be kept as offspring. More quality parents mean a better population but at the same time a population that is less diverse and therefore more likely to become stuck in a local optimum. From the results, elitism 1 and 2 are quite similar to elitism 2 a few fitness points better.



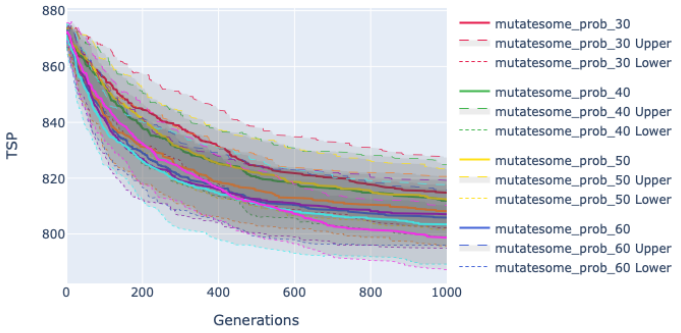
2.7.4. Round 4

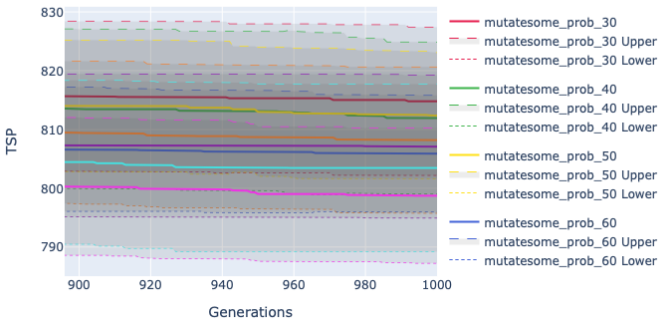
As the mutation rate results were not as expected, the rates were rerun with the best parameters calculated so far:

- Crossover: Edge
- Mutation: Inversion
- Parent Selection: Tournament Selection
- Replacement: Elitism 2
- Crossover Rate: 0.7
- Tournament Size: 30
- Initialization: n-queens

Variation Number	Mutation %	Folder Name
43	0.3	mutationsome_prob_30
44	0.4	mutationsome_prob_40
45	0.5	mutationsome_prob_50
46	0.6	mutationsome_prob_60
47	0.7	mutationsome_prob_70
48	0.8	mutationsome_prob_80
49	0.9	mutationsome_prob_90
50	1	mutationsome_prob_100

From the results of the rerun values, the higher % rates perform better. 100% performs the best here. The change from the last mutation % result could come from the use of different initialization methods. Perhaps as the n-queens start the GA with better results, the need for better local searching is higher (thus a higher mutation %). With the other initialization methods, the fitness was higher and a broader search was required.

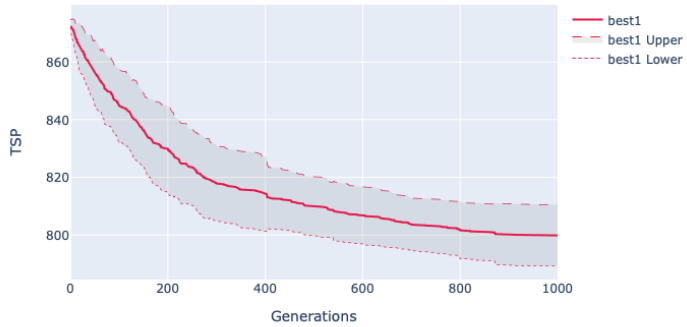


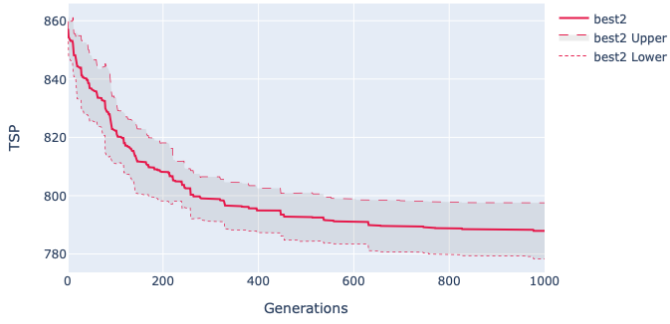


In the end, the best parameters for the TSP problem are found to be:

- Crossover: Edge
- Mutation: Inversion
- Parent Selection: Tournament Selection
- Replacement: Elitism 2
- Crossover Rate: 0.7
- Mutation Rate: 1.0
- Tournament Size: 30
- Initialization: n-queens

Running just these parameters gives the best-case scenario for this dataset, with the graph on the left using 100 iterations of the n-queens and the one on the right using 10000 iterations.





The best solution, with a fitness of 776.7047, was found to be:

[74, 17, 40, 34, 76, 55, 59, 68, 0, 50, 27, 66, 9, 28, 53, 46, 79, 10, 45, 51, 1, 6, 62, 49, 18, 70, 13, 41, 11, 15, 80, 77, 67, 22, 83, 89, 16, 14, 60, 78, 39, 7, 25, 31, 37, 64, 69, 35, 63, 71, 19, 29, 43, 5, 12, 54, 4, 56, 42, 36, 88, 21, 73, 32, 23, 47, 57, 87, 3, 81, 48, 8, 85, 52, 84, 2, 65, 20, 86, 82, 72, 75, 61, 24, 33, 44, 30, 58, 38, 26]

3. Portfolio Investment Problem (PIP)

Using the past returns of companies that compose the SP500 index, the objective of this problem is to maximize the return of a hypothetical portfolio. Despite the problem have been proposed as taking into account the initial investment budget, the team decided to implement it as a return-only problem, i.e., to evaluate the fitness considering only the portfolio return rate what makes the solutions independent of the invested monetary values.

3.1. Encoding

The solution (portfolio) was encoded as a list of length 503, the number of assets in the SP500 index. Before generating a solution, each list is created with zeros in all positions. Whether a company is in the portfolio is determined by whether its position on the list contains a nonzero value. Then, a number between 1 and `Maximum-Portfolio-Size` is randomly selected to be the portfolio size, i.e., the number of assets that will compose the portfolio.

Having selected the portfolio size, the assets that will compose the portfolio are also randomly selected, being represented by a set of positions in the solution that will be updated to nonzero values (`assets_in_portfolio`).

After that, two different ways of generating portfolio distribution were implemented:

- `portfolioDistributionV1`: divides the [0,100] range in a number (portfolio size) of random-length slices. The solution is updated with each slice being randomly assigned to a position contained in `assets_in_portfolio`.

- `portfolioDistributionV2`: selects a random number between [1,100] that represents the first slice. After that, the algorithm iteratively selects new slices with a length between 1 and the sum of lengths of the other slices. The number of slices will again be equal to the portfolio size and each slice is randomly assigned to a position contained in `assets_in_portfolio`. This second approach biases the portfolio distribution to have larger weights at the beginning of the distribution and small ones in the last rounds.

Since the two distribution methods explore the solutions space in different ways, the problem algorithm randomly selects which approach will be used to generate a solution.

A possible solution using the encoding above for a `Maximum-Portfolio-Size = 20` is:

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 0, 0,
3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 11, 6, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

3.2. Initialization

The random initialization was the most used approach for this problem. Hill Climbing was also tested with results shown in section 3.7.7.

3.3. Solution evaluation

The solution evaluation function (`evaluate_solution`) was divided into three steps:

- Calculating the return of the portfolio (fitness);
- Calculating the standard deviation of the portfolio (used in the Sharpe Ratio);
- Calculating the Sharpe Ratio for the portfolio.

3.3.1. Fitness

The return R_P of each solution (portfolio) represents its fitness value, calculated by the formula:

$$R_p = \sum_{i=1}^{n} w_i r_i, \text{ where } \sum_{i=1}^{n} w_i = 1,\ r_i = asset\ return,\ w_i = asset\ proportion$$

3.3.2. Standard deviation

The standard deviation σ_P of the portfolio is given by the following formula:

$$\sigma_P = \sqrt{\sum_{i=1}^{N} w_i^2\, \sigma^2(k_i) + \sum_{i=1}^{N}\sum_{i\neq j}^{N} w_i\, w_j\, Cov(k_i, k_j)}$$

where N is a number of assets in a portfolio, w_i is a proportion of *i*th asset in a portfolio, w_j is a proportion of *j*th asset in a portfolio, $\sigma^2(k_i)$ is variance of return of *i*th asset, and $Cov(k_i,k_j)$ is covariance of returns of *i*th asset and *j*th asset.

In terms of correlation coefficient, the formula above can be transformed as follows:

$$R(k_i, k_j) = \frac{Cov(k_i, k_j)}{\sigma(k_i)\,\sigma(k_j)}$$

$$Cov(k_i, k_j) = R(k_i, k_j)\,\sigma(k_i)\,\sigma(k_j)$$

where $R(k_i,kj)$ is the correlation coefficient of returns of *i*th asset and *j*th asset, $\sigma(k_i)$ is the standard deviation of return of the *i*-th asset, and $\sigma(kj)$ is the standard deviation of return of the *j*-th asset.

$$\sigma_P = \sqrt{\sum_{i=1}^{N} w_i^2\, \sigma^2(k_i) + \sum_{i=1}^{N}\sum_{i\neq j}^{N} w_i\, w_j\, R(k_i, k_j)\,\sigma(k_i)\,\sigma(k_j)}$$

3.3.3. Sharpe Ratio

Sharpe Ratio is a measure of risk for the portfolio, it's a way to control the risk exposure of the investor and the higher its value, the higher the risk. The ratio is calculated by the following formula, where R_{RF} is the risk-free return, equivalent to a US treasury bond (fixed on 1.56%/yr for this problem):

$$\frac{R_P - R_{RF}}{\sigma_P}$$

3.4. Mutations

All mutation techniques have been applied to this problem, with results in section 3.7. Considering the size of the list that represented each solution and the `Maximum-Portfolio-Size`, the ordinary swap mutation was not efficient, swapping zeros most of the time. In order to address this concern, a specific approach was implemented for the PIP: `swap_mutation_pip`.

Its only difference is the test made with the first chosen position, which must be nonzero, guaranteeing that at least one mutation point has a gene different from zero. The results for this mutation approach are also shown in the Results section.

3.5. Crossovers

All the crossover approaches considered in the TSP could lead to unfeasible solutions, since they are limited by the portfolio size and the sum of the weights (must be equal to 100). The weights sum, at least, can be corrected by normalizing the solution (dividing all the elements by the new sum of weights). Taking into account that PMX, Cycle, Edge, and Order 1 crossovers would frequently create solutions with invalid portfolio sizes, only Single Point Crossover was applied to PIP. In addition, four new approaches were implemented:

- `singlepoint_crossover_pip`: the same as the single point crossover but normalizes the portfolio after performing the operation.
- `two_point_crossover_pip`: the same as the two-point crossover, also normalizing the portfolio.
- `pipArithmeticCrossover`: the offsprings are arithmetic combinations of the parents. Despite the possibility of invalid sizes and the need of normalizing the resulting portfolios, this approach was implemented to better explore the portfolios with higher number of assets.
- `pipHeuristicCrossover`: the first offspring is the parent with the best fitness and the second offspring is the arithmetic combination with the best fitness.

3.6. Admissibility

The admissibility function evaluates three aspects of the solution:

- The number of assets is smaller than the maximum portfolio size;
- The sum of weights is equal to 100%;
- The Sharpe ratio is higher than 1.0.

3.7. Results

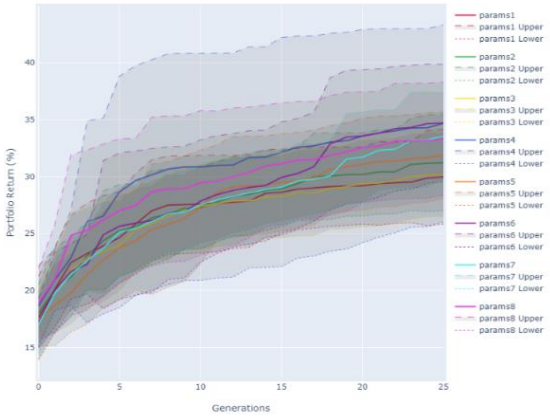
In order to find the best possible configuration for the PIP, the possible parameter combinations were tested in rounds. In round 1, a variety of combinations was tested to achieve a first notion of which parameters should be fixed or fine-tuned and which operators and parent selection would work better.

3.7.1. Round 1

As stated above, in this round many parameters were changed to pursue an initial direction for the algorithm tuning. Despite configurations 4 and 6 had similar fitness, the second one was considered to be the best since it showed a lower standard deviation in the results.

Fixed parameters:
Population Size: 40
Generations: 25
Runs: 15
Initialization: Random
Replacement: Elitism

Captions:
Changes from baseline
Best configurations



The following configurations were tested:

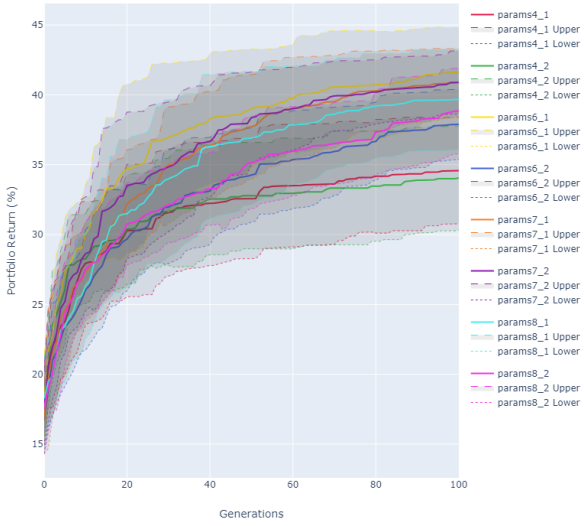
Configuration Number	Crossover (probability)	Mutation (probability)	Parent Selection (Size)	Folder Name
1 - Baseline	pipArithmeticCrossover (0.8)	inversion_mutation (0.8)	Tournament (5)	../log/pip/params1
2	pipArithmeticCrossover (0.2)	inversion_mutation (0.8)	Tournament (5)	../log/pip/params2
3	pipHeuristicCrossover (0.2)	inversion_mutation (0.8)	Tournament (5)	../log/pip/params3
4	pipHeuristicCrossover (0.2)	inversion_mutation (0.8)	Tournament (2)	../log/pip/params4
5	pipArithmeticCrossover (0.8)	swap_mutation_pip (0.8)	Tournament (5)	../log/pip/params5
6	pipArithmeticCrossover (0.2)	swap_mutation_pip (0.8)	Tournament (5)	../log/pip/params6
7	pipHeuristicCrossover (0.2)	swap_mutation_pip (0.8)	Tournament (5)	../log/pip/params7
8	pipHeuristicCrossover (0.2)	swap_mutation_pip (0.8)	Tournament (2)	../log/pip/params8

3.7.2. Round 2

Since four of the previous configurations (4, 6, 7 and 8) had similar fitness results, this round was used to test slight changes in the parameters. The results confirmed configuration 6 (here named 6_1), to be the best initial hypothesis.

Fixed parameters:
Population Size: 40
Generations: 100
Runs: 15
Initialization: Random
Replacement: Elitism

Captions:
Changes from first configuration
Best configuration(s)



The following configurations were tested:

Configuration Name	Crossover (probability)	Mutation (probability)	Parent Selection (Size)	Folder Name
4_1	pipHeuristicCrossover (0.2)	inversion_mutation (0.8)	Tournament (2)	../log/pip/params4_1
4_2	pipHeuristicCrossover (0.2)	inversion_mutation (0.9)	Tournament (2)	../log/pip/params4_2
6_1	pipArithmeticCrossover (0.2)	swap_mutation_pip (0.8)	Tournament (5)	../log/pip/params6_1
6_2	pipArithmeticCrossover (0.2)	swap_mutation_pip (0.8)	Tournament (2)	../log/pip/params6_2
7_1	pipHeuristicCrossover (0.2)	swap_mutation_pip (0.8)	Tournament (5)	../log/pip/params7_1
7_2	pipHeuristicCrossover (0.2)	swap_mutation_pip (0.9)	Tournament (5)	../log/pip/params7_1
8_1	pipHeuristicCrossover (0.2)	swap_mutation_pip (0.8)	Tournament (2)	../log/pip/params8_1
8_2	pipHeuristicCrossover (0.2)	swap_mutation_pip (0.9)	Tournament (2)	../log/pip/params8_1

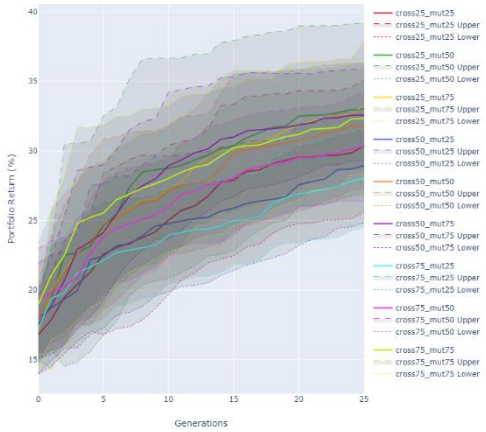
3.7.3. Round 3

In the third round, pipArithmeticCrossover and swap_mutation_pip were tested with different probabilities. The goal was to know in what regions the probabilities should be positioned to achieve the best fitness. The cross25_mut50 and cross25_mut75 configurations returned the best fitness, but the second one resulted in a much lower standard deviation.

To leverage existing results and in line with this round findings, the crossover and mutation probabilities were set to 20% and 80%, respectively, for the following rounds.

Fixed parameters:
Population Size: 40
Generations: 25
Runs: 15
Initialization: Random
Replacement: Elitism
Tournament Size: 5

Captions:
Changes from first configuration
Best configuration(s)



The following configurations were tested:

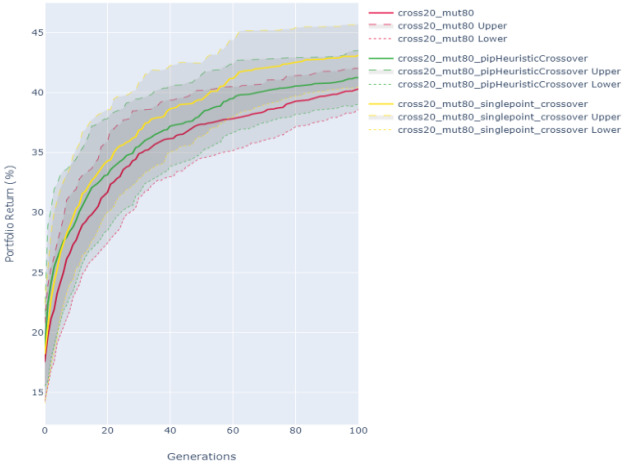
Configuration Name	Crossover (probability)	Mutation (probability)	Folder Name
cross25_mut25	pipArithmeticCrossover (0.25)	swap_mutation_pip (0.25)	../log/pip/cross25_mut25
cross25_mut50	pipArithmeticCrossover (0.25)	swap_mutation_pip (0.50)	../log/pip/cross25_mut50
cross25_mut75	pipArithmeticCrossover (0.25)	swap_mutation_pip (0.75)	../log/pip/cross25_mut75
cross50_mut25	pipArithmeticCrossover (0.50)	swap_mutation_pip (0.25)	../log/pip/cross50_mut25
cross50_mut50	pipArithmeticCrossover (0.50)	swap_mutation_pip (0.50)	../log/pip/cross50_mut50
cross50_mut75	pipArithmeticCrossover (0.50)	swap_mutation_pip (0.75)	../log/pip/cross50_mut75
cross75_mut25	pipArithmeticCrossover (0.75)	swap_mutation_pip (0.25)	../log/pip/cross75_mut25
cross75_mut50	pipArithmeticCrossover (0.75)	swap_mutation_pip (0.50)	../log/pip/cross75_mut50
cross75_mut75	pipArithmeticCrossover (0.75)	swap_mutation_pip (0.75)	../log/pip/cross75_mut75

3.7.4. Round 4

The fourth round was dedicated to testing the possible crossover operators. Despite the two operators that were specifically designed for this problem, the single-point crossover was the one that showed the best results.

Fixed parameters:
Population Size: 40
Generations: 100
Runs: 15
Initialization: Random
Replacement: Elitism
Tournament Size: 5
Mutation (probability): swap_mutation_pip (0.8)

Captions:
Changes from first configuration
Best configuration(s)

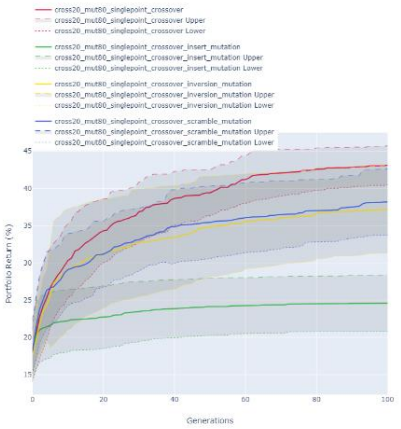


The following configurations were tested:

Configuration Name	Crossover (probability)	Folder Name
cross20_mut80	pipArithmeticCrossover (0.2)	../log/pip/cross20_mut80
cross20_mut80_pipHeuristicCrossover	pipHeuristicCrossover (0.2)	../log/pip/cross20_mut80_pipHeuristicCrossover
cross20_mut80_singlepoint_crossover	singlepointCrossover (0.2)	../log/pip/cross20_mut80_singlepoint_crossover

3.7.5. Round 5

Similar to the previous round, this one served to test the possible mutation operations and swap_mutation_pip revealed to be the best one.



Fixed parameters:
Population Size: 40
Generations: 100
Runs: 30
Initialization: Random
Replacement: Elitism
Tournament Size: 5
Crossover (probability): singlepoint_crossover (0.2)

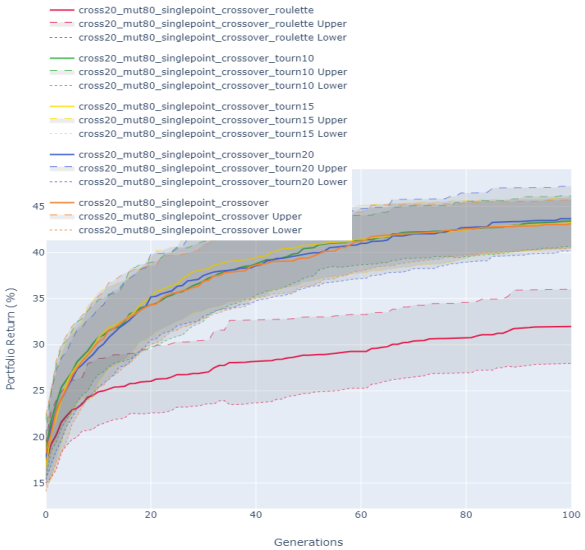
Captions:
Changes from first configuration
Best configuration(s)

The following configurations were tested:

Configuration Name	Mutation (probability)	Folder Name
cross20_mut80_singlepoint_crossover	swap_mutation_pip (0.8)	../log/pip/cross20_mut80_singlepoint_crossover
cross20_mut80_singlepoint_crossover_insert_mutation	insert_mutation (0.8)	../log/pip/cross20_mut80_singlepoint_crossover_insert_mutation
cross20_mut80_singlepoint_crossover_inversion_mutation	inversion_mutation (0.8)	../log/pip/cross20_mut80_singlepoint_crossover_inversion_mutation
cross20_mut80_singlepoint_crossover_scramble_mutation	scramble_mutation (0.8)	../log/pip/cross20_mut80_singlepoint_crossover_scramble_mutation

3.7.6. Round 6

The sixth round tested two possible parent selection algorithms and, in the case of Tournament Selection, different tournament sizes were applied. The first finding was that the Roulette Wheel algorithm has a very poor performance in comparison to Tournament Selection. Besides that, three tournament sizes had similar performances, but the first one achieved the lowest standard deviation and is less computer-intensive.



Fixed parameters:
Population Size: 40
Generations: 100
Runs: 30
Initialization: Random
Replacement: Elitism
Mutation (probability): swap_mutation_pip (0.8)
Crossover (probability): singlepoint_crossover (0.2)

Captions:
Changes from first configuration
Best configuration(s)

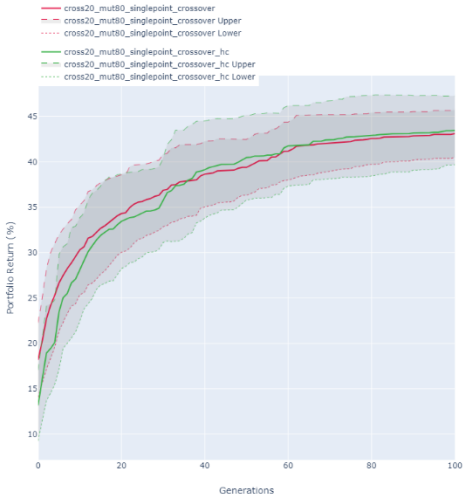
The following configurations were tested:

Configuration Name	Parent Selection (Size)	Folder Name
cross20_mut80_singlepoint_crossover	Tournament (5)	../log/pip/cross20_mut80_singlepoint_crossover
cross20_mut80_singlepoint_crossover_tourn10	Tournament (10)	../log/pip/cross20_mut80_singlepoint_crossover_tourn10
cross20_mut80_singlepoint_crossover_tourn15	Tournament (15)	../log/pip/cross20_mut80_singlepoint_crossover_tourn15
cross20_mut80_singlepoint_crossover_tourn20	Tournament (20)	../log/pip/cross20_mut80_singlepoint_crossover_tourn20
cross20_mut80_singlepoint_crossover_roulette	Roulette Wheel	../log/pip/cross20_mut80_singlepoint_crossover_roulette

3.7.7. Round 7

Hill climbing was also implemented, as an alternative to the random initialization on the PIP. The neighborhood function was designed to swap every nonzero value with its subsequent value. Each swap generated a different neighbor, which means that a 20-assets-maximum portfolio would have at most 20 neighbors. The maximum number of iterations was set to 100.

This algorithm, though, heavily increased the problem solution time, in a way that the test needed to be limited to 25 runs instead of 30. Despite that, hill climbing didn't significantly improve the final solution quality, causing its use to be discarded.



Fixed parameters:
Population Size: 40
Generations: 100
Runs: 25
Parent Selection (Size): Tournament (5)
Replacement: Elitism
Mutation (probability): swap_mutation_pip (0.8)
Crossover (probability): singlepoint_crossover (0.2)

Captions:
Changes from first configuration
Best configuration(s)

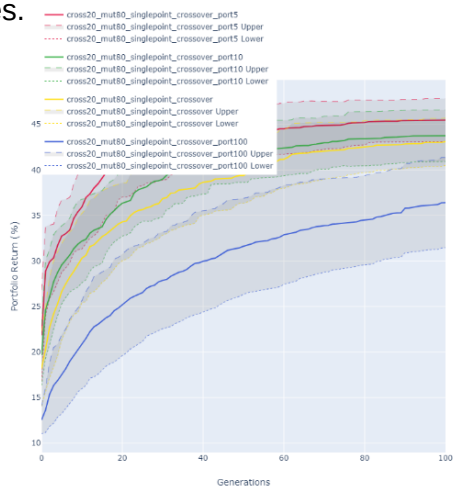
The following configurations were tested:

Configuration Name	Initialization	Folder Name
cross20_mut80_singlepoint_crossover	Random	../log/pip/cross20_mut80_singlepoint_crossover
cross20_mut80_singlepoint_crossover_hc	Hill Climbing	../log/pip/cross20_mut80_singlepoint_crossover_hc

3.7.8. Round 8

In order to evaluate the influence of the maximum portfolio size in the algorithm efficiency, this round was dedicated to testing three different sizes (5, 10 and 100) in comparison with the 20-assets-maximum portfolio adopted as default.

The following results show that as the maximum portfolio size decreases, the fitness means get higher and faster the results converge to the best values. It means that for each dataset the maximum portfolio size can also be tuned to accelerate the convergence to optimal values, despite higher maxima configurations can also test solutions with lower portfolio sizes.



Fixed parameters:
Population Size: 40
Generations: 100
Runs: 30
Parent Selection (Size): Tournament (5)
Replacement: Elitism
Mutation (probability): swap_mutation_pip (0.8)
Crossover (probability): singlepoint_crossover (0.2)

Captions:
Changes from first configuration
Best configuration(s)

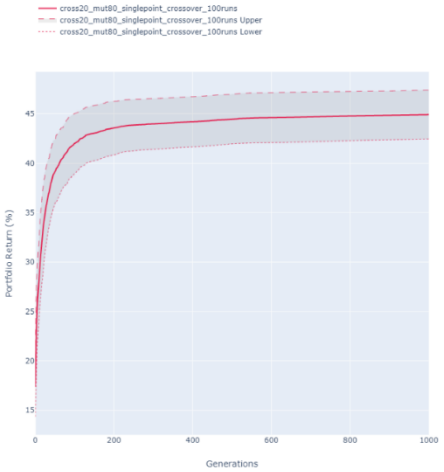
The following configurations were tested:

Configuration Name	Maximum Portfolio Size	Folder Name
cross20_mut80_singlepoint_crossover_port5	5	../log/pip/cross20_mut80_singlepoint_crossover_port5
cross20_mut80_singlepoint_crossover_port10	10	../log/pip/cross20_mut80_singlepoint_crossover_port10
cross20_mut80_singlepoint_crossover	20	../log/pip/cross20_mut80_singlepoint_crossover
cross20_mut80_singlepoint_crossover_port100	100	../log/pip/cross20_mut80_singlepoint_crossover_port100

The best results obtained for 5, 10, 20 and 100 maximum portfolio sizes were, respectively, 49.01%, 50.27%, 48.92%, and 47.25%, which means that the best solutions are similar while the necessary number of runs to achieve the results may vary significantly. In the real world, it is necessary to take into account the balance between algorithm efficiency and portfolio diversification obtained by adding more assets (and possibly obtaining a higher Sharpe Ratio).

3.7.9. Round 9

In the final round, the best configuration for a maximum portfolio size of 20 assets was tested with 100 runs and 1000 generations.



Fixed parameters:
Population Size: 40
Generations: 1000
Runs: 100
Parent Selection (Size): Tournament (5)
Replacement: Elitism
Mutation (probability): swap_mutation_pip (0.8)
Crossover (probability): singlepoint_crossover (0.2)

The best fitness (return) obtained in this final run was **52.79%/yr**, with the following portfolio:

Share (%)	symbol	Name	exp_return_3m (%)	std_deviation (%)
89	ALGN	Align Technology	54.02	46.40777565
1	BAC	Bank of America Corp	22.34	9.457633208
1	COTY	Coty Inc	35.64	30.68877112
1	HPE	Hewlett Packard Enterprise Comp	32.3	8.769131347
1	STT	State Street Corp	44.55	29.85719963
6	TGT	Target Corp	47.82	40.67472137
1	TIF	Tiffany & Company	49.93	24.09284566

The best fitness obtained in all runs was 54.02%/yr (round 5, with inversion mutation), with a one-asset-only portfolio concentrated in ALGN. Since this is the asset with the highest expected return in SP500 and its standard deviation returns a Sharpe Ratio higher than 1, being a feasible solution, it's possible to say that this is the global optimum, despite not being applicable to real life as a risk control strategy.

A possible way to force a more diversified (and less risky) portfolio would be to increase the Sharpe Ratio requirement. In a single test with **a minimum of 2 as Sharpe Ratio**, the following 11-asset portfolio was achieved with **47.17%/yr** return:

Share (%)	symbol	Name	exp_return_3m (%)	std_deviation (%)
46	ALGN	Align Technology	54.02	46.40777565
1	AMD	Adv Micro Devices	33.4	27.36014811
1	CE	Celanese Corp	11.19	8.481856032
1	COTY	Coty Inc	35.64	30.68877112
1	GE	General Electric Company	35.92	16.31810614
5	HPE	Hewlett Packard Enterprise Comp	32.3	8.769131347
17	PVH	Phillips-Van Heusen Corp	36.15	13.20410045
4	QRVO	Qorvo Inc	39.46	20.47011228
1	STT	State Street Corp	44.55	29.85719963
5	TGT	Target Corp	47.82	40.67472137
18	TIF	Tiffany & Company	49.93	24.09284566