NaCo 21/22 assignment report, group 03

Chen Ji Rong Li, Suzanne Honders, Nina Henninger

Leiden Institute of Advanced Computer Science, The Netherlands

Abstract. This paper aims to analyze and understand a Genetic Algorithm (GA), its operators and its application to optimization problems. For the description of the GA, scientific literature is reviewed and is given by explanation of recombination, mutation, selection and representation. The implementation of the GA is executed in Python in addition to the IOH library. For this a skeleton code is provided by the Natural Computing course. For the application to optimization problems, experiments with inverting problems are performed. Expected is that the degree of success of the GAs to solve optimization problems depends on the chosen operators of the GA. Based on both two and three integer domain inverting problems this paper will give conclusions about the degree of success to solve inverting problems with the help of GAs and their respective operators.

1 Introduction

The following paper will be about a Genetic Algorithm, introduced in the 1970s. The approach of the genetic algorithm is derived from population genetics, including the principles of natural selection and natural genetics. [1]
In natural selection the best adapted or fittest individuals of a population will be selected. The selected individuals will produce offspring which will inherit the beneficial characteristics, resulting in a better fitness in every iteration of this process. This same approach can also be applied to optimization algorithms, like a Genetic Algorithm.
A Genetic Algorithms has a population. This population consists of chromosomes. The amount of chromosomes are preserved throughout the generations. At each generation, the fitness of each chromosome is evaluated so that the best parents for the next generations can be selected. These parents will produce the offspring, which will result in crossovers and mutations. Chromosomes with a higher fitness will have a higher probability of selection so that the new generation will have a higher overall fitness. The cycle is repeated until the desired fitness or the maximum amount of iterations has been reached.
The paper will include a description of the implementation in Python as well as an overview of the principles used in a Genetic Algorithm.

2 Algorithm Description

A Genetic Algorithm consists of five phases: the initial population, the fitness function, the selection, recombination and mutation. The first phase, initial population, is the population at the beginning of the whole process. [1] These initial individuals will be randomly generated to create a diverse population. In the second phase every individual's fitness will be determined with the fitness function. This score will be used to determine if the individual will be selected as a parent or not. The third phase is the selection itself, the fittest individuals will be or have a higher chance to be selected for recombination. In recombination the parents will interchange their genes and create offspring that will take their place in the population. The last phase of the GA is mutation, this will help maintain the diversity in the population. The mutation will cause some of the genes the offspring to change. The offspring will then be part of the new generation. The algorithm stops when the desired fitness or the maximum amount of iterations has been reached.

2.1 Recombination

Recombination is a term originally used in biology for a process in which DNA is broken in pieces and recombined to produce a new combination of genes. This process is essential for the diversification of a population which is important for the success of evolution. Therefore it is also used in genetic algorithms. There are different ways recombination can be achieved.
A straightforward technique for achieving recombination is crossover. In Hasançebi et.al (2000) variations are compared, namely one-, two- and multi-point crossover. Crossover techniques are characterized by paired individuals with a certain amount of cut points. [9] The created fragments are then exchanged between the two individuals to create two combinations of the original pair. In figure 1 the process of a one-point crossover is shown.

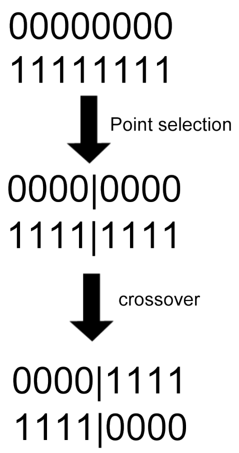


Fig. 1: Example: One-point crossover

In Ombuki et.al (2006) a multi objective genetic algorithm for Vehicle Routing Problem (VRP) was made with a notable recombination operator. [2] In these problems vehicles need to get to a certain set of customers with a known demand. The algorithm contains an interesting cross-over operator that aims to minimize the cost and the number of vehicles used while checking feasible constraints. The process of this operator is shown in figure 2. In the Best Cost Route Crossover (BCRC) the two parents, P1 and P2, are permutations of customers given as numbers. Each of these parents exist of routes shown by RP1 and RP2. In this operator one route of each parent is chosen. The customers of these routes will be removed from the other parent resulting in a shorter child sequence (C1, C2). The removed customers will be reinserted in an arbitrary order.

The location of insertion is infeasible if the routes do not meet the vehicle capacity or time window constraint. The best insertion point is the one that results in the total minimum cost routes. If no feasible insertion point is found then a new route is started as seen in figure 2c.

The one-, two- and multi-point crossover operator can only be used in representations where their are no order or duplication constraints such as with permutations. While the best cost route crossover is made specifically for those kinds of constraints in routing problems.

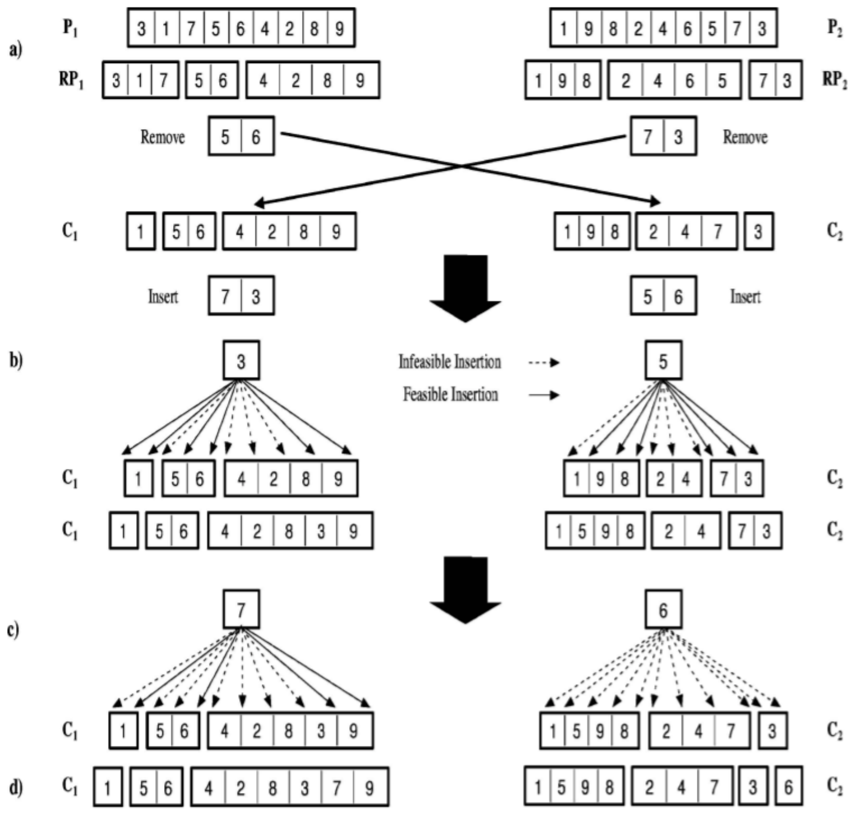


Fig. 2: Example: Best cost route crossover (BCRC) operator. [2]

2.2 Mutation

Mutation is a subtle but important component in the evolution. It rarely causes noticeable changes yet it is needed to introduce new genes and variation to the population. Just like in the biology it is important to have a mutation operator in a genetic algorithm to introduce a random aspect as well as to avoid premature convergence.
The simplest way to introduce variance in a bitstring representation is the use of random bit flips

as referenced in Forest et.al (1996). [6] When this operator is applied there will be a chance for mutation for every bit in the candidate. The mutation causes the bit to "flip" from 0 to 1 or 1 to 0. A well known mutation applied to the Travelling Salesman Problem (TSP) is the inversion mutation. In this mutation two points in a normal permutation are chosen and the genetic material between these points is reversed. In Ombuki et.al (2006) an adaptation to the inversion mutation, Constrained Route Reversal (CRR) mutation, was made to minimize the total route disruption. [2] The Constrained Route Reversal mutation is specific for problems like VRPs because instead of selecting points in the whole solution only one route is selected to minimize total route disruption. Furthermore is the length of the inversion limited to three customers. This latter part is done because violation of customer time windows can easily segment the chosen route into multiple routes creating sub optimal candidates. An example of this mutation is shown in figure 2.

As the name suggests random bit flip mutations can only be used on binary representations. On the other hand there is the CRR mutation. This operator was specifically made for the routing problem in ombuki et.al(2006) but can also be used is other routing problems where large changes in the permutation can cause useless candidate solutions. It is also possible to use this operator on other representations such as binary ones, but the effectiveness of this is doubtedly any better than normal inversion mutation.

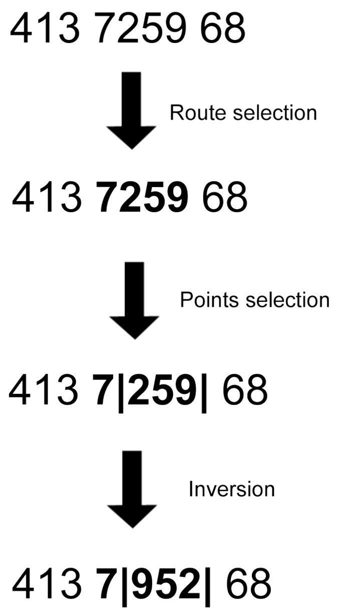


Fig. 3: Example: Constrained Route Reversal Mutation operator.

2.3 Selection

In evolution individuals that adapt faster or are better suited for the environment will survive longer and tend to pass on their genes more successfully. This will improve the quality of the next generation. In selection in a genetic algorithm the individuals to produce offspring are also selected based on their fitness. The individuals with a higher fitness will have a higher chance to be selected than individuals with a lower fitness.
For selection holds: $p_i = \frac{f(a_i)}{\sum_{j=1}^{\lambda} f(a_j)}$. With f for fitness and λ for population size. For selection in a genetic algorithm are different strategies at hand. Resulting tournament selection to perform better than roulette wheel selection and rank based roulette wheel selections. [20] In tournament selection random individuals are selected and their fitness is compared in so called "tournaments". The individuals with the best fitness are selected. After a certain number of tournaments, the selected individuals will become part of the next generation.

2.4 Representation

In a genetic algorithm many operations are done on solution candidates. For this to work these candidates need to be represented in the algorithm. The simplest form of representation is a binary representation but other representations such as float point representation and permutation representation are also used. [11] [3] Using different representations will need different kind of operations. For example when using a permutation representation normal crossover operations can not be applied like one would do to a binary representation. This will frequently cause duplication of numbers in a single candidate and make the candidate unusable. A crossover operator that could be used would be like order crossover, where a subset of corresponding genes are interchanged between the two parents. [17] This also goes for the mutation operator which can not be a simple bit flip any more. Operations like swap mutations are the most subtle. It takes a two genes in the sequence and swap their position so only the order is changed. [18] This shows that the representation of solutions has a large impact on the overall genetic algorithm.

3 Implementation

In this paper an implementation was done of a genetic algorithm in python. A skeleton implementation has been provided by the lecturers of the course Natural Computing. Furthermore the use of the IOHexperimenter [4] package should be noted.
The implementation of the genetic algorithm exist of the main loop, mutation operator, recombination operator and selection operator. For each of these operators there are two variants that can be interchanged by changing the function name and parameters.
The used mutation operators are the flip bit mutation and the random bit mutation that was looked at earlier. The flip bit mutation flips the whole bitstring when the mutation occurs. So a if an certain candidate solution was 0000 before the mutation it would be 1111 afterwards. The random bit mutation is far more subtle in it's way of introducing new information to the population. Which is by flipping only one bit in the whole sequence of the individual. These mutations will be applied to the offspring that has been formed to, as before mentioned, introduce variety in the population. The frequency of such mutation can be set by a probability parameter denoted by P_m.

The used recombination operators are the k-point crossover that has been shown before and the uniform crossover. In the latter there is an equal chance for each bit to be inherited from either of the parent making this crossover a lot more random than the k-point crossover, where a random point in the bitstrings will be chosen splitting the sequence in k+1 parts. These parts will then be interchanged between the parents forming an offspring with parts from each of them. The probability of this is denoted by P_c.
The used selection operators are the aforementioned roulette wheel selection and the tournament selection. The tournament selection selects a random subset of k candidates of the population and selects the most fit candidate as one of the parents for recombination. By increasing the subset size the chance for the better individuals to be selected will also increase. This is more secure way to keep the genes of the best individuals of a population compared to the roulette wheel, where the chance for an individual is determined by the division of their fitness by the total fitness.
In this implementation the parameters can be configured to optimize the genetic algorithm. These can be seen in table 1.

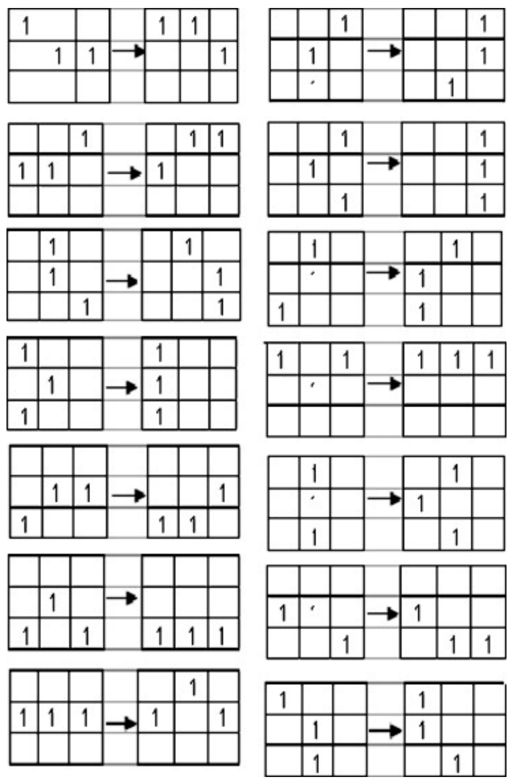
Table 1: Genetic algorithm parameters

M	Number of individuals in the population
P_m	Probability of mutation
P_c	Probability of crossover
s	Subset for tournament selection

4 Application of GA in practice

In practice genetic algorithms can be used all kinds of fields. One of these fields where genetic algorithm have not been striving as much is in the medical field. In this field a lot of information is going unused due to the lack of familiarity with genetic algorithms.
In Ghaheri et.al (2015) many different applications of genetic algorithms are shown in the all kinds of fields in medicine. [7] In the radiology subfield many different imagine technique are being used which generate a large amount of data. The analysis and interpretation by the radiologist needs to be done in a relatively short time. Genetic algorithms can be used in this case for computer-aided detection and diagnosis of complications. This results in faster and more accurate analysis. In one specific case a histogram-based thresh-holding technique was used to generate a binary image to separate the breast and non-breast region in a mammogram (x-ray picture of the breast). [12] In this technique a genetic algorithm was used to enhance the vividness of the border.
The operations in this genetic algorithm were an uniform crossover, a random bit mutation and a linear search through a roulette wheel selection operator. The bitstring in the genetic algorithm represent binary 3x3 kernels corresponding to the gray level mammogram image. The fitness value was calculated using the sum of the intensity values. In addition to the random bit mutation a pre-determined mutation is used. When a kernel show any configuration in fig 4 the corresponding transformation will be performed.

Fig. 4: Pre-determined mutation operator. [12]



The border points and nipple position in the mammogram are aligned and subtracted to obtain an asymmetry image. From this image harmless asymmetries will be isolated and microcalcifications can be extracted. Microcalcifications are associated with invasive breast cancer. An important criterion in order to evaluate diagnostic performance is the detection rate compared to other techniques. This is shown in table 3. It is concluded that the genetic algorithm performed well.

Table 2: Detection rates

Sl. no.	Author and methods	Detection rate (%)
1	Ferrari and Rangayyan [13] , directional filtering with Gabor wavelets	74.40
2	Lau and Bischof [14] , asymmetry measures	85.00
3	Sallam and Bowyer [15] , unwarping technique	86.60
4	The proposed metaheuristic approach-GA, bilateral subtraction	90.60

The reason for the use of a genetic algorithm is that the technique was proposed in Thangavel et al.(2007) but not yet performed in practice. [16] The approach used for this problem is the obvious approach after viewing the propose of Thangavel et al. At last improvements to their genetic algorithm are difficult to tell due to the unpredictability of the change.

5 Cellular Automata

In this project a basic 1-dimensional Cellular Automata (CA) of size n, transition rule Θ, possible values k and neighbourhood size r = 1 is implemented.
In this paper the CA refers to an one dimensional grid of so called cells. The grid is denoted by c_t. Each cell holds a value of the domain. The size of the domain will be denoted as k with k being an element of {2, 3}. The state of a cell is denoted by a_i with i being the position of the cell. The t refers to the amount of times the state has gone through the transition rule. This is a rule that determines the new state of a certain cell, a_i by looking at its previous value and the values of cells within a certain radius r of a_i. In this paper r = 1 so the neighbourhood of a_i^t denoted by s_i^t = $\{a_{i-1}^t, a_i^t, a_{i+1}^t\}$. The t here refers again to the amount of transitions the cell has gone through. These parameters and other denotation are shown in table 3.

With these properties a function can be made that provides an output state after t amount of transitions from a given input state. The code that has been used contains three functions:
1. The neighbourhood function that takes as input the cell i and the whole CA state and gives as output the neighbourhood of cell i.
2. The rule function that takes an integer, changes it into binary and makes a list of these values. These values will determine the state of cell i after the transition.
3. A transition function that takes the binary list and neighbourhood list. For each cell it will take the neighbourhood list to determine the corresponding binary value and will fill the new CA state.

The result is an algorithm that can look at a given state with a certain rule and amount of transitions to determine the end state.

Table 3: Cellular Automata parameters

n	Number of integers in the CA
$k \in \{2,3\}$	Domain size of the integers in the CA
$r \in \{1\}$	Radius that defines what integers in the CA are part of the neighbourhood
$C^t = \{a_1, a_2, ..., a_n\}$	The state of the CA at time t
$a_i^t \in \{0, ..., k\}$	The state of cell i at time t
$s_i^t \in \{0, ..., k\}^{2r+1}$	The state neighbourhood of a cell i at time t
Θ: $\{0, ..., k\}^{2r+1} \longrightarrow \{0, ..., k\}$	Transition rule corresponding to the Wolfram code

6 Solving the inverting problem

The other way around, given an end state of a CA with the corresponding rule number and amount of transitions it has gone through one could determine the begin state, C_0. Such a problem would be called an inverting problem and it is this problem that will be used to test the different operators of the aforementioned genetic algorithm. For this experiment an objective function is also needed to determine the quality of the candidate solutions of the genetic algorithm (also called fitness function). For this problem two different objective functions are implemented:
1. An objective function that takes the candidate solution and uses the rule and transition to get the end state of that particular begin state. This new end state will be compared to the real end state and the percentage of similar cells will determine the score of the candidate solution. This score will be a number between 0 and 100.
2. A similar objective function that will look at the percentage of similar cells, but in this function longer sequences of similar cells will be valued more. This is done by iterating through the candidate solution and counting every similar cell just like the first objective function. However when the function finds a similar cell it will look for consecutive alignments with the true end state until it finds one cell that is not the same. All these consecutive similar cells will be added to the score and then the iteration will go further from where it was before this sequence.

The cells in such inverting problem could also contain values of a ternary domain. To accommodate such a problem in the genetic algorithm, two new mutation functions needed to be created:
1. The integer mutation function changes a random cell within a candidate solution and randomly changes its value to one of the other values. For example value of 0 has equal chance to become 1 or 2.

2. The reverse integer mutation function reverses the order of a sequence of cells. This sequence is chosen at random and the size of this sequence can vary from 2 to the maximum amount of cells. The probability of these events will be the same as the mutation operator for the binary domain, denoted with P_m.

7 Experiment

To determine the nature of the inverting problem, different parameters values can be used. The parameters of these parameters are listed in table 4.
With these parameters, ten different problems are distinguished and are listed in table 5. Unfortunately the experiments for problems 5 and 10 are unable to be performed in a reasonable amount of time.

For these problems there are two kinds of operators for the mutation, recombination, selection and the objective function. All the combinations will be tested on all the problems leading to a total of $2^4 \times 10 = 160$ experiments. The chosen operators for one setup will be given by a binary string e.g. 1010. These numbers determine the mutation, recombination, selection and objective function operators respectively. What each value represents is shown in table 6.The experiments will all be done with standard values for the genetic algorithm as shown in table 7 and a maximum amount of iterations of 500.

Compared to the Random Search (RS) algorithm the Genetic algorithm seems to perform overall better in finding the optimal solution. Except for a few GAs such as:
GA1010, GA1110, GA0010 and GA1100 for problem 3 in figure 7.
GA1110, GA1010, GA0110, GA1100, GA0010, GA0000, GA0100 and GA1000 for problem 4 in figure 8.
GA0100 and GA1000 for problem 6 in figure 9.
GA0101 and GA0001 for problem 7 in figure 10.
GA1011 for problem 8 in figure 11.
GA1010, GA0101, GA1111, GA0011, GA1011, GA1101, GA1001, GA0001 and GA0111 for problem 9 in figure 11.

It should be noted that for k=2 inverting problems the GAs that performed worse than the RS all had the objective function 2 while for the k=3 inverting problems the GAs that performed worse had the objective function 1.

For each problem the performance of the different algorithms can be found in the appendix. For the problems 1 and 2 in figures 5 and 6 respectively the algorithms tend to perform similarly. It can be seen that the all algorithms that end on 1 (objective function 1) tend to start off with a higher score. The convergence in the global optimum is almost equal for all the algorithms.

For problem 3 in figure 7 there seen to be variance. The algorithms using the objective function 1 seem to be stuck in local optima and unable to reach the global optimum while the two GAs that did reach the global optimum both had a code of 01x0 (x being any bit). This means that the use of a flipbit mutation, one point cross over and the objective function 2 are promising in solving the third inverting problem.

For problem 4, figure 8, the objective function 2 seems to be performing significantly worse than the objective function 1. The other operators don't seem to impact the performance of the overall GA except for GA1110 where the bit mutation, one point cross over and tournament selection seem to be doing better in combination with the objective function 2.

For problems with a k=3 such as problem 6 and 7 there seem to be an inconsistency in what algorithm perform better. In problem 6, figure 9, it seems that there is no clear combination of operators resulting in a better performance. In problem 7, figure 10, all the algorithms with 1xx1 (integer mutation and objective function 1) and 11xx (integer mutation and one point cross over) seem to be able to find the global optimum.

For problem 8 it seems that every algorithm combination except for GA1011 is able to reach the global optimum around the same amount of iteration. The only algorithms that seem to be a bit slower are GA0001, GA0011, GA0101 and GA0111. The operators that set these three apart from any other is the use of the integer mutation and objective function 1 (0xx1). The similarity to the results of problem 7 should also be noted.

The results of problem 9 are similar to problem 7 and 8 where GAs with the objective function 2 seem to do better than the GAs with the objective function 1. However in this problem none of the GAs where able to reach the global optimum. Just as in problem 8 the GA1011 seem to perform the worst.

Table 4: Inverting problem parameters

$k \in \{2,3\}$	Domain size of the cells
rule $\in \mathbb{R}$	Transition rule corresponding to the number
$T \in \mathbb{R}$	amount of transitions
CT	True end state of the CA after t transitions

Table 5: CA input.

k	Rule #	T	CT
2	30	1	[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, 1, 1, 1, 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	34	5	[0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
2	45	5	[1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0]
2	30	25	[0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]
2	90	99	[0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0]
3	611715	1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1]
3	611715	5	[0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 2, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 2, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1]
3	611715	5	[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, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 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	1134859650423	25	[0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
3	5039098696122	99	[1, 2, 2, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 0, 0, 0, 2, 2, 2, 1, 2, 0, 1, 1, 0, 0, 2, 2, 2, 2, 2, 1]

Table 6: Setup binary string meaning

Operators	0	1
Mutation (for k = 2)	Flipbit mutation	Bit mutation
Mutation (for k = 3)	Reverse integer mutation	Integer mutation
Recombination	Uniform crossover	One point cross over
Selection	Roulette selection	Tournement selection
Objective function	Objective function 2	Objective function 1

Table 7: Genetic algorithm parameters

M	100
P_m	0.01
P_c	0.7
s	5

8 Application of CA in practice

The report chosen for this part is called *"Treatment Analysis in a Cancer Stem Cell Context Using a Tumor Growth Model Based on Cellular Automata".* [19] The paper uses different approaches with CA models to model cancer growth. Cancer is united by different hallmarks. hallmarks are traits that govern the transformation of normal cells into malignant cells. [19]

Cellular Automata (CA) was the tool that was most used in artificial life for studying and characterizing the emergent behavior. It is defined by a set of rules which establish the next state of each of the sites of a grid environment by using the previous state of this same site and the states of its defined neighborhood, where the states can be associated with the cell states in the intended simulations of tumor growth. [19]

The main goal of this paper was to analyse the effect of treatment applications on cancer growth in a Cancer Stem Cells (CSC) context by using this CA model. The CA model is based on the presence of hallmarks in the cells, and shows the behavior of CSC and their implications for the resultant growth behavior. [19]

The kind of CA that has been used in this study is a three dimensional grid with 125000 cell sites, 50 in each dimension. Initially all cells are healthy except for a few CSCs that are introduced in the inner area (1-5% of grid size). CSCs will divide symmetrically with a probability of $P_s = 0.01$ or asymmetrically to produce a CSC and a non-stem cancer cell with probability $1 \; p_s$. as seen in Enderling et al. [5]

There are some alternatives for modeling cancer growth. For example the traditional approach, which is using differential equations to describe tumor growth. However the approaches that rely on CA models or agent-based models facilitate modeling at cellular level, where the state of each cell is described by its local environment. [19]

They used a CA model that is based on the Hanahan and Weinberg hallmarks [8] in the cells. This model is beneficial for studying the behavior in various scenarios with different hallmarks. [19]

The results of the paper were successful, as they aligned with the outcome of another research done, Hillen et al. [10] In both papers it stated that if Differentiated Cancer Cells (DCCs) compete with CSCs for space and resoures, the DCCs can prevent the CSC division slow down tumor growth. Furthermore the effects of different treatments was analyzed by using continuous low-intensity treatments and periodic high-intensity treatments. The periodic high-intensity treatments favor CSC proliferation and differentiation in the short term and thereby could result in the future regrowth of the tumor. The continuous low-intensity treatments are thus more beneficial as they allow for easy control of future regrowth of tumors. [19]

Due to researches having the tendency to focus on genes and proteins, and therefor do not actually understand how to fight the disease. For this you need to view it as a system, rather than merely as a set of cellular activities. [21] Therefor using CA is very beneficial as this models at a cellular level. [19]

9 Conclusion

Genetic algorithms can be used to solve all kinds of optimization problems and can be improved by problem specific operators. The problem that is looked at in the scope of this research are two and k=3 inverting problems.
Overall the GAs performed better than the RS algorithm but when the amount of transitions increased so did the amount of GAs that would perform worse than the RS. All operators perform equally well on inverting problems with only one transition. When the amount of iterations increases the objective function 1 perform better on k=2 inverting problems while for k=3 problems the objective function 2 performs better.
As for the other operators, the uniform mutation and the one point cross over operator outperform their alternative operator for the k=2 inverting problems. There is no clear difference in performance between the two selection operators.
It can also be concluded that the reverse integer mutation performs better than the integer mutation for k=3 inverting problems. The selection operators and recombination operators do not have a clear performance difference for these problems.

10 Discussion

The low initial score for GAs with the objective function 2 can be explained by the difficulty of this function to get a higher score compared to the objective function 1. However there does not seem to be a direct cause for the switch of behaviour between the GAs with objective function 1 and 2 as seen in problem 7, 8 and 9. This could be the result of the transition rule, where many neighbourhoods lead to a 0 state and the given end state containing many consecutive 0s which leads to a much higher score in the objective function 2. But this should also be seen in the GAs with objective function 1. The inconsistency can be explained by an annotation error which would explain the different result for problem 6 compared to the other three integer inverting problems. Furthermore the difference in objective function would make the comparison between the GAs with different objective functions impractical.
For future research the consistency of these results can be tested as well as the results for problem 5 and 10. Furthermore one could look at inverting problems with a domain with more than three integers. This would of course require a new set of mutation operators or mutation operators in a

similar fashion as the k=3 inverting problems. At last comparison between GAs with the same objective function could be more focused on as well as a RS algorithm that runs on the same objective function.

References

1. Introduction to Genetic Algorithms, howpublished = https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3, note = Accessed: 10-13-2021
2. Beatrice Ombuki, B.J.R., Hanshar, F.: Multi-objective genetic algorithms for vehicle routing problem with time windows. Applied Intelligence **24** (2006), https://doi.org/10.1007/s10489-006-6926-z
3. Bierwirth, C., Mattfeld, D.C., Kopfer, H.: On permutation representations for scheduling problems. In: Voigt, H.M., Ebeling, W., Rechenberg, I., Schwefel, H.P. (eds.) Parallel Problem Solving from Nature — PPSN IV. pp. 310–318. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
4. Doerr, C., Wang, H., Ye, F., van Rijn, S., Bäck, T.: Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. CoRR **abs/1810.05281** (2018), http://arxiv.org/abs/1810.05281
5. Enderling, H., Hahnfeldt, P.: Cancer stem cells in solid tumors: Is 'evading apoptosis' a hallmark of cancer? Progress in Biophysics and Molecular Biology **106**(2), 391–399 (2011). https://doi.org/https://doi.org/10.1016/j.pbiomolbio.2011.03.007, https://www.sciencedirect.com/science/article/pii/S0079610711000381, systems Biology and Cancer
6. Forrest, S.: Genetic algorithms. ACM Comput. Surv. **28**(1), 77–80 (Mar 1996). https://doi.org/10.1145/234313.234350, https://doi.org/10.1145/234313.234350
7. Ghaheri A, Shoar S, N.M.H.S.: The applications of genetic algorithms in medicine. Oman Medical journal (2015). https://doi.org/doi:10.5001/omj.2015.82, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4678452/
8. Hanahan, D., Weinberg, R.A.: The hallmarks of cancer. Cell **100**(1), 57–70 (2000). https://doi.org/https://doi.org/10.1016/S0092-8674(00)81683-9, https://www.sciencedirect.com/science/article/pii/S0092867400816839
9. Hasançebi, O., Erbatur, F.: Evaluation of crossover techniques in genetic algorithm based optimum structural design. Computers Structures **78**(1), 435–448 (2000). https://doi.org/https://doi.org/10.1016/S0045-7949(00)00089-4, https://www.sciencedirect.com/science/article/pii/S0045794900000894
10. Hillen, T.: Jour
11. Janikow, C.Z., Michalewicz, Z.: An experimental comparison of binary and floating point representations in genetic algorithms. In: ICGA (1991)
12. Karnan, M., Thangavel, K.: Automatic detection of the breast border and nipple position on digital mammograms using genetic algorithm for asymmetry approach to detection of microcalcifications. Computer Methods and Programs in Biomedicine **87**(1), 12–20 (2007). https://doi.org/https://doi.org/10.1016/j.cmpb.2007.04.007, https://www.sciencedirect.com/science/article/pii/S016926070700079X
13. Karnan, M., Thangavel, K.: Automatic detection of the breast border and nipple position on digital mammograms using genetic algorithm for asymmetry approach to detection of microcalcifications. Computer Methods and Programs in Biomedicine **87**(1), 12–20 (2007). https://doi.org/https://doi.org/10.1016/j.cmpb.2007.04.007, https://www.sciencedirect.com/science/article/pii/S016926070700079X
14. Karnan, M., Thangavel, K.: Automatic detection of the breast border and nipple position on digital mammograms using genetic algorithm for asymmetry approach to detection of microcalcifications. Computer Methods and Programs in Biomedicine **87**(1), 12–20 (2007). https://doi.org/https://doi.org/10.1016/j.cmpb.2007.04.007, https://www.sciencedirect.com/science/article/pii/S016926070700079X

15. Karnan, M., Thangavel, K.: Automatic detection of the breast border and nipple position on digital mammograms using genetic algorithm for asymmetry approach to detection of microcalcifications. Computer Methods and Programs in Biomedicine **87**(1), 12–20 (2007). https://doi.org/https://doi.org/10.1016/j.cmpb.2007.04.007, https://www.sciencedirect.com/science/article/pii/S016926070700079X
16. Karnan, M., Thangavel, K.: Automatic detection of the breast border and nipple position on digital mammograms using genetic algorithm for asymmetry approach to detection of microcalcifications. Computer Methods and Programs in Biomedicine **87**(1), 12–20 (2007). https://doi.org/https://doi.org/10.1016/j.cmpb.2007.04.007, https://www.sciencedirect.com/science/article/pii/S016926070700079X
17. Koohestani, B.: A crossover operator for improving the efficiency of permutation-based genetic algorithms. Expert Systems with Applications **151**, 113381 (2020). https://doi.org/https://doi.org/10.1016/j.eswa.2020.113381, https://www.sciencedirect.com/science/article/pii/S0957417420302050
18. Liu, B., T. Haftka, R., A. Akgün, M., Todoroki, A.: Permutation genetic algorithm for stacking sequence design of composite laminates. Computer Methods in Applied Mechanics and Engineering **186**(2), 357–372 (2000). https://doi.org/https://doi.org/10.1016/S0045-7825(99)90391-2, https://www.sciencedirect.com/science/article/pii/S0045782599903912
19. Monteagudo, , Santos, J.: Treatment analysis in a cancer stem cell context using a tumor growth model based on cellular automata. PLOS ONE **10**(7), 1–16 (07 2015). https://doi.org/10.1371/journal.pone.0132306, https://doi.org/10.1371/journal.pone.0132306
20. Noraini Mohd Razali, J.G.: Genetic algorithm performance with different selection strategies in solving tsp. Proceedings of the World Congress on Engineering **II**(1), 3–18 (2011). https://doi.org/10.1111/itor.12001, http://www.iaeng.org/publication/WCE2011/WCE2011_pp1134-1139.pdf
21. Savage, N.: Modelling: Computing cancer. Nature (2012). https://doi.org/10.1038/491s62a, https://pubmed.ncbi.nlm.nih.gov/23320290/

A Appendix

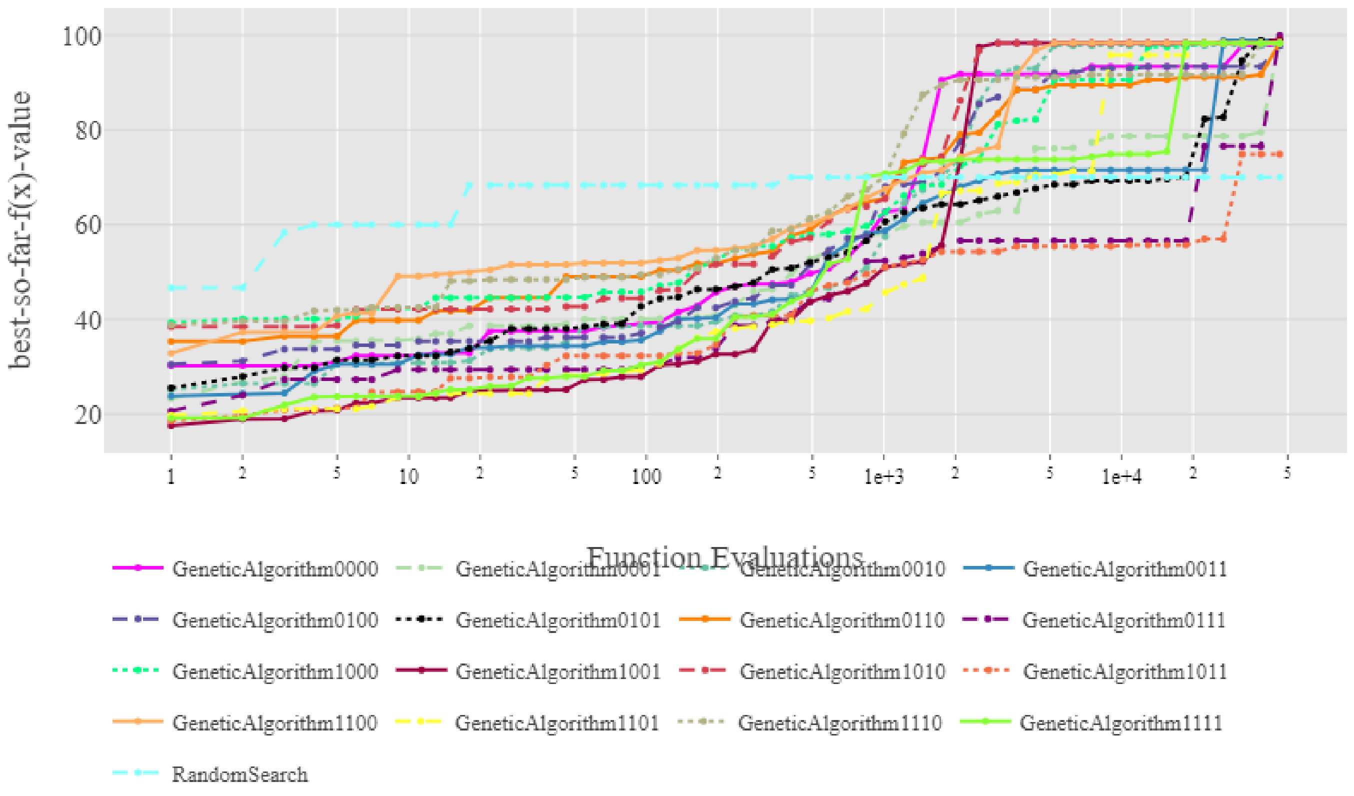


Fig. 5: Plot of performances on problem 1

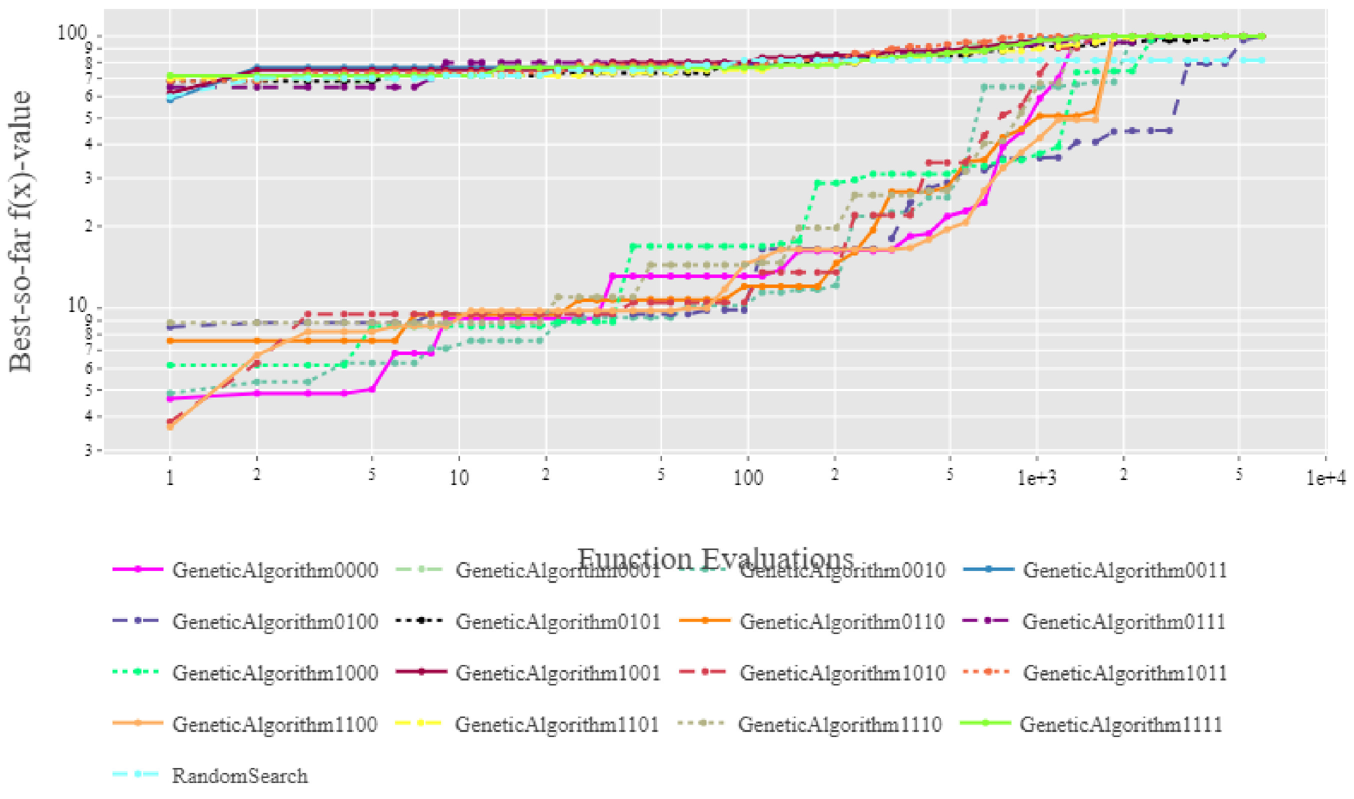


Fig. 6: Plot of performances on problem 2

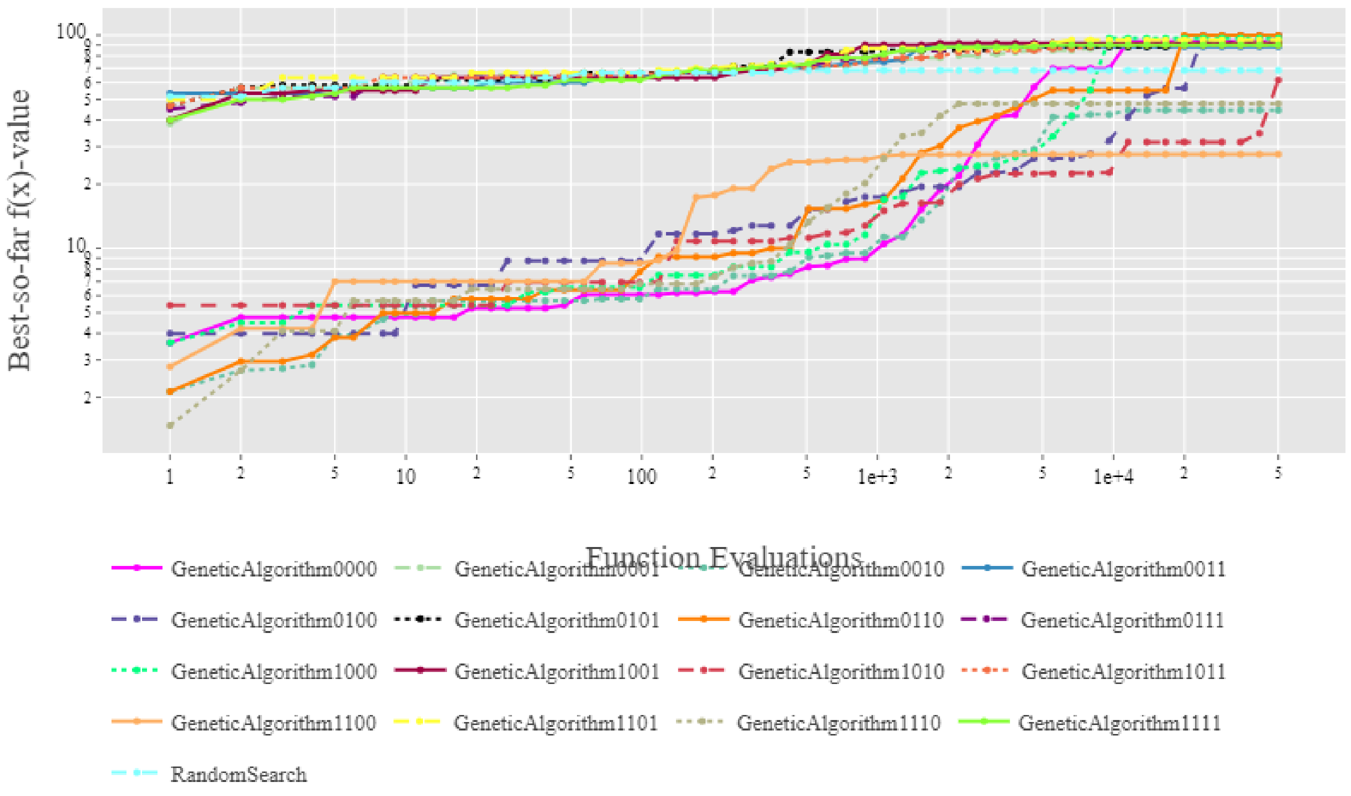


Fig. 7: Plot of performances on problem 3

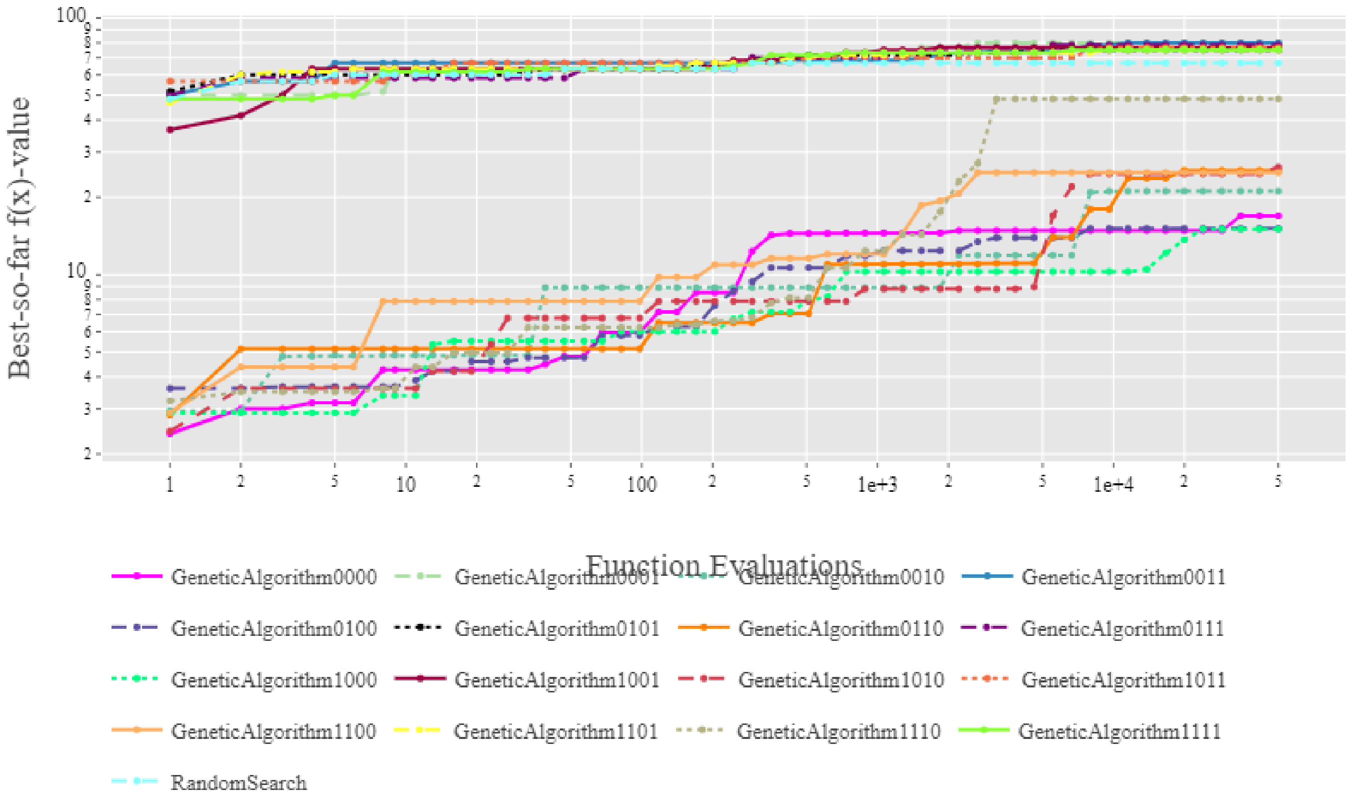


Fig. 8: Plot of performances on problem 4

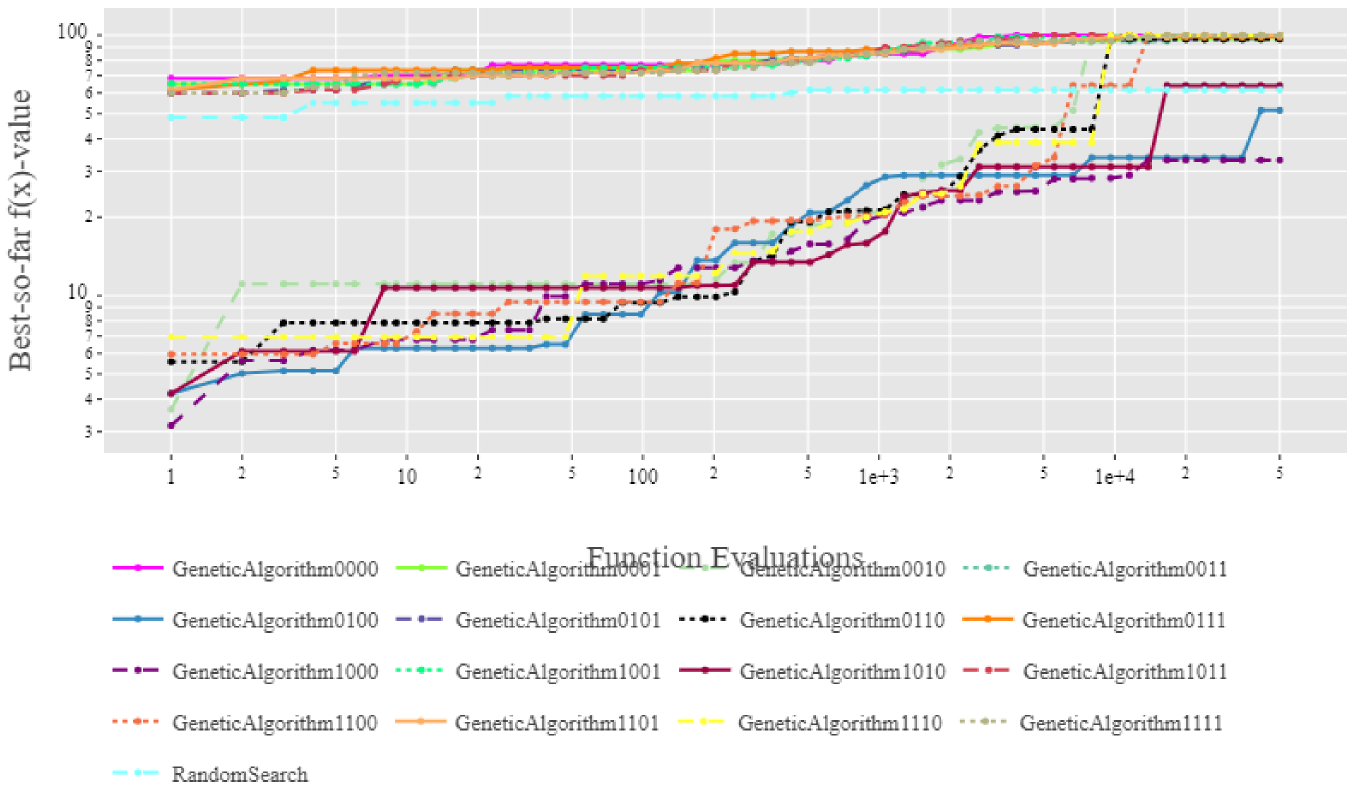


Fig. 9: Plot of performances on problem 6

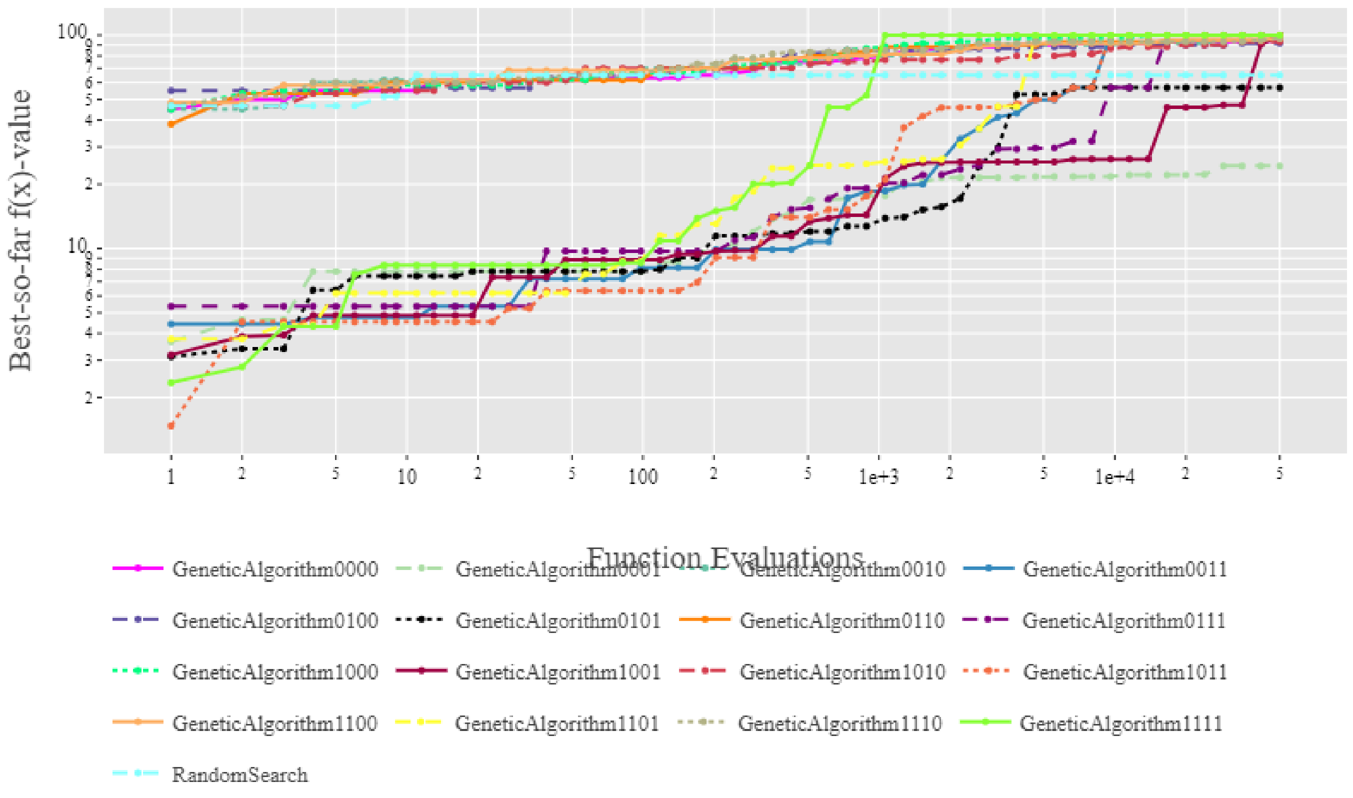


Fig. 10: Plot of performances on problem 7

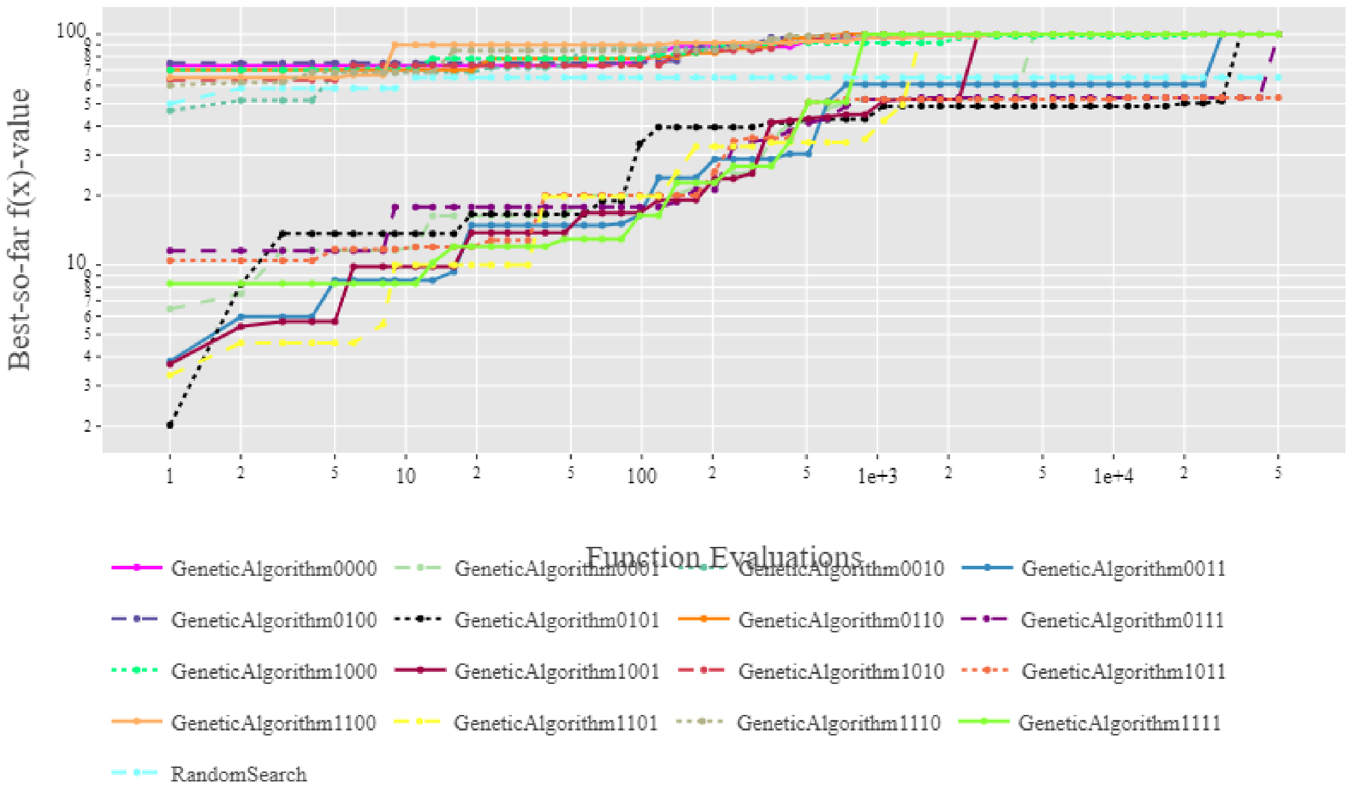


Fig. 11: Plot of performances on problem 8