Genetic Algorithm for a medical optimal control problem

BELFADIL Anas

February 22, 2020

Abstract

Genetic algorithms (GAs) are heuristic search and optimization algorithms, they have been used in a wide range of optimization problems where the space search is too big to be tackled directly by deterministic algorithms. In this exercise we are going to use a genetic algorithm for a medical optimal control problem, i.e. finding the best curative treatment for a tumor if possible, or a palliative treatment otherwise.
We are going to give a statement of the problem, an overview of the genetic algorithm and it's adaptation to this particular problem, finally we are going to present our implementation of the algorithm in C, discuss the results and the improvements performed.

Part I

Statement of the problem:

1 A model for cancer treatment:

Consider *N(t)* the number of tumor cells at time *t*, which, according to the GOMPERTZ growth model, can be modeled with the following differential equation defined only for $N\ (t) \geq 0$:

$$\begin{aligned} N'(t) = N(t) &\left(\lambda ln(\frac{\Theta}{N(t)}) - \sum_{j=0}^{d} k_j \sum_{i=0}^{n-1} C_{ij} x_{(\tau_i,\tau_{i+1}]}(t) \right) \\ =& N(t)(\lambda ln(\Theta) - \lambda ln(N(t)) - drift_i) \qquad \text{whenever } t \in (\tau_i, \tau_{i+1}] \end{aligned}$$

where:

- $drift_i = \sum_{j=1}^{d} k_j C_{ij}$
- λ and Θ are general parameters of the model, being λ the sensitivity or "evolution speed" of the tumor and Θ a kind of a carrying capacity
- n is the number of doses or treatment sessions, given at times $\tau_0 = 0 < \tau_1 < ... < \tau_{n-1}$. The dose at time τ_{n-1} is supposed to act until a given time $\tau_n > \tau_{n-1}$

- $x_{(\tau_i,\tau_{i+1}]}$ is the indicator function on the interval $(\tau_i, \tau_{i+1}]$:

$$x_{(\tau_i,\tau_{i+1}]} = \begin{cases} 1 & t \in (\tau_i, \tau_{i+1}] \\ 0 & otherwise \end{cases}$$

- d is the number of drugs in each dose
- C_{ij} with $i \in \{0, 1, ..., n-1\}$ and $j \in \{1, 2, ..., d\}$ denotes the concentration of drug j at the dose i
- k_j with $j \in \{1, 2, ..., d\}$ is the "effectiveness" of the drug j and is estimated by means of clinical trial results

There are also some general restrictions for the above model:

Constraint1 : There is a maximum concentration for each drug $C_{max,j}$ so that $C_{ij} \leq C_{max,j}$ for *i=0,1,...,n-1*

Constraint2 : There is also a maximum of the cumulative concentration for each drug $C_{cum,j}$, i.e. $\sum_{i=0}^{n-1} C_{ij} \leq C_{cum,j}$

Constraint3 : As a side effect, there are m important organs such that the organ $k \in \{1, 2, ..., m\}$ can assume a maximum $C_{s-eff,k}$ of damage, and each drug j damages the organ k an amount of η_{kj} per concentration unit. So,

$$C_{s-eff,k} \geq \sum_{j=1}^{d} \eta_{kj} C_{ij}$$

for every i, k such that $0 \leq i \leq n-1$ and $1 \leq k \leq m$. Of course, we can multiply the parameters η_{jk} by a constant depending on k such that we can normalize $C_{s-eff,k} = 1$

Constraint4 : There is a maximum in the number of tumor cells N_{max} , so, $N_{max} \geq N(t)$ for all t

2 Aim of the exercise:

At first we want to find a curative treatment which, among the curative options, it is the most effective in the sense that it MINIMIZES the following fitness function:

$$\int_{\tau_0}^{\tau_l} N(t)dt$$

We consider that the solution is curative if $N(\tau_i) \leq 1000$ for 3 consecutive values τ_{l-2}, τ_{l-1} and τ_l with $l \in \{2, 4, ..., n\}$ (from [McC99][III])

If a curative treatment does not exist we want to find a palliative treatment that maximizes the lifespan of the patient. A palliative treatment has no dose or treatment neither at time τ_n nor later on. So, the term $\sum_{j=1}^{d} k_j C_{ij}$ in GOMPERTZ model must be taken equals to zero for $t > \tau_n$

We consider that the lifespan of the patient terminates when CONSTRAINT 4 is violated.

3 Parameters of the model:

Some of the parameters are taken from [McC99], while others are (computationally) experimental:

- $\Theta = 10^{12}$ (from [McC99])
- $\lambda = 0.336$
- $N(0) = 20000$ (from the picture in [McC99])
- Number of drugs: $d = 10$
- Parameters $\overrightarrow{k} = [k_1, ..., k_{10}]$:

$$\overrightarrow{k} = [0.12, 0.0502, 0.0637, 0.1347, 0.0902, 0.0546, 0.0767, 0.1121, 0.0971, 0.0403]$$

- Number of doses: $n = 10$, with equal separation: $T = \tau_{i+1} - \tau_i = 3$ for $i = 0, 1, ..., n-2$. So, $\tau_0 = 0, \tau_1 = 3, ..., \tau_8 = 24$ and the treatment finishes at $\tau_9 = 27$. We take $\tau_n = \tau_{10} = 33$.
- Constraint 4: $N_{max} = 0.95 \times \Theta = 9.5 \times 10^{11}$
- Constraint 1: $C_{max,j} = 15$ for all j
- Constraint 2: $C_{cum,j} = 127$ for all j
- Constraint 3: We consider $m = 4$ different organs which cannot be severally affected by the treatment with all $C_{s-eff,k} = 1$. Here we are going to use two different cases of parameters η_{kj}:

k/j	1	2	3	4	5	6	7	8	9	10
1	0.0036	0.0098	0.0061	0.0009	0.0003	0.0108	0.0045	0.0021	0.0096	0.0125
2	0.0063	0.0082	0.0062	0.0062	0.0083	0.013	0.0039	0.0019	0.0015	0.005
3	0.0129	0.0018	0.0116	0.0021	0.009	0.0129	0.0054	0.0049	0.0093	0.0066
4	0.0053	0.0086	0.0067	0.0029	0.0089	0.0054	0.0042	0.0095	0.0112	0.0092

where we have been able to find curative treatments.

The second case corresponds to:

k/j	1	2	3	4	5	6	7	8	9	10
1	0.00612	0.01666	0.01037	0.00153	0.00051	0.01836	0.00765	0.00357	0.01632	0.02125
2	0.01071	0.01394	0.01054	0.01054	0.01411	0.0221	0.00663	0.00323	0.00255	0.0085
3	0.02193	0.00306	0.01972	0.00357	0.0153	0.02193	0.00918	0.00833	0.01581	0.01122
4	0.00901	0.01462	0.01139	0.00493	0.01513	0.00918	0.00714	0.01615	0.01904	0.01564

Part II

The solution:

In this part we will firstly describe the genetic algorithm and it's pseudo-code, we will examine the question of why we need a GA for solving this problem, then we will translate the model in PART I of cancer treatment to the language of the GA, and finally we will explain our implementation of this algorithm in C, comment the obtained results for the basic implementation comment them and improve this implementation to try to get better results.

1 The genetic algorithm:

A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. John Holland introduced genetic algorithms in 1960 based on the concept of Darwin's theory of evolution; afterwards, his student David E. Goldberg extended GA in 1989.[III]

In a genetic algorithm, a POPULATION of candidate solutions (called individuals, creatures, or PHENOTYPES) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or GENOTYPE) which can be MUTATED and ALTERED.

The evolution usually starts from a POPULATION of RANDOMLY GENERATED INDIVIDUALS, and is an iterative process, with the population in each iteration called a GENERATION. In each generation, the FITNESS of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The MORE FIT individuals are STOCHASTICALLY SELECTED from the current population, and each individual's genome is modified (RECOMBINED and possibly RANDOMLY MUTATED) to form a new generation. The NEW GENERATION of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

Pseudo-code:

The following pseudo-code encapsulate the process described above of evolution in a genetic algorithm:

Algorithm 1 Classical genetic algorithm

```
BEGIN /* genetic algorithm */
  generate initial population
  compute fitness of each individual
  WHILE NOT finished DO
  BEGIN /* produce new generation */
    FOR population_size / 2 DO
    BEGIN /* reproductive cycle */
      select two individuals from old generation for mating
        /* biased in favor of the fitter ones */
      recombine the two individuals to give two offspring
      with small probability perform random mutation on the offspring
      compute fitness of the two offspring
      insert offspring in new generation
    END
    IF population has converged THEN
      finished := TRUE
  END
END
```

Why is the genetic algorithm suitable for this problem?

The space of possible solutions for this problem is very big at 2^{400}possible solutions, that's 10^{160} (for reference the estimated number of atoms in the universe is "just" 10^{80}). Furthermore the function we want to optimize $\int_{\tau_0}^{\tau_l} N_{C_{ij}}(t)dt$ is most likely not convex, therefore it is not possible to use a deterministic algorithm to search for the optimal solution. A heuristic algorithm should be used.

GAs are a heuristic solution search technique inspired by natural evolution. They are a robust and flexible approach that can be applied to a wide range of learning and optimization problems. They are relatively easy to implement and are parallelable.

2 Adapting the problem to the genetic algorithm semantics:

To be able to use the genetic algorithm for solving the cancer treatment problem, we need to define precisely what are the PHENOTYPES? the GENOTYPES? what is our FITNESS FUNCTION? how are we going to select STOCHASTICALLY the mating partners with bias towards the fittest? how are we going to RECOMBINE the mating partners to produce the OFFSPRING (the crossover operation)? and what is the MUTATION operation and it's probability?

Genotypes:

For this problem the genotypes are the variables $C_{ij} = \{0, 1, ..., 15\}$

Phenotypes:

A phenotype is the vector of dimension $d \times n$ listing all independent variables C_{ij} ; the dose concentrations for all the treatments. Taking into account CONSTRAINT 1: $C_{max,j} = 15$ for all j. So the representation of an individual is an array :

$$[C_{0,1}, C_{0,2}, ..., C_{0,d}, C_{1,1}, C_{1,2}, ..., C_{1,d}, ..., C_{n-1,1}, C_{n-1,2}, ..., C_{n-1,d}]$$

Fitness function:

For the CURATIVE solution the fitness function is : $\int_{\tau_0}^{\tau_l} N_{C_{ij}}(t)dt$

if there's no curative solution, then the fitness function for the PALLIATIVE solution would be: $1/t_f$ where $N(t_f) = N_{max}$

Stochastic selection of the mating pool:

STOCHASTIC UNIVERSAL SAMPLING:

We are going to use the STOCHASTIC UNIVERSAL SAMPLING method, this method guarantees a selection proportional to the individuals fitness. And it works as follows for a maximization problem:

Consider a population of N individuals $I_1, I_2, ..., I_N$ with finesses $F_{I_1}, F_{I_2}, ..., F_{I_N}$ and let $F = \sum_{i=1}^{N} F_{I_i}$ be the total fitness. We subdivide the interval $[0, F)$ to N intervals $[0, F_{I_1}), [F_{I_1}, F_{I_2}), ..., [F_{I_{N-1}}, F_{I_N})$.

Then, we select a random number r between 0 and $P = \frac{F}{N}$, and finally we take the numbers $Q_i = r + i \times P$ for $i = 0, 1, ..., N-1$, and if $Q_i \in [F_{I_j}, F_{I_{j+1}})$ we add I_j to the new population.

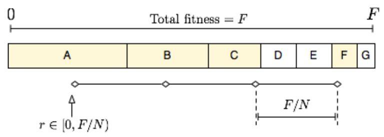


Figure 1: Visualizing the SUS method

This population will be composed of N individuals with each individual I_j been selected with probability $p_j = \frac{F_j}{F}$.

In our case we want to minimize the cost function, for this we can simply take $F_{I_j} = \frac{1}{F'_{I_j}}$ where F'_{I_j} is our fitness function for the individual I_j. And, in this case $F = \sum_{j=1}^{N} \frac{1}{F'_{I_j}}$.

Since this is one of the most crucial parts that impacts the performance of the GA, we will use the a more flexible version of the SUS where we take $F_{I_j} = \frac{a}{(F'_{I_j}+b)^c}$ whit a, b and c parameters that will enable us to tweak the selection process and try different versions that could be better adapted for this particular exercise.

TOURNAMENT SELECTION:

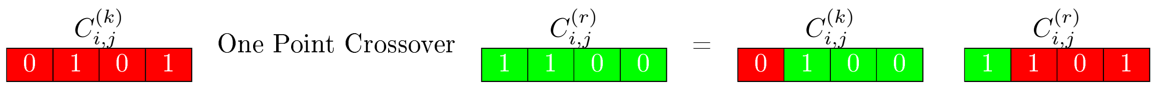
We will also use the tournament selection method for populating the mating pool. It is a simple algorithm where we select randomly and with replacement two (or n in the general case) members of the population, and we add the fittest one of the two to the mating pool with a probability p>0.5. we repeat until the mating pool is full.

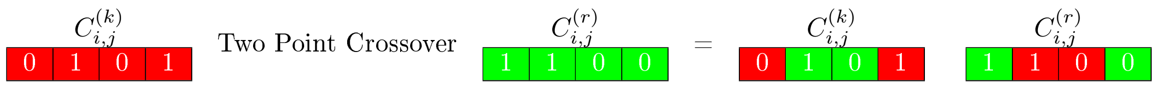
Selecting parents from the mating pool for crossover and mutation:

After constituting the mating pool, we select each time randomly without replacement two parents for the crossover and mutation.

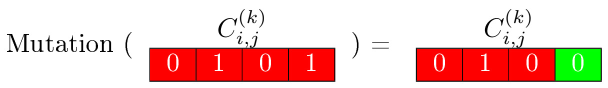
Crossover and mutation:

Considering the genotypes C_{ij} as our units for this operations, a crossover between "individual k" and "individual r" would be as follows (with point of crossover randomly chosen for each i,j / $i \in \{0, 1, .., n-1\}$ and $j \in \{1, 2, ..., d\}$):





The mutation in the case would also be at the level of C_{ij} and it would look like the following, with the mutation position chosen randomly from the 4 possible positions:



Convergence criteria:

We will consider that the population has converged when the difference between two GA solutions is less than 10^{-4}

3 Exploration vs Exploitation:

In this part we are going to explain how parameters in our program can modify the behavior of the GA in terms of exploration and exploitation. The first term refers to the ability of the algorithm to explore unknown regions of the space for possible solutions, and the second stands for it's capacity to make use of knowledge found at points previously visited to help find better points.

There are a lot of parameters that can be tuned for more exploitation or exploration, we synthesized them in the following table:

Table 1: Effect on exploitation and exploration when we increase the parameter value

Description	Name	Effect
Population size	N_Indv	increases exploration
Mutation probability	Mutation_prob	increases exploration
Maximal number of generations (iterations)	itermax	increases exploitation
Tournament selection probability	Tournament_selection_prob	increases exploitation
SUS parameter a	a	increases exploitation
SUS parameter b	b	increases exploitation
SUS parameter c	c	increases exploration
Minimal number of fittest in the population	Perc_fittest	increase exploitation

Implementation and results:

Generating random numbers:

For generating quality random numbers we used the function `ran1()` from III.

To adapt it for generating integers with module *mod* we used a simple function to divide [0,1) to *mod* equiv-distant intervals, then we situate the random number generated by `ran1()` in one of these intervals, and finally our generated random integer is this intervals order:

```c
/* This function gets as inputs a seed and interger mod
and returns an integer between 0 and mod-1 */

unsigned long my_rand(unsigned long mod, int *my_seed) {
    float p = 1.0/mod;
    double b = ran1(my_seed);
    for (unsigned long i=0; i<mod; i++) {
        if ((b >= i*p) && (b<(i+1)*p)) return i;
    }
}
```

Data structure for individuals:

A structure for storing the important information about an individual:

- The array for concentrations C_{ij}
- The fitness of the individual
- The number of copies of this individual selected for the mating pool
- And the status of the individual: curative=1 or not =0

```c
typedef struct Individual
{
    unsigned char Cij[d_par*n_par];
    double Fitness;
    int counter; // number of this individual in the mating pool
    int curative; // equal 1 if the solution is curative and 0 otherwise
} Individual;
```

Scenarios to be tested:

Scenario 1: stochastic universal sampling + One point crossover

Scenario 2: stochastic universal sampling + Two point crossover

Scenario 3: tournament selection + One point crossover

Scenario 4: tournament selection + Two point crossover

Scenario 5: stochastic universal sampling + imposing a percentage of the generation to be composed of the fittest + One point crossover

Scenario 6: stochastic universal sampling + imposing a percentage of the generation to be composed of the fittest + Two points crossover

Scenario 7: tournament selection + imposing a percentage of the generation to be composed of the fittest + One point crossover

Scenario 8: tournament selection + imposing a percentage of the generation to be composed of the fittest + Two points crossover

We will also be varying the different parameters that affects exploration vs exploitation, and this will be done for the curative case and then the palliative.

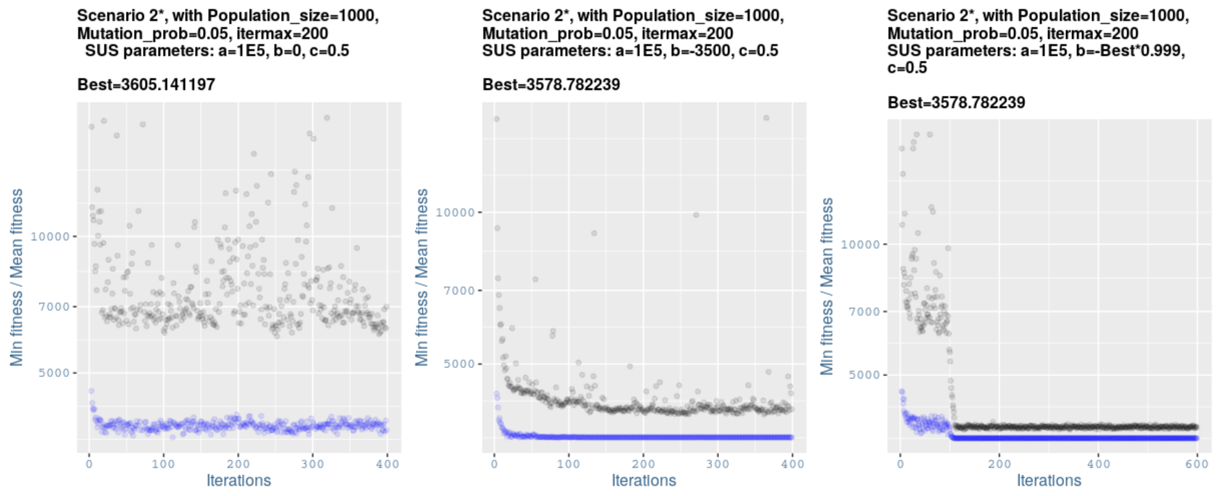
Results:

We are going to present the mean fitness (points in black) and the minimum fitness (points in blue) for the populations as the algorithm progresses from iteration to iteration.

For a good visualization, we are going eliminate big values that are usually in the first couple of generations.

The case of a curative solution:

One of the most crucial steps is to find the appropriate parameters for the function F_{I_j} in the Stochastic Universal Sampling method. Changing the parameter b we got the following results for scenario 2 with `mutation_prob=0.03`.



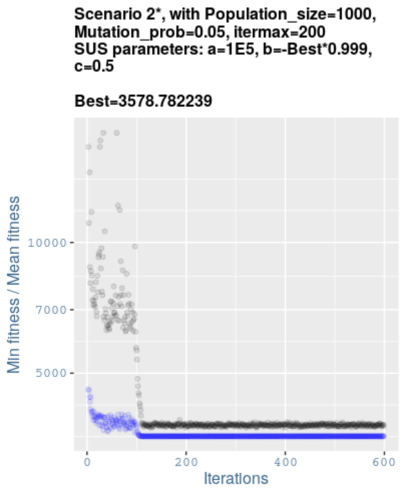
(a) `b=0`

(b) `b=-1000`

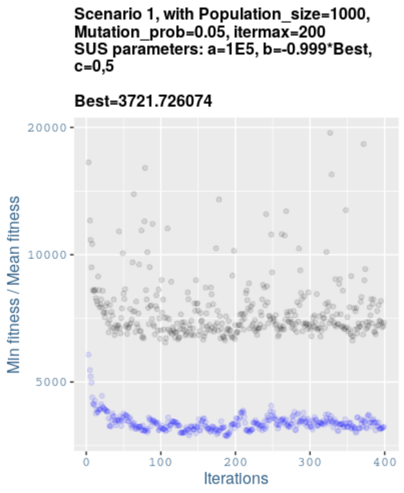
(c) `b=-3500 if Best_until_now < 3650 else b=-0.999*Best_until_now`

Figure 2: Effect of parameter b on the GA

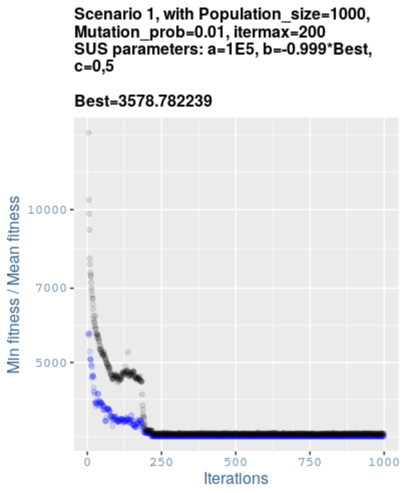
We can see clearly from Figure 2 that to push selection more towards the fittest we should take values of b that will make $F_{I_j} = \frac{a}{(Fitness_{I_j}+b)^c}$ very big near the fittest. This is not a sufficient condition to find the global minima, for example if we use the same parameters that gave the best solution until now (Fitness = 3578.78), but for scenario 1 which is different just by the crossover strategy: one point vs two points, we get a totally different result where the algorithm seem not to converge. See Figure 3:



(a) Scenario 2



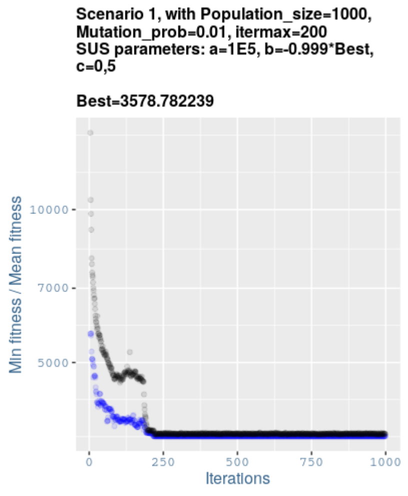
(b) Scenario 1



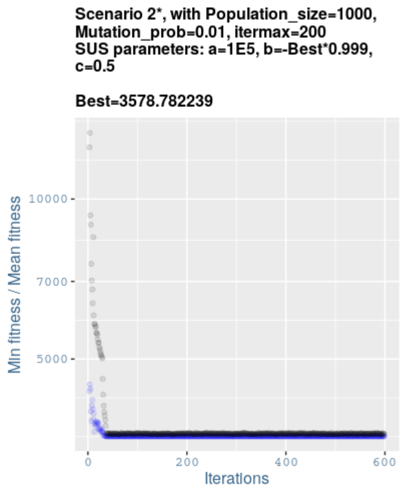
(c) Scenario 1 with Mutation_prob=0.01

Figure 3: Population_size=1000, Mutation_prob=0.05, itermax=200 a=1E5, b=-0.999*Best and c=0.5

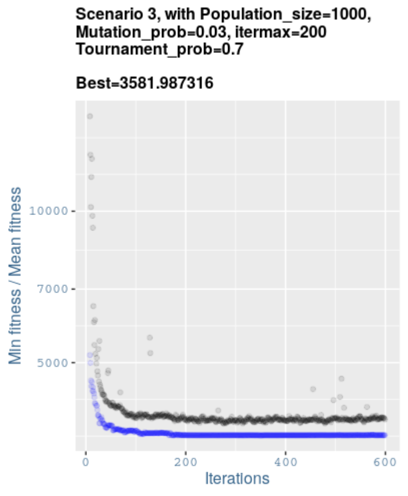
Now we present the results of the different scenarios after tuning the parameters to get a proper behavior of the GA algorithm:



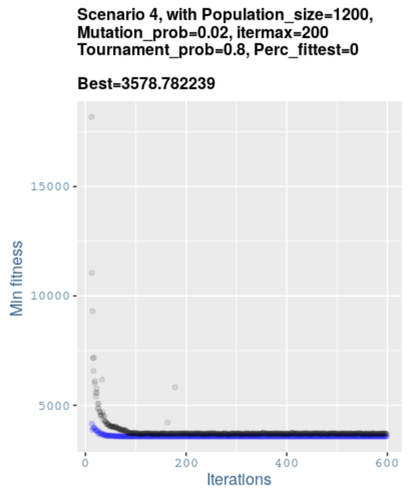
(a) Scenario 1



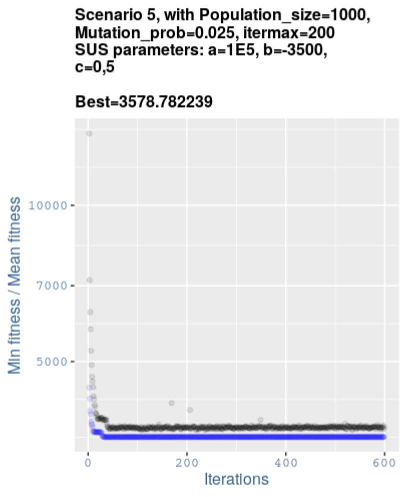
(b) Scenario 2



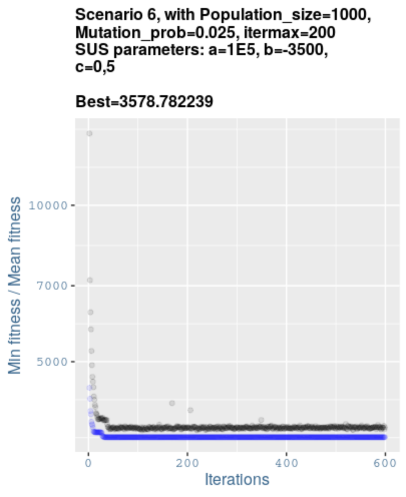
(c) Scenario 3



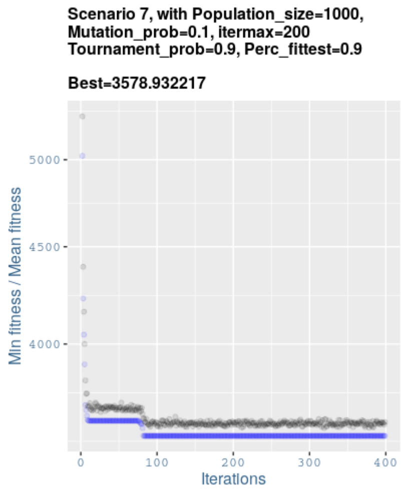
(d) Scenario 4



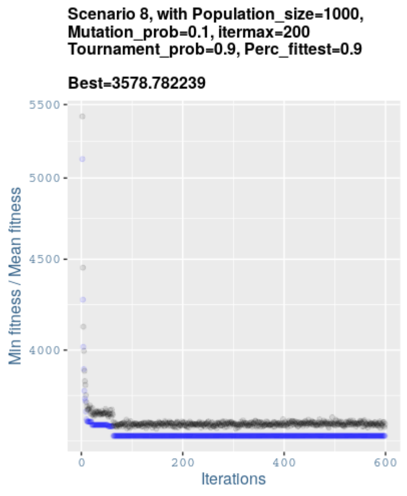
(e) Scenario 5



(f) Scenario 6



(g) Scenario 7



(h) Scenario 8

Figure 4: Results of the different scenarios on the curative case

```
Cij={15, 15, 15, 15, 15, 5, 15, 15, 15, 12, 15, 15, 15, 15, 15, 5, 15, 14, 14, 14, 14,
11, 14, 14, 15, 6, 11, 15, 15, 14, 1, 1, 6, 1, 7, 2, 2, 7, 14, 0, 15, 8, 14, 5, 0, 2,
7, 3, 14, 4, 2, 7, 15, 5, 6, 2, 14, 15, 8, 2, 4, 2, 9, 9, 1, 6, 3, 4, 10, 0, 5, 15, 8,
10, 1, 8, 0, 8, 11, 10, 14, 9, 5, 9, 13, 11, 1, 6, 0, 2, 12, 0, 12, 7, 11, 9, 11, 6, 13,
11, 13, 7, 15, 7, 10, 8, 3, 8, 5, 8,}
```

Is a solution to this first problem, it has a fitness of 3578.944512.

The second case:

We found some curative solutions for this case also, and some palliative ones.

```
Cij={15, 5, 0, 15, 6, 1, 15, 15, 14, 0, 14, 7, 2, 15, 9, 1, 13, 13, 7, 1, 13, 12, 0, 15,
0, 2, 15, 15, 12, 0, 0, 14, 0, 0, 12, 2, 3, 5, 2, 2, 15, 3, 1, 4, 2, 3, 2, 10, 8, 11,
0, 10, 0, 9, 12, 4, 4, 9, 3, 8, 2, 6, 6, 9, 8, 11, 0, 5, 10, 5, 5, 13, 2, 1, 5, 4, 2,
15, 6, 0, 4, 10, 1, 3, 0, 0, 13, 2, 7, 0, 4, 6, 2, 11, 0, 0, 2, 1, 0, 7, 10, 7, 2, 9,
11, 3, 10, 4, 5, 5,}
```

Is a curative solution for the second problem, with fitness=8174.441154, it was obtained with scenario 2 as shown in the figure. The mean fitness is not represented as it was too big:

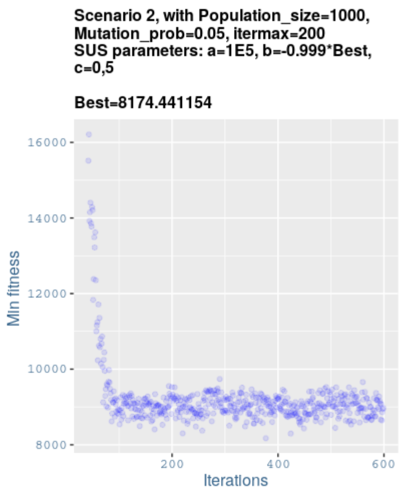


Figure 5: Curative solution for the second problem

We also found some palliative solutions to this problem, for example:

```
Cij={2, 7, 4, 2, 2, 9, 5, 6, 14, 10, 12, 12, 7, 0, 0, 0, 10, 0, 4, 9, 5, 8, 9, 1, 15,
8, 2, 0, 4, 6, 3, 5, 12, 0, 10, 0, 0, 3, 5, 7, 6, 4, 5, 1, 3, 12, 5, 13, 2, 7, 5, 2, 2,
2, 0, 4, 10, 10, 13, 1, 11, 9, 9, 13, 1, 1, 10, 11, 8, 0, 9, 0, 9, 14, 0, 9, 10, 1, 3,
4, 4, 2, 10, 13, 6, 9, 14, 9, 1, 9, 15, 12, 5, 15, 10, 0, 15, 15, 3, 0, 13, 1, 7, 1, 8,
2, 4, 1, 9, 2,}
```

Is a palliative solution with fitness=1/t=0.019075, it was obtained by scenario 7:

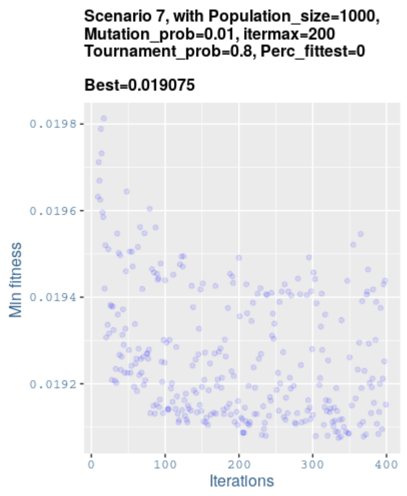


Figure 6: Palliative solution to the second problem

Part III

Conclusions:

Scenarios with the modified Stochastic Universal Sampling method, proved to be more flexible and adaptable to our problem, the main idea was to try to make the selection biased towards the minimal value. This could be done easily at first when the value are spread on large intervals, but as the population evolve, it's variance become more and more small, and it becomes difficult for the static SUS method to select the best individuals.

Two solutions to this problem where implemented:

1. Making a percentage of the population equal to the fittest individual that was discovered until present, this is expressed in scenarios 3,4,5 and 6.

2. Making the SUS method dynamic by changing the parameter $b \begin{cases} constant & while\ far\ from\ the\ minimum \\ -0.999 * Best & when\ close\ to\ the\ minimum \end{cases}$.

   This way we skew the selection heavily towards the minimum. The risk here is that if we estimate the minimum badly this approach will likely drift the GA towards a local minimum. Trial and error is inevitable.

Two points and one point crossover are equivalent when we tune the other parameters to find the best that goes with the crossover strategy.

The mutation probability is one of the most important parameters, it was it and the parameter b for the SUS function, the two most influential parameters that had to be tuned a lot.

When checking the performance of the functions in the algorithm we found that `ran1()` was the bottle neck of the program consuming 84% of the execution time. Therefore, for better performance one should try to choose a more optimized random generator with the least cost on the quality of the numbers generated.

System used:

The system used to execute the program is:

- Memory : 8 GiB
- Processor : Intel® Core™ i7-8550U CPU @ 1.80GHz $\times$ 8
- OS : Ubuntu 18.04
- GCC : gcc version 7.4.0
- Command for compiling : gcc -o Genetic_bit Genetic_bit.c RKF78.c ran1.c CrossoverAndMutation.c Fitness.c -lm

References

- Wikipedia contributors. "Genetic algorithm." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 31 Jan. 2020. Web. 11 Feb. 2020.
- Genetic algorithms for modeling and optimization, John McCall, School of Computing, Robert Gordon University, Aberdeen, Scotland, UK.
- Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P. - Numerical recipes in C_ the art of scientific computing-Cambridge University Press (1997)
- T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, C. Whitley, A Comparison of Genetic Sequencing Operators
- An Overview of Genetic Algorithms : Fundamentals Parts 1 and 2, David Beasley Department of Computing Mathematics, University of Cardi , Cardi , CF2 4YN, UK David R. Bull y Department of Electrical and Electronic Engineering, University of Bristol, Bristol, BS8 1TR, UK Ralph R. Martin z Department of Computing Mathematics, University of Cardi , Cardi , CF2 4YN, UK