University of Technology of Belfort-Montbéliard

AI50 Project

Time Table Scheduling

Developed by :

CHAUSSON	Thibault
EL HAJ HISSEIN	Olivier
GUINOT	Jossua
METALLAOUI	Nassim
VIGUIER	Léo

Supervised by : GAUD Nicolas & LAURI Fabrice & DRIDI Mahjoub

January 7, 2024

Abstract

Time Table Scheduling is a category of Scheduling in which involves generating a formatted schedule for a particular organization. For universities, timetable management is a complicated issue which uses up a great deal of human resources and is therefore very costly hence the need for a more efficient approach. The goal of this project is to automate the creation of an optimized university timetable for University of Technology of Belfort-Montbéliard (UTBM), by considering the list of hard and soft constraints offered in a semester, so that no conflict is made in these assignments. In order to achieve this aim, metaheuristics, such as population-based methods like genetic algorithms, will be used. You can find this project on GitHub.

Keywords Timetabling, Scheduling, Constraints, Metaheuristics, Genetic Algorithms.

Contents

1 Presentation

1.1 Project context

When it comes to academic institutions, especially universities, efficient timetable management is an indispensable aspect of ensuring a smooth and productive learning environment. Timetables dictate the scheduling of classes, exams, and various academic activities, and the challenge lies in creating schedules that not only adhere to strict constraints but also optimize resource allocation. Traditional manual approaches to timetable scheduling consume substantial human resources and financial investments.

For the University of Technologie of Belfort-Montbéliard, this challenge is particularly pronounced. UTBM, like many universities, faces the need to manage complex timetables that take into account a wide range of hard and soft constraints. Hard constraints include factors such as classroom availability, professor availability, and student's choices, which must be strictly adhered to. Soft constraints, on the other hand, refer to preferences and priorities that, while not absolute, should be considered to enhance the quality of the timetable, for example by minimizing student travel times.

In response to these challenges, the goal of this project is to automate the creation of optimized timetables at UTBM. The focus is on minimizing assignment conflicts and assigning as many class sessions as possible to time slots and rooms. To achieve this, etaheuristics will be used, such as Genetic Algorithms.

By automating the timetable creation process and incorporating advanced optimization methods, the aim is not only to save valuable human resources but also to enhance the way timetables are managed at UTBM.

1.2 State of the art

This section reviews the state of the art of some of the works relevant to the literature search carried out in this project.

- **Rossi-Doria et al. (2002)**[6] The goal of the study is to compare the performance of five different metaheuristics on the university course timetabling problem. They implemented Evolutionary Algorithms, Ant Colony Optimization, Iterated Local Search, Simulated Annealing and Tabu Search. The comparison was made under fair conditions with a common solution representation and local search operator.

The results suggest that there is no single best metaheuristic for all timetabling instances, and even for similar instances, the best metaheuristic may vary. The study highlights the challenge of selecting the most suitable metaheuristic for specific timetabling problems. Their study showed that conventional, generic, genetic algorithms perform poorly, and suggested that they should be enhanced with domain-specific knowledge to produce better results for these specific problems.

- **Feizi Derakhshi et al. (2012)**[1] The article highlights the challenges in solving the University Course Timetabling Problem (UCTTP), as it is an NP-hard problem with exponential growth in complexity due to various constraints and changing parameters. Finding an efficient way to satisfy soft constraints is a major challenge. The article classifies the approaches for solving the UCTTP into three categories:

  1. **Operational Research Methods:** This category includes techniques based on graph coloring theory, integer programming/linear programming (IP/LP), and constraint-based techniques.
  2. **Meta-heuristic Methods:** Meta-heuristic approaches involve techniques like genetic algorithms, Ant Colony Optimization, memetic algorithms, and simulated annealing.
  3. **Modern Intelligent Methods:** This category includes a variety of approaches based on artificial intelligence, fuzzy theory, and combinatorial methods.

  The article highlights that operational research-based methods, while simple to implement with integrated software, lack efficiency. On the other hand, Meta heuristic algorithms and modern intelligent techniques are more efficient for analyzing UCTTP. However, there is no one-size-fits-all best method, as the efficiency depends on various factors like dataset variations and the way these methods are applied.

- **Ahmed Redha Mahlous (2023)**[3] To address the challenge that universities face in distributing and allocating students to their required classes, the article proposes a genetic algorithm (GA) that assigns students to classes based on their preferences and implements various metaheuristic concepts and tailored genetic operators to enhance its performance. The article gives a detailed description of the design and choices made for the genetic algorithm, including the choice of chromosome representation. The results obtained were promising and the algorithm guarantees the feasibility of solutions as well as satisfying more than 90% of student preferences even for the most complex problems.

- **Manar I. Hosny (2011)**[2] The article discusses the university timetabling problem, which involves scheduling courses and exams by assigning them to specific time slots, rooms, and other resources while adhering to various hard and soft constraints. Genetic Algorithms (GAs) are introduced as a popular meta-heuristic technique used to address such combinatorial optimization problems. We can find various GA techniques that have been recently used to address different variants of the university timetabling problem, resulting in promising results. It briefly describes the overall technique, focusing on the chromosome representation and the crossover and mutation operators Different forms of GAs have been developed, including Steady State Genetic Algorithms (SSGA), Enhanced Steady Genetic Algorithms (ESSGA) with fuzzy logic-enhanced operators, and Simple Genetic Algorithms (SGA) with non-overlapping populations.

2 Functional scope

2.1 Available data

We were able to retrieve the timetables for each FISE, allowing us to list the dates, types of classes, and rooms used by a UV (Unité de Valeur). Therefore, we created a .csv file in the following format:

- UV Name
- Type
- Room
- Start Time
- End Time Day

Additionally, we have in our possession the list of rooms by site:

- Room Name
- Room Type
- Available Equipment
- Number of Seats

A file with the list of UVs:

- UV Code
- UV Title
- Number of Lecture Hours
- Number of Tutorial Hours
- Number of Practical Hours
- Type of Training

Thus, with extensive data extraction and structuring, we have created a substantial dataset.

2.2 Minimum criteria to meet

One of the advantages of this project is that we can implement it step by step and make it more complex incrementally. Initially, we will focus on meeting the following criteria:

- Assign each student to the UVs they request
- Assign each class session to a room
- Ensure compatibility of students' schedules among their different UVs
- Ensure compatibility of a teacher's schedule based on the classes they teach

Thus, with these criteria, we can ensure the creation of viable and usable timetables.

2.3 Optional criteria

Next comes a set of more complex criteria to implement:

- Prioritize external speakers
- Respect the half-days requested by teachers
- Provide a minimum of 45-minute lunch break

2.4 Constraints

To relax certain constraints, we can work on a timetable over a two-week sliding period.

2.4.1 Hard constraints

1. An instructor can be assigned to only one lecture session at the same time slot.
2. A classroom can accommodate only one lecture session.
3. A classroom must have sufficient capacity.

2.4.2 Soft constraints

1. Avoid gaps in the timetables.
2. Avoid having some days overloaded while others are less so.
3. Minimize student movements within the establishment.

2.5 Function

A priori, it is quite challenging to find good methods for crossover, selection, and mutation, as each has its own set of advantages and disadvantages. Therefore, we have decided to list all the methods that we can consider as classical, and we will compare these methods to find the best one for our case. This approach is similar to that of these studies[4, 5], which determine the most effective method by achieving optimal fitness. To validate a crossover function, we must check if it generates a large number of invalid individuals.

2.5.1 Chromosomes

First of all, we have to define the shape that will take the chromosomes in our algorithm, so that we know how each solution can be altered to produce new ones.

In our case, the problem consists in finding, for an ensemble of classes, a classroom and timeslot such that all constraints are verified. Our chromosomes will therefore be an array of classes, with each cell of the array a gene containing a room number and a timeslot.

2.5.2 Fitness

To evaluate the quality of a chromosome, we use a fitness function, which takes its genes in input and outputs a number, that represents the constraints that the chromosome fails to satisfy. Therefore, the lower the fitness, the better the chromosome.

For each constraint, we calculate the fitness using the following formula:

$$F(constraint) = \sum_{elements} \sum_{k=0}^{conflicts} k$$

We can then multiply each of these numbers with a weight, determined and adjusted empirically, which leads us to this formula:

$$F = \sum(weight_i \times F(constraint_i))$$

With this formula, we ensure that the chromosomes who satisfy most constraints but completely fail on a few of them will end up with a bad fitness, which will help the algorithm come up with balanced solutions.

Since in this problem we have both hard (classroom capacity, teacher availability) and soft (student satisfaction, repartition of classes within the week) constraints, we'll use two different formulas for fitness. If the hard constraints are not all fullfilled, we ignore soft constraints. If they are, then we calculate fitness only for soft constraints. This will ensure that the algorithm will treat hard constraints as a priority.

Since the algorithm will sometimes be comparing strong and weak fitnesses within the same population, we have to make sure it always considers the chromosomes for which the strong fitness was perfect as better than those for which it is not.

To this end, we decided to repsesent strong fitness by a negative number. When calculating strong fitness, we start at 0, then remove points for each gene that breaks one or several strong constraint, such as overlapping with a course from the same UV, or being scheduled in an unavailable classroom.

If the strong fitness computation results in a fitness equal to 0, then we compute the weak fitness.

The weak fitness is a positive number computed for each student of set. It starts at a fixed amount, and then for each course with an overlapping schedule, we remove points.

Since we remove 1 points for the first conflict, 2 for the second and so on, the worst case scenario is obtained with the combination formula for the number of pairs in a set of size n :

$pairs = \frac{n(n-1)}{2}$

Since we have 6 UVs, that gives us $pairs = \frac{6\times5}{2} = 15$ distinct pairs.

The maximum number of points deduced from the fitness is therefore the sum of all natural integers lesser or equal to 15, expressed $\sum_{k=0}^{n} k$, or using the Gauss formula : $\frac{n(n+1)}{2}$

That gives us a maximum fitness loss of $\frac{15\times16}{2} = 120$. The base fitness score will therefore be 120, and then be reduced for each conflict, down to 0 in the worst case.

To make data easier to interpret, we divide the sum of all the students' fitnesses by 1.2, then by the number of students, and end up with a final result between 0 and 100.

Due to repeated conflicts weighing more, the profile of the function $fitness = f(numberofconflicts)$ is akin to a logarithmic curve, which means that the more the algorithm progresses towards a "perfect" solution, successive improvements in fitness tend to get smaller and smaller.

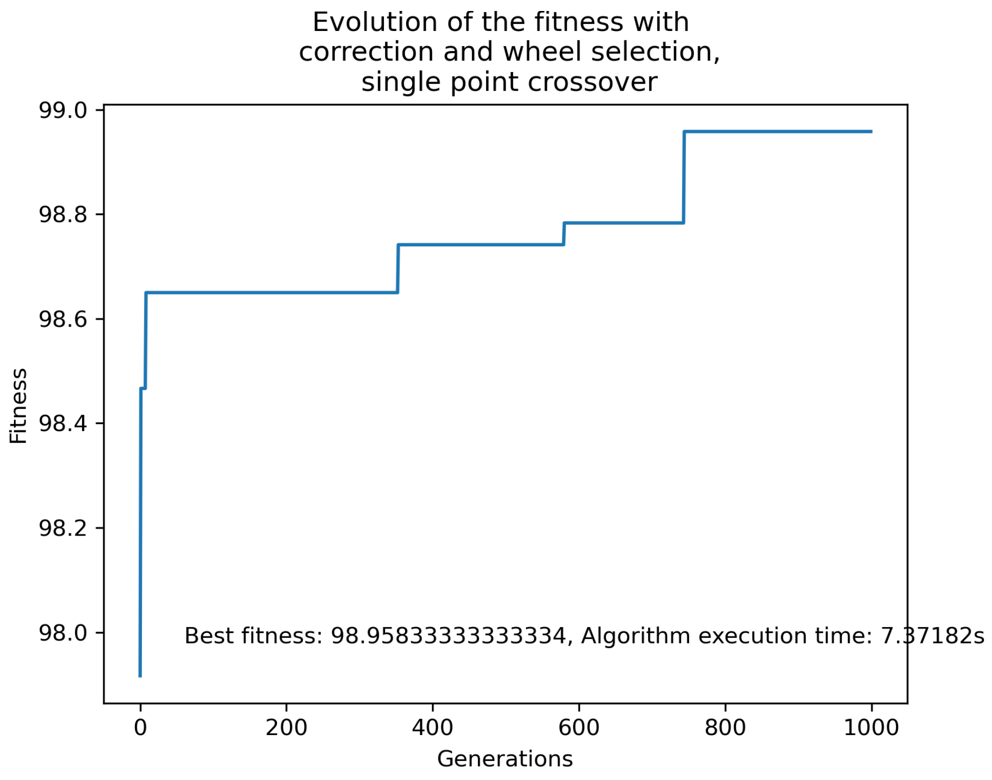

Image 1: The evolution of the fitness over 1000 generations

Since the fitness function is called everytime a new chromosome is generated, and since it needs to compare every possible couple of UV, classroom and students, which there are hundreds of, it leads to a very high computing complexity.

As with any optimization problem, the quality of the solution relies heavily on the number of generations we're able to run in the imparted time. For this reason the optimization of the fitness function was one of the main challenges of this project. We had to rewrite the functions several time to avoid redundancies as much as possible.

For example, to compute the fitness related to students, we used to iterate for every course of every UV of every student.

Since it would mean checking several times the compatibility of the same pairs of UVs, we changed it to compute first a dictionnary of all the UVs who have incompatible courses, and then use it for every student.

It allowed us to reduce the weak fitness computation time by almost 90 percent.

Since the fitness function defines which mutations and crossover will be kept during the selection, we theoretically only need to modify it to have the algorithm obey to any constraint we might want.

If we were to keep working on this project, we could modify the fitness function to ensure that students and teacher get enough time for a lunch break, that they only get lessons on one campus per day, etc.

2.5.3 Selection

In the process of evolving from one generation to the next, reproduction plays a crucial role. This reproduction must necessarily occur between individuals who are best adapted to their environment. Therefore, a selection operation is essential for identifying the chromosomes most capable of ensuring an improvement in the quality of solutions.

- The roulette wheel method is a selection technique that assigns each individual a probability of being selected proportional to their performance. Thus, an individual with a high evaluation has a greater chance of being chosen, while an individual with a lower evaluation is less likely to be selected. In our case we would minimize evaluation function: with P_p individu evaluation and P_{S_p} probability to pic p individu.

  $$P_{S_p} = \frac{\frac{1}{F_p}}{\sum_p \frac{1}{F_p}}$$

  The roulette wheel method relies on a wheel divided into a number of sectors equal to the number of chromosomes in the population. The area of each sector is proportional to the evaluation of the corresponding individual. When the wheel is spun, the stopping position indicates which individual is selected for reproduction. A major drawback of the roulette wheel method is the likely presence of a "Super Hero," i.e., an individual whose probability of selection is significantly higher than that of others. This can be problematic as this individual is likely to be consistently chosen, thereby limiting the scope of exploration and the genetic diversity of the population.

- Tournament selection is a method that relies solely on comparisons between individuals, thus eliminating the need to sort the entire population. To select an individual, "t" are chosen uniformly from the population, and the best among them is selected. The advantage of this method is that it removes the risk of having a "Super Hero", as is the case in roulette wheel selection. However, its major drawback is that the best individual in the population may not be selected, thereby limiting the scope of exploration.

2.5.4 Crossover operators

To create new chromosomes from the best ones of the best generation, we'll defined a crossover operator. While its exact definition will be iterated upon during development, according to the results we get and to the data structure we use, here are some examples of crossover operators we can use :

- Single point crossover (or n-point crossover) : choose n point on both chromosomes to divide them into segments, then choose alternating segments from each chromosome into the new one.
- Uniform crossover : Each gene has as an even chance of being taken from either parent
- Order crossover : Take a sub-sequence from one of the parents, the fill in the gaps with genes from the second one.

2.5.5 Mutation operators

To avoid local minimums, we'll use a mutation operator who takes in a single chromosome and changes a single randomly selected gene, using a predefined function. Here are examples of such functions for this problem :

- Randomly choosing the timeslot of a lesson
- Randomly choosing the classroom of a lesson
- Randomly choosing the day of a lesson
- Swapping the classrooms and timeslots of two lessons

2.5.6 Correction

As a result of the complexity of the problem and the amount of constraints, the probability of a newly generated solution satisfying the minimal constraints is very low, even if it's derivated from chromosomes with a good fitness score. To reduce the number of 'useless' solutions, we can apply a correction function to newly created chromosomes. This function will search for trivial solutions to conflicts, and apply them if they exist. For example, if two lessons take place in the same classroom on the same timeslot, therefore creating a conflict, the correction function can find a suitable empty classroom to which move on of the lessons. It will be necessary to experiment with different degrees and types of correction to determine if they positively affect the performance of the program.

2.5.7 Additional ideas

To avoid local minimums, we kill the weakest half of each generation, then replace these $n/2$ individuals by randomly generated ones.

2.6 UML diagram

Here is a representation in the form of a UML diagram of all the data that we will use to generate a schedule.

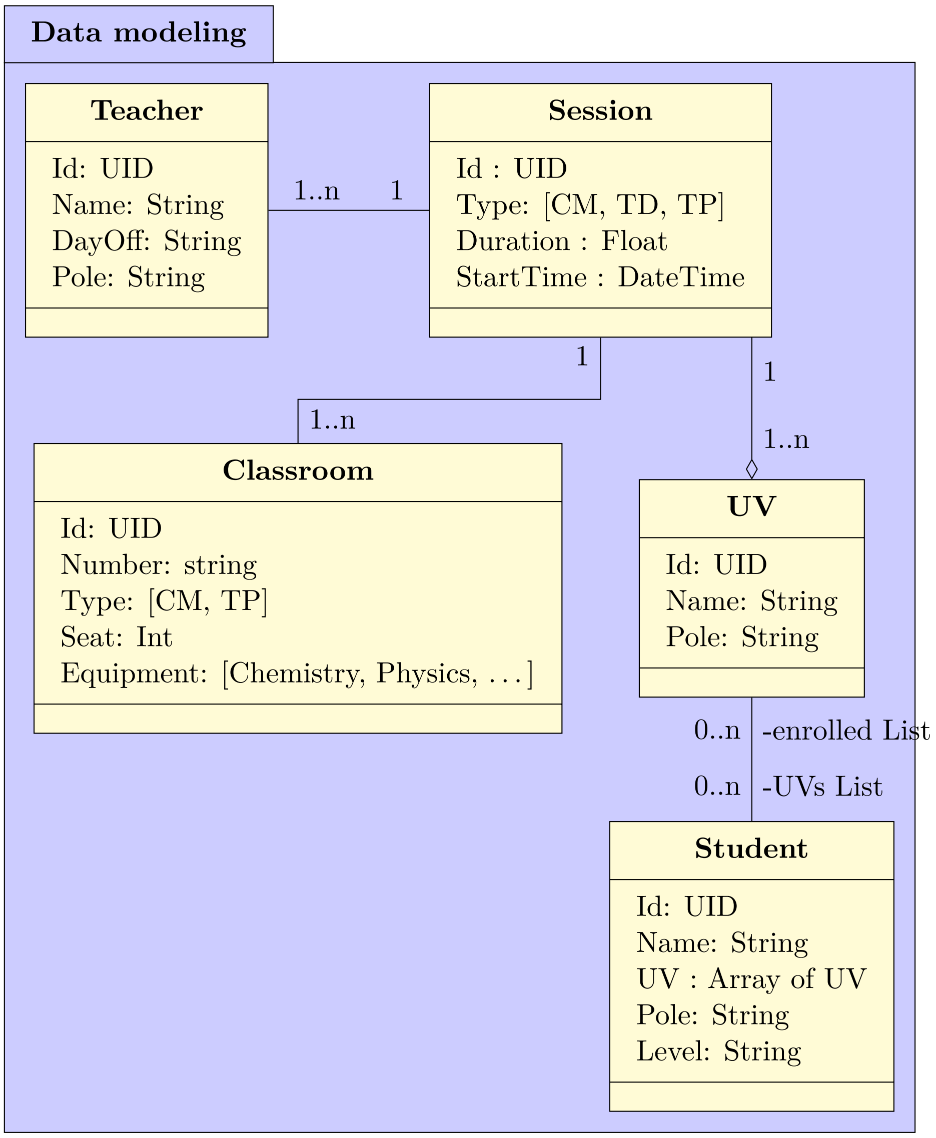

Image 2: UML data

2.7 Data structure

2.7.1 First idea

According to the previous UML diagram, we can model the individuals as follows. Let the set $I = \mathbb{M}_{\text{nbWeeks},\text{nbHours}} \times \mathbb{M}_{1,3}m$ Let $I \in I$ with $m \in \mathbb{N}$ number of parallel courses such that:

$$I = \begin{array}{r|c} & \xrightarrow{\text{Hours} \times 6 \text{ days}} \\ & \begin{array}{ccccc} 8-10 & 10-12 & \cdots & 16-18 & 18-20 \end{array} \\ \text{Weeks}\downarrow \begin{array}{c} 1 \\ 2 \\ \vdots \\ 31 \\ 32 \end{array} & \begin{pmatrix} Session_{1,1} & Session_{1,2} & \cdots & \cdots & \cdots \\ Session_{2,1} & Session_{2,2} & \cdots & \cdots & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ Session_{31,1} & Session_{31,2} & \cdots & \cdots & \cdots \\ Session_{32,1} & Session_{32,2} & \cdots & \cdots & \cdots \end{pmatrix} \end{array}$$

Where the matrix $Session \in \mathbb{M}_{m,1} \times \mathbb{M}_{1,3}$ with $m \in \mathbb{N}$ number of parallel courses :

$$Session_{1,1} = \begin{bmatrix} \begin{bmatrix} Classroom \\ Teachers \\ Students \end{bmatrix} & \begin{bmatrix} B425 \\ Lauri \\ [1, 15, 86, \cdots] \end{bmatrix} & \begin{bmatrix} B315 \\ Gaud \\ [10, 11, 66, \cdots] \end{bmatrix} & \cdots \end{bmatrix}$$

Thus, we can define our population P of $n \in \mathbb{N}$ individuals such that $P \in \mathbb{P}_{n,1}$ as follows:

$$P = [I_1, I_2, \cdots, I_n]$$

The main flaw of this apporach is that it makes implementing crossover operators very complex, since taking timeslots from both parents would often result in schedules containing several instances of the same lesson, and none of some others.

2.7.2 Second idea

Where the matrix $P \in \mathbb{M}_{m,1} \times \mathbb{M}_{1,5}$ with $m \in \mathbb{N}$ number of sessions across all UVs :

$$P = \begin{bmatrix} \begin{bmatrix} Classroom \\ StartTime \\ Duration \\ Teacher \\ UV \\ Type \end{bmatrix} & \begin{bmatrix} B425 \\ Monday\ 10:00 \\ 2\ hours \\ Lauri \\ AI53 \\ CM \end{bmatrix} & \begin{bmatrix} B315 \\ Tuesday\ 14:00 \\ 1\ hour\ 30\ minutes \\ Gaud \\ AI51 \\ TP \end{bmatrix} & \cdots \end{bmatrix}$$

The population matrix P of $n \in \mathbb{N}$ individuals such that $P \in \mathbb{P}_{n,1}$ as follows:

$$P = [Session_1, Session_2, \cdots, Session_n]$$

The main advantage of this data structure is that since it is a vector, we can use the canonical crossover operators on it, because all the chromosomes will contain the same number of items, and always in the same order :

$$P_3(P_1, P_2) = [P_1Session_1, \cdots, P_1Session_m, P_2Session_{m+1}, \cdots, P_2Session_n]$$

3 Genetic algorithm

To solve this type of problem, population-based algorithms have empirically shown that they are very good candidates, hence we have turned to the genetic algorithm.

3.1 Algorithm

To get a more concrete idea of the algorithm used, we can refer to the following flowchart.

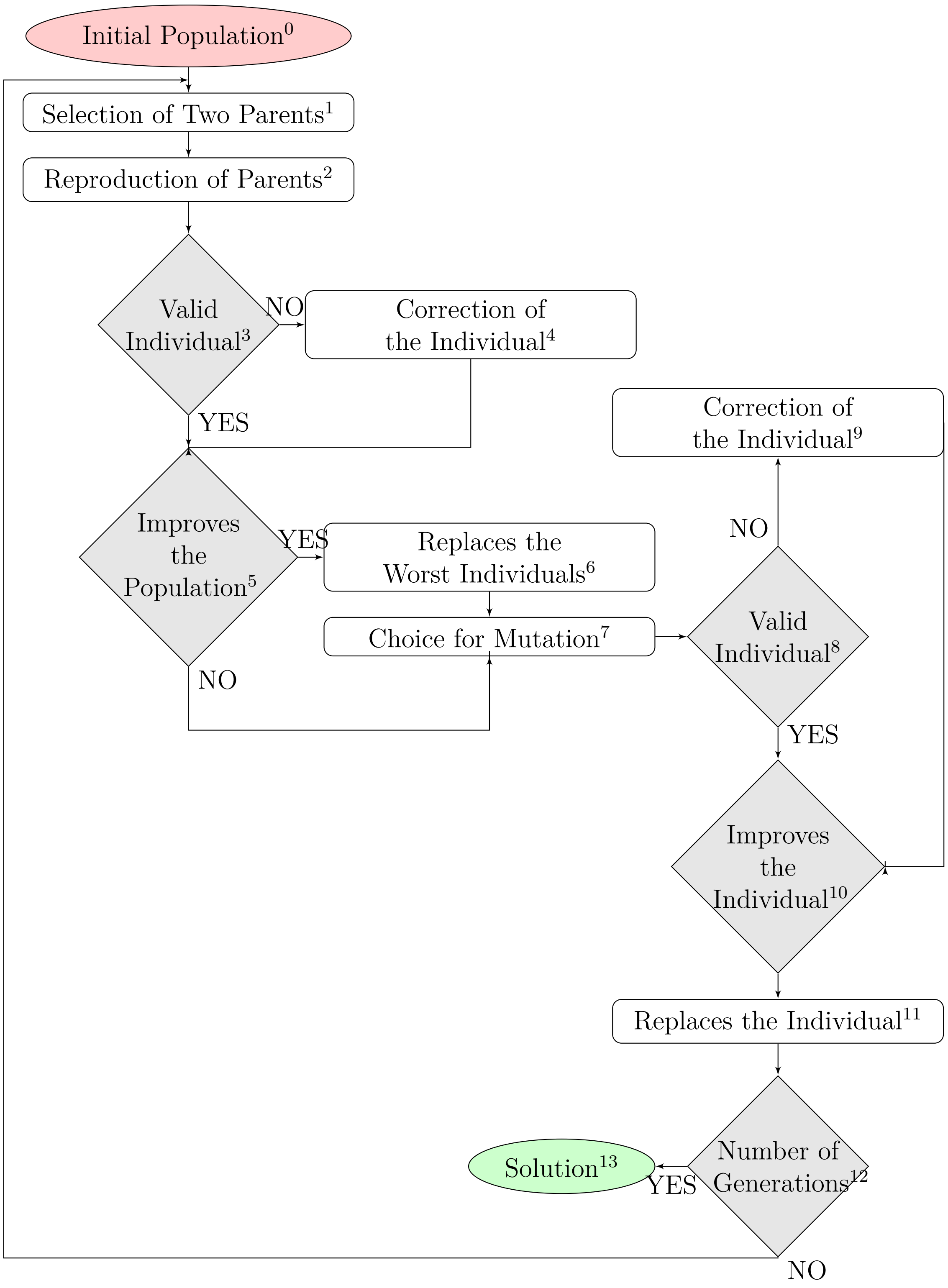
Initial Population^0
Selection of Two Parents^1
Reproduction of Parents^2
Valid Individual^3
NO
Correction of the Individual^4
YES
Improves the Population^5
YES
Replaces the Worst Individuals^6
NO
Choice for Mutation^7
Correction of the Individual^9
NO
Valid Individual^8
YES
Improves the Individual^{10}
Replaces the Individual^{11}
Solution^{13}
YES
Number of Generations^{12}
NO

Image 3: Algorithm

3.2 Program architecture

3.2.1 Initial data

To carry out this work, the UTBM provided us with very little data, we have in our possession the list of UVs by site, the rooms by sites, the timetables of each branch. Furthermore, by automatically retrieving information from the UV Guide, we were able to retrieve the person in charge of each UV, as well as other information. This lack of data forced us to invent students and assume that the UV's responsible is the UV's teacher, moreover, we do not have access to the particular requests of each teacher.

3.2.2 Initial population

In a genetic algorithm, generating the initial population is the first step, and also one of the most important. The important thing here is to generate a population that will converge towards an optimal solution quickly, while avoiding falling into a local minimum. When generating the population, we need to focus on 2 main points: constructing individuals that are very different from each other, in order to explore as much of the solution space as possible, and validating as many constraints as possible, if not all of them. For the first point, we ensure that each assignment is random. For example, the choice of teacher (if there is more than one in a course), room, start time and the day of the UV are determined randomly.

Then, for the second point, in our case it is very important to validate the course overlap constraints, so as not to lose time when running the genetic algorithm. To achieve this, 3 functions help in the construction of an individual, by generating, from a chromosome, a matrix representing the timetable of a teacher, a UV, or a room. This matrix should be binary in the best case, but if it is not and contains numbers greater than 1, this means that an overlap has occurred and therefore that this solution is not valid. Each time a random choice is made, we check that the time window chosen by the algorithm for an UV is empty to avoid overlaps.

3.2.3 Fitness

The implementation of fitness follows the functional scope section (see section: 2.5.2). Fitness will be used in the following parts:

- Sorting of the population (section: 3.2.4) to sort it in ascending order of fitness

- In the selection of parents (section: 3.2.5)
- To validate or reject offspring (section: 3.2.9)

3.2.4 Shorted population

We made the choice to sort the initial population in descending order of fitness for several reasons. This is time-consuming at first, but it will allow for a much more efficient algorithm later on.

Once the population is sorted, this allows us to quickly find the best schedule that the program has generated after a predefined number of generations.

From the point of view of parent selection, this makes it easy to find the elites without having to re-sort the entire population each generation.

Managing a sorted initial population has numerous advantages but complicates the acceptance of a child (see child acceptance: 3.2.9). If a child is accepted, we must insert it into the sorted population without altering its ordering. That's why we developed a function allowing for a dichotomic insertion.

Algorithme 1 Dichotomic Insertion

```
Require: $arr, x$
Ensure: Index where to insert $x$ in $arr$
 1: $start \leftarrow 0$
 2: $end \leftarrow lengthofarr$
 3: while $start < end$ do
 4:     $middle \leftarrow (start + end)//2$
 5:     if $arr[middle] < x$ then
 6:         $end \leftarrow middle$
 7:     else
 8:         $start \leftarrow middle + 1$
 9:     end if
10: end while
11: return $start$
```

With this dichotomic insertion, we know where to insert the offspring resulting from the crossover of its two parents. We move from a complexity which is generally in $O(n^2)$ to a complexity in $O(\log(n))$[1].

[1]Initially, we consider the number of comparisons as the measure of complexity. Let

Besides, its temporal complexity in $O(\log(n))$, we have a spatial complexity: which we can qualify as in-place.

3.2.5 Selection parents

To select parents, we will use the fact that the initial population is sorted to facilitate their selection. In order to compare the different heuristics as effectively as possible, we have implemented four different parent selections, each with their own advantages and disadvantages.

Tournament selection

It works by organizing a "tournament" among a predetermined number of individuals randomly chosen from the population. The winner of the tournament, that is, the individual with the best fitness, is selected for reproduction. This process is repeated until the desired number of individuals is selected.

Algorithme 2 Tournament selection

```
Require: $sorted\_list\_fitness, t\_size, nb\_couple$
Ensure: List of indices of selected parents
1: $tournament$ ← random sample of size $t\_size$ between 0 and length of sorted_list_fitness
2: Sort $tournament$
3: $list\_index\_parents$ ← first $nb\_couple \times 2$ indices of $tournament$
4: Shuffle $list\_index\_parents$
5: return create_couples($list\_index\_parents$, $nb\_couple$)
```

Advantages:

1. Control of selective pressure: The size of the tournament can be adjusted to control selective pressure. A larger tournament favors high fitness individuals, while a smaller tournament reduces selective pressure.

2. Genetic diversity: Allows for maintaining greater genetic diversity than elitist selection, as even individuals with lower fitness have a chance of being selected.

$T(n)$ be the number of comparisons made for an instance of size n. Then the number of comparisons T satisfies the following recurrence: $T(n) = 1 + T(n/2)$. According to the master theorem, this recurrence has a solution in the form $T(n) = O(\log(n))$. Finally, the number of comparisons is linear in the number of operations performed; thus, the algorithm has a logarithmic complexity.

3. Ease of implementation: Relatively simple to implement and does not require knowledge of the entire population, just a sample for each tournament.

Disadvantages:

1. Risk of premature convergence: With a large tournament size, the method can lead to premature convergence, similar to elitist selection.
2. Random selection: The random component can sometimes select less fit individuals, potentially affecting the quality of the next generation.
3. Dependence on tournament size: The performance of this method is strongly linked to the size of the tournament. A poor choice of size can either slow convergence or reduce genetic diversity.

Roulette wheel selection

In this method, each individual in the population has a selection probability proportional to their fitness. Imagine a wheel where each segment is proportional to the fitness of each individual. By “spinning“ this wheel, individuals are chosen randomly, but with a probability linked to their performance.

Algorithme 3 Roulette wheel selection

```
Require: $list\_fitness, nb\_couple$
Ensure: List of indices of selected parents
 1: $total\_fitness \leftarrow$ sum of $list\_fitness$
 2: $proba\_list \leftarrow$ probability of each individual
 3: $parents \leftarrow []$
 4: for _ ← 1 to $nb\_couple \times 2$ do
 5:     $spin \leftarrow$ random number between 0 and 1
 6:     $fitness\_sum \leftarrow 0$
 7:     for $i \leftarrow 0$ to length of proba_list do
 8:         $fitness\_sum \leftarrow fitness\_sum + proba\_list[i]$
 9:         if $fitness\_sum \geq spin$ then
10:             Add $i$ to $parents$
11:             break
12:         end if
13:     end for
14: end for
```

```
15: return create_couples($parents$, $nb\_couple$)
```

Advantages:

1. Probability proportional to fitness: Individuals with higher fitness have a greater chance of being selected, favoring beneficial traits.
2. Population diversity: Unlike elitist selection, there is always a chance for individuals with lower fitness to be selected, maintaining some genetic diversity.
3. Balance between exploration and exploitation: Allows a good balance by not only favoring the most efficient individuals, but also giving a chance to others.

Disadvantages:

1. Slow convergence: The method can be slower to converge to the optimal solution compared to more direct methods like elitist selection.
2. Low selective pressure: If the fitness difference between individuals is small, the selective pressure is also low, which can lead to slow convergence or stagnation (our case).
3. Risk of stagnation: In some cases, the method may favor moderately fit individuals, leading to stagnation around suboptimal solutions.

Elitist selection

Elitist selection involves directly selecting the best individuals from a population to preserve them in the next generation. In this context, “best“ generally means those with the highest fitness.

Algorithme 4 Elitist selection

```
Require: $nb\_couple$
Ensure: List of indices of selected parents
1: $elite\_indices \leftarrow$ first $nb\_couple \times 2$ indices
2: return create_couples($elite\_indices$, $nb\_couple$)
```

Advantages:

1. Preservation of the best individuals: Ensures that the most efficient individuals are not lost, increasing the chance of maintaining or improving the quality of the population over generations.
2. Rapid convergence: Accelerates the algorithm's convergence process, as the advantageous traits of the best individuals are quickly propagated throughout the population.
3. Simplicity: Easy to understand and implement.

Disadvantages:

1. Risk of premature convergence: The population can quickly become homogeneous, reducing genetic diversity and increasing the risk of getting stuck in a local optimum.
2. Less exploration: By focusing on the best individuals, there may be less exploration of new potential solutions.
3. Dependence on fitness function: The method heavily depends on the quality of the fitness function to evaluate individuals. If this function is misleading or incomplete, elitist selection can lead to suboptimal results.

Probabilistic selection

Probabilistic selection, often called proportional selection, is a method where each individual in the population has a chance of being selected that is proportional to their fitness. This method is similar to roulette wheel selection, but with a slightly different approach in calculating probabilities. In probabilistic selection, individuals are chosen for reproduction based on their relative fitness compared to the entire population.

Algorithme 5 Probabilistic selection

```
Require: $arg\_list\_fitness, nb\_couple$
Ensure: List of indices of selected parents
1: $inverted\_fitness \leftarrow$ invert values of $arg\_list\_fitness$
2: $total\_inverted\_fitness \leftarrow$ sum of $inverted\_fitness$
3: $probabilities \leftarrow$ probability of each individual
4: $index\_population \leftarrow$ indices of the population
5: $index\_parents \leftarrow$ select $nb\_couple \times 2$ parents based on $probabilities$
6: return create_couples($index\_parents$, $nb\_couple$)
```

Advantages:

1. Favors high-performance individuals: Individuals with better fitness have a higher probability of being chosen, favoring the selection of high-quality individuals.
2. Genetic diversity: As all individuals have a chance of being selected, genetic diversity is better preserved compared to methods like elitist selection.
3. Dynamic adaptation: The method dynamically adapts to changes in the fitness distribution within the population.

Disadvantages:

1. Risk of slow convergence: If fitness differences between individuals are minimal, selection can become almost random, slowing convergence to an optimal solution (our case).
2. Domination by super individuals: In some cases, one or a few highly efficient individuals can dominate the selection, reducing genetic diversity.
3. Balancing probabilities: The method requires careful balancing of probabilities to avoid either too slow convergence or loss of genetic diversity.

3.2.6 Reproduction parents

For crossover, we can perform an n-point crossover, or mask-based as discussed in the subsection : 2.7.2.

As with the selection, we have implemented several crossover methods, which will allow us to better study these heuristics.

Single point and Two point crossover

This method involves randomly selecting one or more crossover points on the chromosomes of the parents. The chromosome segments before this point are swapped between the two parents to create a new chromosome. This means that the offspring inherits part of the chromosome from each parent, with the crossover point determining which part comes from which parent.

Algorithme 6 Single Point Crossover

```
Require: $chr1, chr2$
Ensure: New chromosome created by crossing over $chr1$ and $chr2$
1: $middle \leftarrow$ random integer between 0 and length of chr1
2: return $chr1[: middle] + chr2[middle :]$
```

Algorithme 7 Two Points Crossover

```
Require: $chr1, chr2$
Ensure: New chromosome created by crossing over $chr1$ and $chr2$
1: $middle1 \leftarrow$ random integer between 0 and length of chr1
2: $middle2 \leftarrow$ random integer between 0 and length of chr1
3: return $chr1[: middle1] + chr2[middle1 : middle2] + chr1[middle2 :]$
```

Advantages:

1. Simplicity and efficiency: It is a simple method to understand and implement, and it is computationally efficient.
2. Preservation of chromosome structure: By maintaining large blocks of chromosomes intact, it can preserve combinations of genes that work well together.
3. Genetic diversity: Encourages genetic diversity by creating unique combinations of genes.

Disadvantages:

1. Disruption of patterns: Can disrupt beneficial gene patterns if the crossover point splits genes that interact favorably (for example, if UVs superimpositions are created after a crossover).
2. Risk of slow convergence: May lead to slower convergence towards the optimal solution if the crossover point does not generate useful combinations.
3. Dependence on crossover point: The efficiency of the method heavily depends on the location of the crossover point, which is randomly chosen.

Uniform crossover

Unlike one-point crossover, where a single crossover point is used, uniform crossover involves deciding for each gene individually from which parent it will be inherited. This is usually done using a fixed probability (often 50%) to determine whether a specific gene is taken from the first or the second parent. Thus, the offspring is a kind of 'patchwork' of genes from both parents.

Algorithme 8 Uniform Crossover

```
Require: $chr1, chr2$
Ensure: New chromosome created by crossing over $chr1$ and $chr2$
1: $child\_chr \leftarrow []$
2: for $i \leftarrow 0$ to length of chr1 do
3:     if random integer between 0 and 1 is 0 then
4:         Append $chr1[i]$ to $child\_chr$
5:     else
6:         Append $chr2[i]$ to $child\_chr$
7:     end if
8: end for
9: return $child\_chr$
```

Advantages:

1. High genetic diversity: Allows for greater genetic diversity in the population, as it can combine genes in a more varied manner.
2. Less disruption of patterns: Reduces the risk of disrupting beneficial gene sequences, as it does not cut chromosomes into large blocks.
3. Efficient exploration of the search space: Encourages the exploration of new regions of the genetic search space, potentially leading to better solutions.

Disadvantages:

1. Risk of disrupting good combinations: Although it disrupts gene blocks less, it can still break up combinations of genes that work well together.
2. Potentially slower convergence: Due to the great variability it introduces, it can sometimes slow down convergence towards an optimal solution.

Ordered crossover

This method starts by selecting a subsequence from the first parent and copies this subsequence into the offspring. Then, it fills the rest of the offspring's chromosome with genes from the second parent, following the order in which they appear, while excluding those already present in the subsequence inherited from the first parent.

Algorithme 9 Ordered Crossover

```
Require: $chr1, chr2$
Ensure: New chromosome created by crossing over $chr1$ and $chr2$
1: $chromosome\_length \leftarrow lengthofchr1$
2: $start, end \leftarrow$ sorted random sample of two numbers from $range(chromosome\_length)$
3: $child\_chr \leftarrow [None, \ldots]$
4: Set $child\_chr[start : end + 1]$ to $chr1[start : end + 1]$
5: $chr2\_elements \leftarrow$ elements in $chr2$ not in $child\_chr[start : end + 1]$
6: Fill $child\_chr$ with elements from $chr2\_elements$ where $child\_chr$ is $None$
7: return $child\_chr$
```

Advantages:

1. Preservation of sequence order.
2. Diversity with structure: Allows for maintaining some structure while introducing genetic diversity. .

Disadvantages:

1. Risk of losing diversity: If the chosen subsequence is too large, it can limit the genetic diversity introduced by the crossover process.

3.2.7 Mutation

The main factor influencing a schedule is the time slot of the course. Therefore, to make an interesting mutation, we can swap the time slots of two different courses.

For mutation, we have implemented two solutions: one where we exchange the time slot between two courses, or we randomly change the time slot of a course.

For the time slot change:

Algorithme 10 Change Timeslot

```
Require: $chr$
Ensure: Chromosome with new timeslot
1: $gene \leftarrow$ select randomly gene in chromosome $chr$
2: $new\_day \leftarrow$ integer between 0 and 4
3: $new\_start\_time \leftarrow$ integer between 0 and x (to respect the end of the
   day at 8 p.m., so $gene.duration + gene.start\_time < 8p.m$)
4: Change $day$ and $start\_time$ of gene with $new\_day$ and $new\_start\_time$
5: return $chr$
```

For exchanging two time slots between two courses:

Algorithme 11 Swap Timeslot in Chromosome

```
Require: $chr$ (chromosome)
Ensure: Chromosome with mutated genes
1: $indices \leftarrow$ random sample of 2 indices from length of chr
2: if $gene\_1 with other cours finish before 8p.m.$
3:    and $gene\_2 with other cours finish before 8p.m.$ then
4:    $temp\_start\_time \leftarrow gene\_1.start\_time$
5:    $gene\_1.start\_time \leftarrow gene\_2.start\_time$
6:    $gene\_2.start\_time \leftarrow temp\_start\_time$
7:    $temp\_start\_day \leftarrow gene\_1.start\_day$
8:    $gene\_1.start\_day \leftarrow chr[indices[1]].start\_day$
9:    $gene\_2.start\_day \leftarrow temp\_start\_day$
10: else
11:    $chr \leftarrow change\_timeslot(chr)$
12: end if
13: return $chr$
```

3.2.8 Correction

Several correction cases can occur:

- A room occupied by two courses
- A room too small for the course capacity
- A professor required to teach two courses in the same time slot

- A missing course in the chromosome

We start by correcting the courses of a professor, without worrying about the rooms, and then we correct the rooms.

Teacher

If the professor has two courses at the same time, we change the time slot of one course.

Room

We start by modifying all the genes that occupy the same room at the same time by assigning an available room, without taking into account the capacity.

Algorithme 12 Room Not Ubiquity

```
Require: $chr$, $room\_name$, $planning\_room$, $plannings$
Ensure: Chromosome without room with more one course
 1: $indies$ ← get all timeslots where there are two courses, find $values > 1 in planning\_room$
 2: for $indice$ in $indies$ do
 3:     while $New room don't find$ do
 4:         $one\_planning$ ← take one planning from $plannings$
 5:         $free\_timeslot$ ← get free timeslot of $one\_planning$
 6:         if $indice \in free\_timeslot$ then
 7:             Replace the old room ($room\_name$) in the area concerned with the available room
 8:             Update schedules
 9:         end if
10:     end while
11: end for
12: return $chr$
```

At the end of this algorithm, we ensure that each room has at most one course.

Subsequently, we check if the capacity of the room is adequate. If not, we change the room for another available one with sufficient capacity.

Algorithme 13 Room Capacity

```
Require: chr (chromosome), room_capacity, courses_capacity, room_planning
Ensure: Chromosome with good room capacity
1: for gene in chr do
2:     if courses_capacity of gene $>$ room_capacity of room then
3:         room_number $\leftarrow$ choice a disponible room in room_planning with good capacity in room_capacity
4:     end if
5: end for
6: return chr
```

We now have, for each room, one course and sufficient capacity.

Courses

Given the creation of our chromosome, where we order the courses in the same way, it is impossible after crossover or mutation for a course to be removed from the chromosome.

3.2.9 Child acceptance

We accept an offspring if it improves the worst individual of our population. We remove the worst individual and add the offspring using the dichotomic insertion function (algorithm: 1). What's more, we only accept non-presenting children into the population, i.e. we don't want two identical individuals in the population.

3.2.10 Implementation

Here is the algorithm of our genetic algorithm. Our algorithm ensures that professors have one and only one course at a time, a room has at most one course, and the capacity is sufficient to accommodate all students.

Algorithme 14 *genetic_algorithm → timetable*

```
Require: list_promo, list_room, initial_population, generation, mutation_rate, parents_nb, correction
Ensure: the best timetable
 1: $list\_fitness \leftarrow fitness(initial\_population)$
 2: $sorted\_list\_fitness, sorted\_initial\_population \leftarrow sorting\_fitness(list\_fitness, initial\_population)$
 3: for _ in range 0 to $generation$ do
 4:   $index\_list\_parent \leftarrow selection(sorted\_list\_fitness, parents\_nb)$
 5:   $list\_children \leftarrow []$
 6:   for $i$ in range 0 to length of index_list_parent) do
 7:     $parent_1 \leftarrow sorted\_initial\_population[index\_list\_parent[0]]$
 8:     $parent_2 \leftarrow sorted\_initial\_population[index\_list\_parent[1]]$
 9:     $child \leftarrow crossover(parent_1, parent_2)$
10:     if $correction$ then
11:       $child \leftarrow correction(child)$
12:     end if
13:     $list\_children \leftarrow add(child)$
14:   end for
15:   for $j$ in range 0 to length of list_children do
16:     if $random\_in\_[0, 1] < mutation\_rate$ then
17:       $mutate\_child \leftarrow mutation(list\_children[j])$
18:       if $correction$ then
19:         $mutate\_child \leftarrow correction(mutate\_child)$
20:       end if
21:       $list\_children[j] \leftarrow correction(mutate\_child)$
22:     end if
23:   end for
24:   $list\_fitness\_children \leftarrow fitness(list\_children)$
25:   $sorted\_list\_fitness\_children, sorted\_list\_children \leftarrow sorting\_fitness(list\_fitness, initial\_population)$
26:   for $k$ in range 0 to length of sorted_list_fitness_children do
27:     if $sorted\_list\_fitness\_children[-k] < sorted\_list\_fitness[-k]$ then
28:       $sorted\_list\_fitness, index \leftarrow in\_place\_insertion($ $sorted\_list\_fitness\_children[-k]$, $sorted\_list\_fitness)$
29:       $sorted\_initial\_population[index] \leftarrow sorted\_list\_children[-k]$
30:     end if
31:   end for
32: end for
33: return $sorted\_initial\_population[0], sorted\_list\_fitness[0]$
```

3.2.11 Use

```
gen.genetic_algorithm(arg_promo, arg_room_list,
arg_population, arg_fitness, generation=100,
mutation_rate=0.01, nb_couple_elite=1,
selection_choice='elitiste',
crossover_choice='single_point',
arg_tournament_size=5, correction=True)
```

Here's an explanation of each argument:

- arg_promo : student list

```
[   {
        "student_id": "ebf54ee7-034f-4600-b400",
        "name": "Studentebf54ee7-034f-4600-b400",
        "uvs": [
        "AI53",
        "CC05",
        "HE09",
        "EV00",
        "VA50",
        "HE08"
        ]
    },...
]
```

- arg_room_list : the UTBM room list

```
{
        "E107": {
        "capacity": 117,
        "type": "AMPHI (vid\u00e9oprojecteur)",
        "description": "",
        "site": "Belfort"
    },...
}
```

- arg_population : a planning package

```
        [
        [
            {
            "room": "B421",
                "start_time": 540,
                "start_day": 3,
                "duration": 30,
                "teacher": "Franck GECHTER",
                "code": "IR5A",
                "type": "cm"
            },...
            ], ...
            ]
```

- arg_fitness : a table showing all the fitnesses of a chromosome
- generation : an integer defining the number of generations to be produced
- mutation_rate : the mutation rate $\in [0, 1]$
- nb_couple_elite : the number of couples per generation (i.e. the number of children)
- selection_choice : the type of selection in
  - tournament_selection : 'tournoi'
  - roulette_wheel_selection : 'roulette'
  - selection_probabiliste : 'proba'
  - selection_elitiste : 'elitiste'
- crossover_choice : the type of crossover in
  - single_point_crossover : 'single_point'
  - two_points_crossover : 'two_points'
  - uniform_crossover : 'uniform'
  - ordered_crossover : 'order'
- arg_tournament_size : the size of the tournament with $arg\_tournament\_size > nb\_couple\_elite \times 2$
- correction : a boolean, at *True* a correction is made at *False* the correction is ignored.

3.3 Schedules

To conclude the section on genetic algorithms, we developed a timetable visualizer for students, rooms, and teachers. Here is an example for the following parameters:

- Population size: 100
- Class size: 100 students
- Number of UVs: 20
- Number of generations: 10000
- Number of pairs: 2
- Crossover: single point
- Selection: roulette
- Correction: Yes

Here are the results:

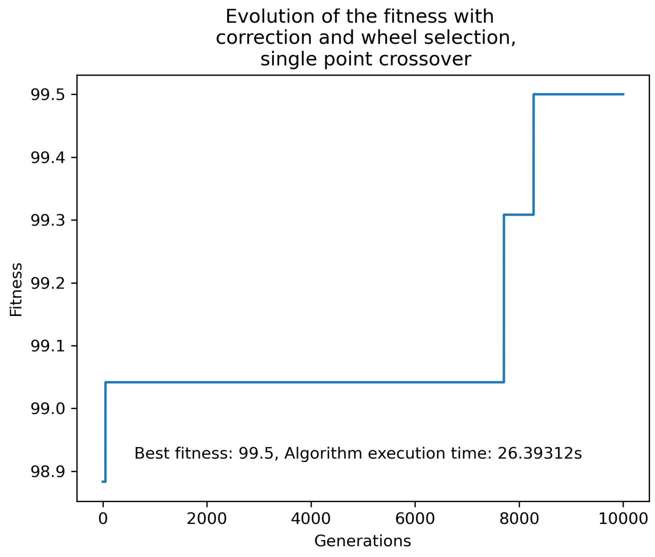

Image 4: Fitness

Here is the schedule of a room:

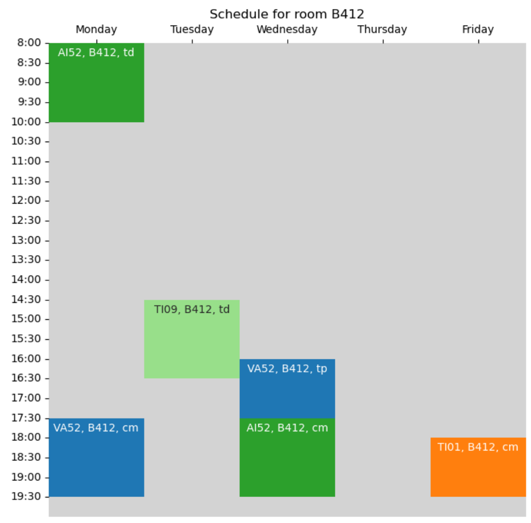

Image 5: Room schedule

Here is the schedule of a teacher:

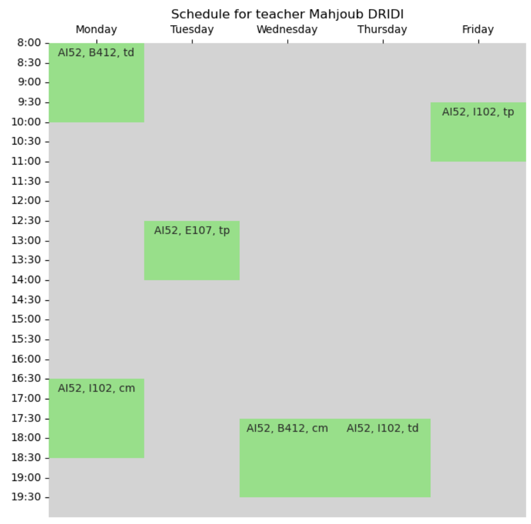

Image 6: Teacher schedule

Here is the schedule of a student:

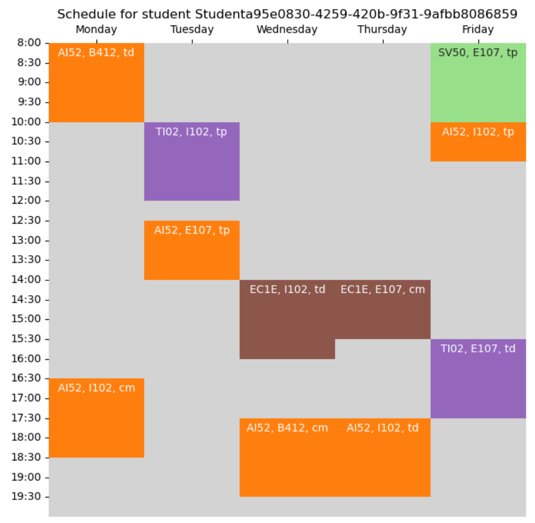

Image 7: Student schedule

To execute the code, you need to choose the following information:

```
NAME_TEST = "visuels_rapport" # Name of instance
PATH_DATA = "./datas/instances/" + NAME_TEST

GENERATION_NUMBER =
MUTATION_PROBABILITY =
NUMBER_OF_COUPLES =
CORRECTION = True # or False

"""
List of selection functions:
- tournament_selection : 'tournoi'
- roulette_wheel_selection : 'roulette'
- selection_probabiliste : 'proba'
- selection_elitiste : 'elitiste'

List of crossover functions:
- single_point_crossover : 'single_point'
- two_points_crossover : 'two_points'
- uniform_crossover : 'uniform'
- ordered_crossover : 'order'
"""

SELECT =
CROSS =
TOURNOI_SIZE = # Number of individuals in a tournament


TITLE_GRAPH = # Title
CREATE_SCHEDULE = # If you want to create all schedule

"""
Create a population and school year
SITE = ["Belfort", "Sevenans", ...]
FORMATION = ["FISE-INFO", "FISA-INFO", ...]
"""

SITE =
FORMATION =
POPULATION_SIZE =
PROMO_SIZE =
NUMBERS_UV =  # Number of UVs available for student
# selection
EXPORT_INSTANCES = # Save instance

room_list = []
for ville in SITE:
room_list = room_list + tb.get_rooms(ville)  # UTBM data
uvs_list = []
for formation in FORMATION:
uvs_list = uvs_list + tb.get_uvs(formation)  # UTBM data

# To create new instance uncomment

"""
pop, fit , promo, room_capa, uv_promo_capa =
gi.get_instance(SITE, FORMATION, POPULATION_SIZE,
PROMO_SIZE, room_list , uvs_list , EXPORT_PATH_INSTANCES,
EXPORT_INSTANCES)
"""
```

Execution :

```
python main.py
```

4 Experimental protocol

4.1 Context

The purpose of this experimental protocol is to detail the methodology of the experiment carried out as part of the optimization of UTBM's timetables. It is a guide describing the procedures and conditions necessary for carrying out a series of experiments designed to validate or invalidate our study hypotheses. The aim is to assess the effectiveness of the Genetic Algorithm by analyzing the impact of each parameter.

4.2 Research Questions

- How can Genetic Algorithms be effectively utilized to automate timetable creation at UTBM?
- What impact do different parent selection, crossover, and mutation methods have on the efficiency and quality of the generated timetables?
- How does our algorithm handle constraints such as room occupancy, student wishes, teacher schedules, and course capacities?

4.3 Objectives

- Develop a Genetic Algorithm for automated timetable creation.
- Evaluate and compare the performance of different parent selection, crossover, and mutation methods.
- Implement correction mechanisms to ensure compliance with constraints.
- Optimize the timetable creation process to minimize assignment conflicts and enhance resource utilization.

4.4 Methodology

4.4.1 Evaluation of the Algorithm

- **Performance Metrics:** Measure the quality of timetables generated according to defined criteria.

  - *Execution Time:* Measurement of the algorithms temporal efficiency.
  - *Solution Quality (Fitness):* Evaluation of the quality of generated solutions.

- **Compairing methods:** Compare the effectiveness of different selection, crossover, and mutation methods by combining them. Here is a list of possible combinations between the methods, without being limited to just that :
  - Elitist selection with single-point crossover
  - Elitist selection with two points crossover
  - Elitist selection with uniform crossover
  - Roulette selection with uniform crossover
  - Tournament selection with uniform crossover
  - Tournament selection with two points crossover

  Choosing the mutation methods will be done randomly for each offspring by selecting one of the two methods, either Change Timeslot or Swap Timeslot.

- **Optimizing parameters:** Identify the optimal parameters of the algorithm to improve its performance. We've identified the following :
  - Number of generation
  - Population size
  - Mutation rate
  - The number of couples per generation
  - The size of a tournament

4.4.2 Experimental design

The most influential parameters will be selected for the creation of a test grid, comparing execution time and solution quality.

- **Test Grid Creation**

  Identification of Most Influential Parameters Selection of the most influential parameters based on their impact on performance metrics.

Test Grid Construction of a grid for each influential parameter, covering a range of values to comprehensively explore the parameter space.

As an example, consider the test grid for the "Mutation Rate" parameter. To carry out these tests, we took a population of size 10, 100 students, and 20 UVs (as the students take 6 UVs, this makes about 30 students per UV). The number of generations will be 1000, the selection by a roulette wheel, the crossover a single point, and the number of pairs per generation 1. The averages are calculated over 10 repetitions. The best fitness of the population is 98.25.

Mutation Rate	**Times in s (Avg.)**	**Fitness (Avg.)**
0	0.75	99.27
0.1	1.04	98.83
0.3	1.46	98.55
0.5	1.6	98.7
0.7	1.85	98.62
0.9	2.09	98.67
1	2.29s	98.60

With what precedes, we are going to take a mutation rate of 5%. The lower it is the better, but there is a risk of falling into a local minimum.

Or the test grid for combining Selection and Crossover methods :

Selection method	**Crossover method**	**Time in s (Avg.)**	**Fitness (Avg.)**
Elitist	One point	1.12	98.37
Elitist	Uniform	1.16	98.84
Roulette	Uniform	1.22	98.87
Tournament	Two points	3.1	98.3

4.5 Results discussion

4.5.1 Selection and Crossover methods

In order to find the most effective Selection and Crossover method, we have carried out numerous tests, combining the different methods that have been implemented.

These tests were obtained with a minimal number of couples, i.e. 2. This parameter is further adjusted in the second part of the tests. For reasons of convenience, we have kept the same parameters, namely:

- Initial population: 70 because $2 \times 30 = 60 < 70$
- Number of UVs: 20
- Number of students: 100
- Number of generations: 1000
- Mutation rate: 0.5
- Selection: roulette
- Crossover: single point

Initial fitness is : 98.59.

4.5.2 Optimizing parameters

- The number of couples

Nb couples	**Times in s (Avg.)**	**Fitness (Avg.)**
1	1.33	98.95
5	8.43	99.11
10	17.02	99.17
15	25.57	99.27
30	53.29	99.54

- The size of a tournament

Tournament size	**Times in s (Avg.)**	**Fitness (Avg.)**
4	1.36	98.85
20	1.58	98.87
30	1.56	98.86
60	1.61	98.8

- Number of generation

Nb generation	**Times in s (Avg.)**	**Fitness (Avg.)**
100	0.09	98.83
500	0.43	98.84
1000	0.94	99.1
10000	11.55	99.47

- Population size

Population size	**Time in s (Avg.)**	**Fitness (Avg.)**
10	0.96	98.83
100	0.93	99.04
500	0.86	99.11

4.6 Results discussion

The experiment to automate timetable creation at UTBM using Genetic Algorithms has been successfully conducted, and the results provide valuable insights into the efficiency of the proposed methodology.

The primary outcome of the experiment is the generation of optimized timetables. The algorithm demonstrated its capability in optimizing timetable creation, achieving a significant reduction in assignment conflicts. The algorithm successfully addressed constraints related to room occupancy, teacher schedules, course capacities, and student choices.

The initial population generation, despite being random, efficiently produced a diverse set of individuals. The sorting mechanism significantly improved the algorithm's performance, enabling the quick identification of the best schedules. The comparison between various Selection and Crossover methods revealed distinct advantages and disadvantages, aiding in choosing the most effective combinations. Roulette wheel selection exhibited robust performance with an important number of couples, balancing selective pressure and genetic diversity. The mutation strategies, involving the swapping of time slots between courses or random time slot changes, effectively introduced variability into the population. Tests showed that a higher mutation rate contributes to the algorithm's ability to escape local optima and improve the overall quality of timetables.

Additionally, certain trade-offs were observed. The algorithm's sensitivity to parameter settings, such as population size, number of couples, and mutation rates, emphasizes the importance of fine-tuning these parameters for optimal results.

In conclusion, the results of this experimental protocol demonstrate the feasibility and effectiveness of using Genetic Algorithms for automating timetable creation at UTBM.

Conclusion

The culmination of our efforts in automating the timetable creation process for UTBM has yielded promising results. Through the implementation of a Genetic Algorithm, our solution has demonstrated the ability to efficiently generate optimized timetables while respecting the complex constraints described in the functional scope. The testing and evaluation phase revealed that our algorithm exceeded expectations in managing hard constraints.

The utilization of a Genetic Algorithm enabled us to explore vast solution spaces, adapt to the constraints, and iteratively refine schedules to meet the minimum criteria we initially set. The emphasis on meeting minimum criteria ensures practical usability and aligning timetables with real-world scheduling constraints. It resulted in timetables that respect students choices, classroom availability, course capacities, and various other critical factors.

Looking ahead, the project can explore advanced optimization methods, refine correction mechanisms, fine-tune algorithm parameters, include additional constraints, and collaborate with educational institutions to adapt the algorithm to specific needs.

In summary, this project marks an important step in the pursuit of efficient and automated timetable management at UTBM. The final objective of this project has been achieved successfully by generating optimized timetables and we managed to build a robust solution that can be incrementally enhanced, offering the potential to save valuable human resources and improve overall timetable management.

Glossary

UTBM University of Technologie of Belfort-Montbéliard. 2, 6, 22, 36, 42, 46–48

List of Figures

List of Algorithmes

References

[1] Mohammad Reza Feizi Derakhshi, Hamed Babaei, and Javad Heidarzadeh. A survey of approaches for university course timetabling problem. 09 2012.

[2] Manar Hosny. A survey of genetic algorithms for the university timetabling problem. *International Proceedings of Computer Science and Information Technology*, 13, 01 2011.

[3] Ahmed Mahlous and Houssam Mahlous. Student timetabling genetic algorithm accounting for student preferences. *PeerJ Computer Science*, 9:e1200, 02 2023.

[4] Alade O Modupe, Omidiora E Olusayo, and Olabiyisi S Olatunde. Development of a university lecture timetable using modified genetic algorithms approach. *International Journal*, 4(9):163–168, 2014.

[5] Omar Ibrahim Obaid, M Ahmad, Salama A Mostafa, and Mazin Abed Mohammed. Comparing performance of genetic algorithm with varying crossover in solving examination timetabling problem. *J. Emerg. Trends Comput. Inf. Sci*, 3(10):1427–1434, 2012.

[6] Olivia Rossi-doria, Michael Sampels, Mauro Birattari, Marco Chiar, Marco Dorigo, Luca Maria Gambardella, Joshua Knowles, Max Manfrin, Monaldo Mastrolilli, Ben Paechter, Luis Paquete, and Thomas Stützle. A comparison of the performance of different metaheuristics on the timetabling problem. 03 2003.