UNIVERSITY OF OVIEDO

School of Computer Engineering

FINAL DEGREE PROJECT

“EXAM TIMETABLING FOR THE SCHOOL OF COMPUTER ENGINEERING USING ARTIFICIAL INTELLIGENCE TECHNIQUES”

Author

Luis Presa Collada

Tutors

Carlos Mencía Cascallana

Raúl Mencía Cascallana

Acknowledgements

I am grateful to my tutors Carlos and Raúl, and also to my client Fernando. Not only I appreciate your involvement in the project but also your patience as personal circumstances forced me to postpone it.

Many thanks to all the colleagues that I have made this last years. Its because you all and the experiences that we share together that I have been able to grow as a person as much as I did. Special mention to Adrian, Diego and Pablo which were by my side since the very beginning.

Dear friends, you know how much I have been through while doing this project. Your support, your encouragement, I cannot thank you enough for those precious things.

Finally, thank you to my parents for guiding me since my childhood. Your education is what allowed me to make my way here.

Abstract

The aim of this research project is to analyze the real problem that the University of Oviedo Computer Engineering School has with the generation of exam schedules and whether or not a solution can be found. The actual way to solve the problem is studied and an analysis of the state of the art is carried out in order to see how similar problems are solved in the literature.

As the core of the project, the problem is formalized and a solution is proposed by means of Genetic Algorithms and Greedy Algorithms. A Greedy Algorithm is proposed to find good local solutions, and it is complemented with a Genetic Algorithm which is in charge of leading the search towards better areas of the search space.

A prototype application is developed to test the acceptability of the proposed solution. To do so, a set of experiments are carried out considering simulated real scenarios. An additional tool is implemented to generate similar problem instances to the real ones provided by the school.

Such instances are used both to tune the algorithm parameters as well as to showcase its effectiveness. In the end, the results show that the proposed algorithm is efficient at solving the problem, and it simplifies considerably the complexity of the actual process.

Keywords

Meta-heuristics, Scheduling Problems, University Exam Timetabling, Genetic Algorithms, Greedy Algorithms.

Contents

List of Figures

Part I

Project Memory

Chapter 1

Project Memory

1.1 Project Motivation

This project springs from the need to find a solution to a real problem that the University of Oviedo School of Computer Science has. Every year many resources, specially time, are invested in manually making the exam timetable for the three annual university calls.

If the problem was formalized and a solution was proposed, such solution could be implemented into an application and so solve in a automated way the problem of the School of Computer Science. This would result in the saving of many resources invested annually in manually solving this problem, specially time.

1.2 Personal Motivation

When I first started my studies at the School of Computer Science I was expecting to work on application development in a business oriented environment, because I did not really know enough about other alternatives. However, as years passed by I discovered other topics that interested me much more such as automating tasks, monitoring health values, etc. Thus, by doing research on my own I found out about neural networks, machine learning, deep learning, and some other technologies.

The Degree in Software Engineering is more oriented to a business software development career, which is why I initially went into it, so that is why none of the commented technologies was taught in the first three years. On the contrary, on the fourth year, there was a subject which was centered on this field. I honestly like it, and I became fascinated by optimization problems.

Also on my fourth year I obtained a grant on artificial intelligence which allowed me to experience the academic environment from a researcher point of view. I learnt how research is done, how to read and write papers, and many other things. I was not honestly expecting to like it as much a I did.

This Final Degree Project not only allows me to put into practice what I have learned on the grant, but also it is also oriented to optimization which gives me a reason to do research on this particular type of problems. Furthermore, the final prototype app that is going to be developed is not just a simple delivery but something which is expected to be used in reality in the school.

1.3 Objectives

This project is going to study and analyze the real life problem of making the exam calendar at the University of Oviedo School of Computer Science. For doing so, it is necessary to formalize the problem and identify its particular constraints before starting to think about an actual solution. This will be done with the help of the director and the deputy director, both in charge of performing this task manually every year.

Once the problem is fully understood and formalized, a method to solve it must be designed. In this particular case this method will be based on evolutionary algorithms, concretely Genetic Algorithms in combination with Greedy Algorithms. This design decision was chosen among some alternatives as commented in Section 2.3.

In the end, the goal is to implement a prototype application to provide a solution for the problem so that the actual users can see the complexity of their task reduced. It is desired for such prototype to be easy to extend and modify since circumstances can change easily, as has been seen during the Covid-19 pandemic.

1.4 Scope

As stated above, the purpose of the project is to study the problem and to develop a prototype application that provides a solution to the problem. Even though this is a research project, and so the prototype application was initially thought of as a visual key to showcase the proposed solution, it was adapted to be utilized by the user. This mainly implies the inclusion of some particular user constraints (see Section 4.3.3) as well as some format requests about the input files.

Although the prototype is an important piece of the project because it must be easy to extend and adapt to the particularities of the school environment, it is not the only contribution of the project.

1.5 Project Summary

The aim of this project is to look for a solution that solves the School of Computer Science Exam Scheduling problem. For doing so, the problem will be analyzed and formalized with the help of the people currently in charge of manually solving it, being such the director and the deputy director of the school.

Once the problem is understood, a literature review will be carried out in order to find how similar problems have been solved recently. This review will also be considered while explaining the theoretical concepts associated to the project, such as are Metaheuristics or timetabling problems.

Afterwards, a solution for the problem will be proposed according to the reviewed literature. One of the alternatives will be selected which will allow us to consider all the distinct features of our problem in order to obtain high quality solutions. Whenever the solution is designed, a prototype implementing it will be built to test the efficiency of the proposed solution.

Finally, to test the suitability of the prototype a series of experiments will be performed. Some real instances of the problem were provided by the school, but since there are many new functionalities and constraints with respect to what has been done before there was no real precedent. Because of that the real instances were studied, and another simple application was designed to create instances similar to the real ones, which were the ones finally used in the experiments.

To run such experiments a cluster will be used in order to run in parallel many of them. The results will be presented and discussed and a final conclusion will be reached.

1.6 Project Structure

In this section the structure of the document will be described.

1. **Project memory.** Exposes a general view of the project.
2. **Introduction.** Describes in detail the motivation of the project and which are its aims.
3. **Theoretical aspects.** An analysis of the state of the art is done, and theoretical explanations are provided to understand both the problem and the proposed solution.
4. **Problem definition.** The problem is described and formalized.
5. **Proposed solution.** The proposed solution to the problem is described.
6. **Project scheduling and budget.** The project time schedule and the client budget are provided.
7. **Analysis.** The requirements table is presented. Subsystems of the prototype application are identified along with system actors and use cases.
8. **System design.** The design of the prototype application is explained by means of UML diagrams. A detailed explanation of taken design decisions such as design patterns application is also provided.
9. **System implementation.** The technologies used in the project are described.
10. **System tests.** The design of the tests is presented, as well as the combination techniques applied to generate the test cases.
11. **Experimental study.** The algorithm parameters and the fitness function are tuned. Furthermore the performance of the proposed algorithm is tested on different scenarios.
12. **User manual.** The input and output files are detailed as well as the procedure to run the algorithm.
13. **Budget.** The project budget is detailed, and the client budget creation is described.
14. **Conclusions and future work.** The conclusions of the project are described, and some extensions are proposed.

Chapter 2

Introduction

2.1 Project Justification

Three times a year, one per call, the University of Oviedo School of Computer Science must publish the exam calendar for the 43 subjects available at the Degree in Software Engineering. Each subject coordinator can state one or more events that must be present in the schedule, such as are theoretical or practical exams, or even some deadlines for deliveries.

Subject coordinators may make certain requests, or even state some constraints about when their exams must take place. For instance, it may be interesting to state a time distance of X days between two exams, which would prevent a theoretical and a practical exam of the same subject of being on consecutive days. Another common example is that a professor is out for some reason, such as attending a conference. That implies that there will be some days in which he will not be able to be at the exam, and so the exam cannot be placed when he is out.

Moreover there are other desired conditions to be considered: as far as possible there must not be two exams of the same year scheduled on the same day, there must be some free time for lunch,...

A random assignment, which could be seen as the easy and straightforward approach, is not viable because of those constraints. This is why this task has been performed in a fully manual approach by the deputy director for many years. Even though it may seem not too complex, as the number of constraints increases so does the dependencies among the exams, which causes that even the slightest change can have a significant impact on the constraints fulfillment.

Not having the certainty of whether or not the constraints are fulfilled, and forcing the user to check them manually one by one is a tedious experience. However, if every condition could be understood by a machine that could compute good enough solutions satisfying the provided constraints, user work would be just to pick among one of the solutions provided and adjusting some exam hours, which is a noticeable simplification with respect to the previous task.

This project is aimed to find whether a solution for this real life problem can be found and if so, to provide an automatic tool prototype that solves the problem. Recently a similar problem was solved using AI algorithms (4) which suggests that this project may be successful.

2.2 Project Goals

As mentioned before, the main aim of this project is to study if a solution for the University of Oviedo School of Computer Science exam calendar problem can be found. To this aim, research will be carried out to observe how similar problems were solved in the past.

To analyze the problem, it must be formalized first. Thus, we will need to establish the search space, the state representation, and how to compare the possible solutions. This will be explained in more detail later. To obtain all the necessary information collaboration with the school director and deputy director will be required.

Once the problem is formalized, we will design an algorithm to solve it, and implement a prototype application implementing such algorithm. For this prototype definition process, the current files used at the University of Oviedo School of Computed Science will be considered, as well as the actual exam calendar format. The main aim of this prototype is to provide information about the quality of the solutions computed, not only to check if a solution can be found, but also to improve the performance of the algorithm in case it is possible.

The algorithm will be tested using real data from previous exam calls as well as some randomly generated instances similar to the real instances. This combination will allow us to see its behavior in real life scenarios. Results will be evaluated manually in order to see the performance of the algorithm.

At the end, the prototype and the research results will be provided to the University of Oviedo School of Computer Science for its use or further extension.

2.3 Analysis of the Present Situation

This problem has been solved manually for many years in the school. As briefly commented before, solving it is a tedious task prone to errors, which takes a lot of time and effort. Not only the user must schedule all the exams but also do it according to the university rules and fulfilling as much constraints as possible.

Furthermore, due to the Covid-19 pandemic, the current user was forced to consider some extra circumstances, such as the need to establish a cleaning time for the rooms after exams have finished. This increases even more the complexity of the task to be performed, because that extra time can differ from exam to exam depending on the classrooms assigned to it.

This Final Degree Project came up due to a request from the school. It was specially interesting for them the creation of an automatic tool for solving the problem, since that would allow the consideration of new constraints whose satisfaction would be unfeasible in a manual way.

Chapter 3

Theoretical Aspects

3.1 Introduction

This section is devoted to provide a theoretical context to understand the problem and the possible solutions to solve it. To do so a study of the state of the art is done, by which we have classified the problem as a Timetabling problem (Section 3.2), and enumerated some common approaches to solve them.

Because this project solution uses Genetic Algorithms, the Metaheuristics are generally explained (Section 3.3), and emphasis is done in Population-Based Metaheuristics (Section 3.4). Common operator implementations are enumerated and an example algorithm is presented.

3.2 Timetabling Problems

Scheduling problems consist in the creation of a plan for a given number of temporal events or tasks. There are many problems that fit in this category such as the optimization of the total time spent by a group of machines to perform a series of task, or the creation of a timetable for a high school.

The problem addressed in this project is similar to the second one mentioned above, and they are both known as *timetabling* problems, which are a subset of the scheduling problems. In this particular type of problems the aim is to build a timetable according to series of domain-dependent constraints. It would not be the same to create the weekly timetable of a high school than to create a exam calendar at the university. There are more concrete classifications among which we could see our problem as an university exam timetabling one.

Many of these timetabling problems have been proven to be NP-hard, which implies that there is no known approach that can solve them in polynomial type using a deterministic algorithm. Metaheuristics have been widely used approach to solve them, and since they will in fact be used in this project, they will be explained in Section 3.3.

Even though the proven complexity, many techniques have been proven effective with this type of problems along the years (16), such as Integer Programming, SAT, Constraint Programming, Metaheuristic-based approaches, Graph Colouring, among others.

SAT is a well-know approach to solve timetabling problems (2). It focuses essentially on if it is possible to obtain an interpretation that satisfies a provided Boolean formula. It was the first problem to be proven NP-Complete, so there is not known algorithm that solves SAT in polynomial time. There are two ways to solve SAT: using a exact method such as CDCL (Conflict Driven Clause Learning) or by means of local search. From SAT can be derived MaxSAT, an optimization version of SAT which focused on computing the maximum number of clauses that can be made true with an assignment of truth values to the formula variables. Nowadays there are solvers that work efficiently considering instances of the problems composed of tones of propositional variables and clauses. Provided that a timetabling problem can be represented as a Boolean formula, using this type of algorithms could be a solution to the problem (1).

CP (Constraint Programming) (13) is also a common approach for solving scheduling problems. Users must describe which properties must the solution have, that is, they must state in a declarative way the problem constraints. As a declarative approach, a solver will try to achieve that solution by means methods such as constraint propagation and search.

Since our problem is mainly aimed at satisfying a given set of constraints it can be seen as a CSP (Constraint Satisfaction Problem) (12). Provided that the problem can be represented as an homogeneous collection of finite constraints over variables (7), the problem can be solved with constraint satisfaction mechanisms. In fact, SAT can be seen as a particular case of CSP.

Another well-known alternative is IP (Integer Programming), where constraints are specified over a set of integer variables. It has also been proven effective at solving timetabling problems (6), although it can be used in other scheduling domains such as for the knapsack problem.

Our problem is a COP (Constraint Optimization Problem) which is a generalization of CSP but including an objective function over some variables to optimize. Although all the mentioned alternatives are potential approaches to solve the problem of the School of Computer Science, in the end they could lead to unfeasible execution times.

Metaheuristics are the common alternative to obtain good enough solution in reasonable times. Some viable approaches are: Genetic Algorithms, Particle Swarm Optimization and Local Search. In (16) they are compared along with other approaches.

Recently, other Final Degree Project solved an optimization problem for the school using Genetic Algorithms (GA) (4). Even though the problem was different to ours, because it consisted in assigning students needed to class groups, GA worked really well considering an encoding based on permutations.

For such a reason, Genetic Algorithms will be used in this project. They are studied at the Degree in Software Engineering, which means that I am familiar with them, as is expected of any fourth course student.

Genetic Algorithms can provide good enough solutions for the problem and can be easily configured by any user, even if non-expert. By setting the available parameters, the deputy director will be able to quickly obtain a set of solutions to choose from.

Because of that, coupled with the author's lack of familiarity with the other exposed alternative, GA were in the end the chosen alternative.

3.3 Metaheuristics

Optimization problems such as the one studied in this project can be found in plenty of real life contexts, from the typical ones such as the assignment of tasks to machines to minimize time or resources, to an Internet routing architecture. In the end, they all try to achieve some objectives like the minimization of the time to complete a group of tasks, or the minimum energy cost of a routing system.

Since there may be multiple objectives, the optimization tends to be highly non-linear. Because of that, a good enough approximation of such a solution can be acceptable, if computed at a reasonable cost.

Metaheuristic algorithms are designed to achieve acceptable solutions in a reasonable amount of time. These methods integrate several mechanisms to explore the solution space and intensify the search in its most promising regions. Usually there is no way for these algorithms to state whether a solution is optimal or not, neither to bound the error of the provided solutions.

Most metaheuristics are focused on solving combinatorial optimization problems which consist in finding an optimal solution among a discrete set of possible solutions. A well-known example of this kind of problems is the TSP (Travelling Salesman Problem) as well as many scheduling and timetabling problems.

Metaheuristics can be classified according to different criteria. It is interesting to mention single-solution [1] vs. population-based algorithms. The first kind of methods focus on iteratively improving the quality of a single solution to the problem (e.g. local search methods), whereas the second class of algorithms maintain a population of candidate solutions that are exploited in the search for optimal solutions (e.g. evolutionary algorithms, particle swarm optimization (11), etc.). For further information, we refer to (15).

3.3.1 Meta-heuristic design

To design a meta-heuristic two main elements are needed: the representation of the solution, often called encoding, and an objective function that will guide the search, usually called fitness function.

Representation of the solutions

It is crucial to take into account how the solutions are going to be represented, as this will be strongly related to the efficiency of the algorithm. A good encoding must be:

- **Complete:** Every possible solution of the problem can be represented.
- **Connected:** Every solution of the search space can be achieved, and all of them are linked by a search path.
- **Efficient:** The handling of the encoding by the search operators must not have a high time or spatial complexity.

Fitness Function

This function has the responsibility of guiding the search by assigning a quality value to each of the solutions, which will typically be a real value. Therefore we can define the fitness function as: $f : S \to \mathbb{R}$, being f the fitness function, S the set of possible solutions and $\mathbb{R}$ the set of real numbers.

[1] Even if not used in this project they have been used in the timetabling field as can be seen in (14) and (10).

Since the fitness function states how good a solution is, it is a determinant factor on how the algorithm will perform. Different fitness functions can totally change the output of solutions in the same search space, thus it is important to define it right so that the provided solutions match our desired criteria.

3.4 Population-Based Meta-heuristics

As briefly explained before, population-based meta-heuristics work simultaneously with a group of candidate solutions, often called "population". The idea behind this approach is to start from an initial population that may have been generated randomly, or by means of a heuristic, in case that we want to generate a tendency towards a particular area in the search space.

When generating the initial population it is important for it to provide a wide range of different solutions to avoid premature convergence of the algorithm. Once that the initial set of encoded candidate solutions is provided, the algorithms iterate over two phases until meeting a certain criteria such as could be a maximum number of generations or surpassing a certain threshold value for the fitness value. The two phases are:

1. **New generation production:** a new generation is produced considering the actual generation. The behaviour of the algorithm in this phase will be the most differentiating factor among the population-based meta-heuristic algorithms.
2. **Replacement process:** from the previous generation population, and the the one generated in the previous step, a new population must be built. There must be a replacement policy to guide this process, a possible approach could be that all the individuals generated in the previous step replace completely the old generation. Over that approach some slight changes can be implemented such as the concept of elitism, where the best fitness individual from the old generations goes directly to the final population.

In Algorithm 1 we can see a generic pseudocode of these algorithms. In the example, the number of generations (N_{gen}) is the stopping criteria.

Algorithm 1 Generic Population-based Meta-heuristic algorithm.

```
Input: $P_0, N_{gen}$ /* Initial population and number of generations that must be done. */
  $i = 0$;
  for $i \leftarrow 0, ..., N_{gen}$ do
    $P'_i = newGeneration(P_i)$;
    $P_{i+1} = Replacement(P_i \cup P'_i)$;
  end for
Output: Best solution(s) found.
```

3.4.1 Evolutionary algorithms

Evolutionary algorithms are well-known population-based metaheuristic optimization algorithms. The idea behind this family of algorithms came by observing the biological evolution of living beings.

When talking about evolution, it is necessary to refer to Darwin's book: *The Origin of The Species*. Darwin observed that there was an observable principle in nature, which he called

”Natural Selection”. It stated that the most adaptive individuals persist and reproduce, while the non-adaptive ones are eventually discarded by the evolution process.

In addition, Darwin also mentioned that even if normally the evolution tends to the most adaptive individuals, some random mutations can also have a considerable impact on the evolution process.

Evolutionary algorithms are inspired in what Darwin exposed. They consider a random population of individuals, which are candidate encoded solutions, and then try to simulate this evolution process.

To this aim, a fitness function is defined, to measure how good an individual is. At this point the algorithm will work as shown in Algorithm 1. The main difference is that here the process of creating a new generation is a simulation of the biological evolution. Firstly, some individuals will be selected, usually the higher the fitness the higher the probability to be chosen. Secondly, the selected individuals will be used to create a new population, crossing them to generate new ones and even performing some random mutations over the result.

Once that new generation is obtained, the replacement phase comes into place and sets the final population for the new generation. The whole process is detailed in Algorithm 2. Note that again we are considering as stopping criteria for the example the number of generations, it could be to reach a certain fitness or any other.

Algorithm 2 Generic Evolutionary algorithm.

```
Input: $N_{gen}$ /* Generations that must be done. */
  $P_0 = generateInitialPopulation()$;
  $i = 0$;
  for $i \leftarrow 0, ..., N_{gen}$ do
    $P'_i = SelectionProcess(P_i)$;
    $P'_i = CrossoverProcess(P'_i)$;
    $P'_i = PerformRandomMutations(P'_i)$;
    $P_{i+1} = Replacement(P_i \cup P'_i)$;
  end for
Output: Best individual(s) or best population found.
```

Convergence

Along the generations, the population individuals will gradually increase their fitness. At some point, this growth will be stopped or highly reduced, even if continuing computing new generations. That happens because all the individuals tend to have the best fitness, and that means that the more those individuals are mixed along the generations the more similar those individuals will be, until reaching a point in which all of them are identical. This is called *Convergence.*

In the end *Convergence* is just the lack of diversity in the population, which is not by itself something bad. However, an algorithm converging too fast is not something desirable, because that usually implies that the provided solution is not as high of quality as it could be. This is called *Premature Convergence.*

It is needed to introduce now two opposite concepts strongly related with *Convergence.*

- **Exploitation.** Once the algorithm have a reasonably good solution it tries to refine it to a local optimum. Doing this implies that the algorithm will converge to that solution.

Therefore, given two algorithms A and B, if A has a stronger exploitation policy it is more likely to have premature convergence.

- **Exploration.** The algorithm does not focus on a local solution, but instead it tries to search in the whole search space to find the best local optimum among the ones found. In the best scenario, that local optimum is the global optimum.

Focusing too much in exploitation will lead to premature convergence, whereas doing it on exploration will lead to an algorithm that does not converge to a solution. It is needed to keep a balance between these two concepts.

Solutions encoding

The fitness function works over a candidate solution, whereas the evolutionary operators work over an encoding of such solution. This means that this type of algorithms need to perform a "translation" over the population of candidate solutions to perform the evolution process. On the contrary, when it is needed to check by means of the objective function how good the solutions are, those encoded solutions must be decoded.

There are many ways of encoding solutions, we can represent them as: a binary number, a permutation of values, and many others. For instance, if we consider the TSP (Travel Salesman Problem), in which it is needed to find a path thought a number of cities, those paths can be represented as permutations of an ordered list containing all the cities.

3.4.2 Evolutionary operators

As explained before, and as can be seen in Algorithm 2, there are three processes involved in the creation of a new generation of individuals, which are called operators. There are many ways in which each of these operators can be defined and this mostly depends on how the individuals were encoded. However, there are some well-known options that we describe in this section.

Selection

The selection operator has the responsibility of choosing which individuals will be considered for crossover. Typically its definition involves choosing the fittest individuals. This behaviour is called *selective pressure*, and although it intuitively leads to better solutions, also implies that lower fit individuals tend to be discarded. This could be a problem since those "bad" individuals could contain useful information which will be lost instead of mixed with the other individuals. In order words the bigger the *selective pressure*, the more we are focusing on exploitation.

In the end the selection operators must reach a balance between exploitation, prioritizing the fittest individuals, and exploration, considering also individuals with not too high fitness since they can have some useful genetic data.

Among the approaches for the selection operators the following are well-known and widely used:

- **Random selection:** the individuals are randomly selected without following any criteria.
- **Roulette Wheel selection:** this strategy assigns a probability to be selected to all individuals based on their fitness. The selective pressure will depend on how that mapping is done.

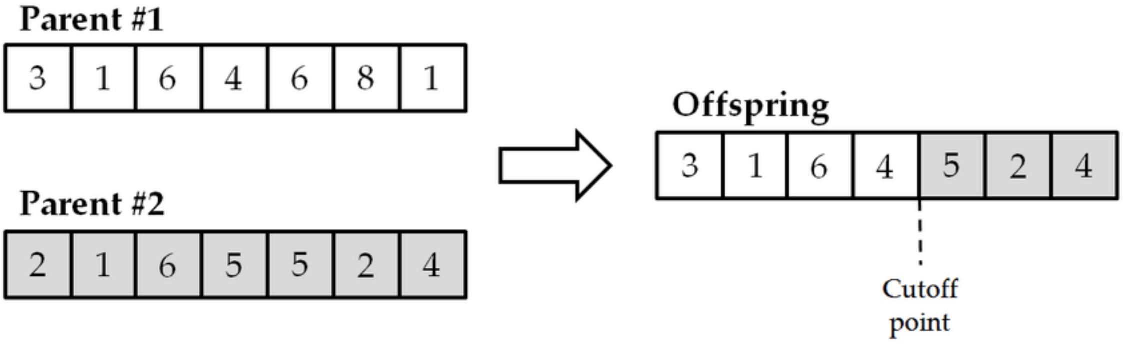


Figure 3.1: Example of 1-point crossover.

- **Tournament selection:** this strategy is a combination of the previous ones. It consists in selecting randomly a given number individuals among which the one with the highest fitness will be the finally selected one. This procedure must be repeated each time a new individual must be selected.

Mutation

The mutation operator performs minor changes over a single individual. This operator is usually yielded according to a mutation probability, given as a parameter. These mutations are performed over new created individuals, typically after the crossover operator has been executed. By performing a mutation, we potentially introduce new genetic data to the population, which favors the exploration of the algorithm.

Mutation probability, let us call it p_m, tends to have a low value, because having a higher one may disrupt the process. In the end this operator is a way to prioritize exploration over exploitation. For such a reason, typically $p_m \epsilon [0.001, 0.15]$. However, depending on the problem and the proposed solution that values can be increased.

Crossover

The crossover operator, also called recombination operator is in charge of mixing the encoded data of the selected individuals to generate their children. It is more often defined as a binary operator, although it is possible for it to be n-ary.

As considered for the mutation operator, the crossover operator is also issued according to a probability, let us call it p_c. This probability tends to be high because in the very end we are interested in the individuals to cross to generate new ones. Because of that $p_c \epsilon [0.5, 1]$.

There are many crossover operators among which 1-point crossover was the first one. This implementation selects randomly a point in the chromosome sequence and builds the child as an addition of the mixture of the parents. It is represented in Figure 3.1. Extending this idea there is also the k-point crossover, which basically chooses k division points and acts in the same way as 1-point crossover. Finally the uniform crossover chooses randomly from each offset of the parents.

Other well-known crossover operator is Order Crossover (OX) (3). Its behaviour can be seen in Figure 3.2. This crossover operator is specialized in permutation-based encondings. Two points are selected and the sequence of offsets between them is directly written in same order and position to the child from the first parent. The other values will be written in the child's

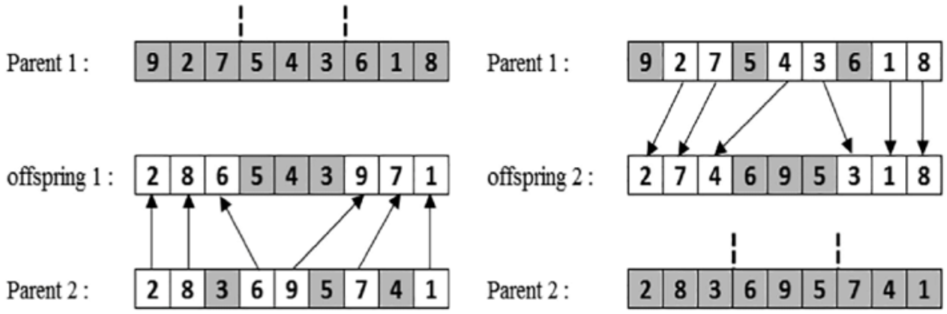


Figure 3.2: Example of OX crossover.

free spaces following the appearance order in the second parent.

3.4.3 Genetic Algorithms

Genetic Algorithms are a typical instantiation for Algorithm 2. They were developed during the 1960s and the 1970s by professor John Holland (8) and his students following the idea of "breeding" encoded solutions to optimizations problems.

Typically a Genetic Algorithm applies a crossover operator over two previously selected solutions by a selection operator. The result of the crossover operator could or not be randomly modified by a mutation operator depending on the mutation probability. Then, considering "parents" and "children", the replacement operator selects the ones that will constitute the final population for the generation.[2]

An encoded solution is called *chromosome* when talking about Genetic Algorithms, and the information units that compose them are called *genes*. Therefore we can consider the chromosome $[1, 4, 3, 2]$, which is composed of the set of genes $\{1, 2, 3, 4\}$ in a particular order.

As exposed before, a Genetic Algorithm needs some parameters, such as the mutation probability or the crossover probability. Some of them are given the initial population whereas others implement a process to generate it randomly or by means of a heuristic. There are plenty of variations in the initial population building process.

A generic pseudocode can be seen in Algorithm 3. Note that this is indeed an example, whose behaviour can be redefined in several ways. For instance, in Algorithm 3 a pair of parents can only generate one child, but they could generate two in other particular implementations. The Genetic Algorithm shown is also using a number of generations as stopping criteria but this could be any other.

Note that the GA works with chromosomes, that is encoded solutions, but to compute the fitness value of the individuals it is needed to decode them. Because of that, in Algorithm 3 we can see "evaluatePopulation". It is at that point when all the chromosomes are decoded into actual solutions, and their associated fitness value is computed, making those values available for the replacement operation.

[2] A more technical explanation can be found at (9).

Algorithm 3 Genetic Algorithm example.

```
Input: $N_{gen}, P_{size}, p_m, p_c$
  $Population = generateInitialPopulation(P_{size})$;
  $evaluatePopulation(Population)$;
  $Idv_{best} = getBestIndividual(Population)$;
  $i = 0$;
  for $i \leftarrow 0, ..., N_{gen}$ do
    $ParentsPairSet = SelectionOperator(Population)$;
    $Children = \emptyset$;
    for $Parent_1, Parent_2 \leftarrow ParentsPairSet$ do
      $Child = crossingOperator(Parent_1, Parent_2, p_c)$;
      $mutationOperator(Child, p_m)$;
      $Children = Children \cup Child$;
    end for
    $evaluatePopulation(Children)$;
    $Population = Replacement(Population \cup Children)$;
    $Idv_{best} = getBestIndividual(Population)$;
  end for
Output: $Idv_{best}$
```

3.5 Greedy Algorithms

Greedy Algorithms are efficient approaches often used in optimization problems. They are basically constructed over a simple idea such as “Classify the exams with larger duration first”, which implies a straightforward implementation process and typically makes them deterministic. Because of that they are largely used, even though they are not guaranteed to provide an optimal solution to hard problems.

A formal explanation to the above-mentioned implementation would be to consider a finite set of values $V = \{v_1, v_2, ..., v_n\}$, that must be present on a candidate solution S.

A Greedy Algorithm will try to compute a solution considering initially an empty set, and usually by means of a local heuristic iteratively adds elements from set V to it. It never backtracks or makes any replacement to the already added values. This can be seen in Algorithm 4.

Two core points need to be considered when designing a Greedy Algorithm:

- **The solution must be a set of elements.** By having such stated, the partial solutions that the algorithm will handle can be seen as a subset of elements.
- **The local heuristic to guide the algorithm.** This would lead the algorithm by pointing which is the best value to be selected next. As stated before it points to the best locally, which does not imply that we are getting a better solution globally.

Algorithm 4 Generic Greedy Algorithm example.

```
  $S = \{\}$; /* Initial empty solution */
  $V_{unplaced} = \{v_1, v_2, ..., vn\}$; /* Initial values set */
  $i = 0$;
  while $|V_{unplaced}| > 0$ do
    $computeLocalHeuristic(V_{unplaced})$;
    $v_i = getBestValue(V_{unplaced})$;
    $S = S \cup v_i$;
    $V_{unplaced} = V_{unplaced} - \{v_i\}$;
  end while
Output: $S$
```

Chapter 4

Problem Definition

4.1 Motivation

Every year the deputy director of the University of Oviedo School of Computer Science is in charge of making the exam schedules for the three official university calls. For each of them a series of exams and deliveries must be considered, some of which may be online while others are in person.

For each call around seventy events must be scheduled in a time lapse of approximately 3 weeks. This may be seen as a problem that can be even be solved in a random way, but there are many constraints to be considered.

The first and most important is that according to the university rules, the school must ensure as far as possible that two exams of the same academic year are not scheduled on the same day.[1]

Furthermore subject coordinators may ask the deputy director to place their exams on specific dates, or establish relative dependencies with other tests. For instance, it would be desirable that it was possible to state a minimum time distance between two exams. Thus, the problem is not only placing the exams but also doing it in such a way that all or the maximum number of constraints are satisfied, which can be really complex due to the "dependency chains"[2] that are built among the exams.

This problem is now being solved in a fully manual approach by the deputy director. Even though there is no exact documentation of how high the number of constraints could be, we could make some estimations. Considering a case in which each exam has a constraint, we would be talking about around 70 constraints to fulfil, many of which will be linked in a dependency graph. This can get really complex even with a low amount of constraints, specially for a human trying to solve it without any kind of help.

Because the current procedure is prone to errors and to a large time investment from the user, this project is motivated to simplify the task. The aim is to create and implement an algorithm that provides as a result a group of schedules for the user to choose from. Ideally the user's task will be transformed into specifying the constraints before the execution, and choosing from the given options the one preferred.

[1] By default two exams of the same academic year must not be scheduled on the same day, however because of time issues, the number of events (exams and deliveries) of an academic year may exceed the total number of available days. In such case the constraint is relaxed a bit as commented in Section 4.3.1.

[2] Exam A must be the same day that exam B, which must be on the same day that exam C, which must be before a given date X and cannot be scheduled on a day Y.

4.2 Problem Description

The University of Oviedo School of Computer Science needs to schedule the exam calendar for each university call. Each call is independent from the rest of calls and has its own exam schedule. Thus, the problem can be reduced to solving it in just one call. Solving it for the three calls will be a matter of solving three particular problem instances.

4.2.1 Formal description of the problem

In this problem three sets are received as input: the set of exams to be scheduled $E = \{e_1, e_2, ..., e_n\}$, the set of days on which those exams can be placed $D = \{d_1, d_2, ..., d_m\}$, and the set of constraints that the final schedule must fulfill $C = \{c_1, c_2, ..., c_k\}$.

Each day has a time interval, bounded by two hours, in which exams can be placed, and it is mandatory that all exams scheduled that day begin and end within that time interval. We would define it as $Ti_{d_i} = [h_{start}, h_{end}]$, which stands as the Time Interval Ti for a day d_i in the set of days D.

The aim of the problem is that every exam in set E has assigned a day, a starting hour and an ending hour. This must be done in such a way that all the hard constraints on set C are fulfilled and within the time availability marked by days in set D. Furthermore as many as possible soft constraints must also be fulfilled.

4.2.2 Rounds

The School of Computer Science offers a bilingual option for the Degree in Software Engineering, which means that a large number of the subjects in such a degree can be studied in Spanish or in English. This means that the exams must also be in the selected language. Normally students of both languages take the exams at the same time and the professors give them the test and speak to them in the corresponding language. However this is not always possible, which leads to having two different time events for the same exam. In our problem this will be considered as two different exams even though they are indeed the "same".

For this particular problem, which also arises when not having enough resources to test all the students at the same time,[3] rounds are considered. A round is a group of exams that must take place the same day because they are linked. This is used to represent that two different exams are the same but that cannot be carried out as a single time event.

4.2.3 Elective subjects

Nine elective subjects are offered in the Degree in Software Engineering. These subjects are the only ones that can be taken for the first time both in third and fourth academic years, which means that they do not belong to a single one.

The events related to this subjects are free of fulfilling the university rule previously commented in Section 4.1 and that will be listed afterwards in Section 4.3.1.

[3] An example of this could be the lack of computers in the laboratory.

4.2.4 Already placed exams

It is common for the user to have some events already scheduled, even before starting making the actual exam calendar. This implies that the algorithm to be developed should be able to start from partial solutions and do not modify what was given as input.

This ultimately implies that the user must be able to take a partial solution of one produced by the algorithm and use it as an input for the algorithm.

4.2.5 Day time interval for the exams

Not every day in the exam calendar must have the same time interval devoted to the exams. There can be some cases in which maybe only the morning is available, while in others only the afternoon is. This can happen for many reasons such as it could be that the school is hosting an event or that there is some construction that is scheduled that day. For such reason, while a day d_i can have the exam time interval $Ti_{d_i} = [9:00, 21:00]$, other day d_j could have the time interval $Ti_{d_i} = [11:00, 18:00]$.

An exam can only be scheduled on a day if it fits within the time interval of that day.

4.2.6 Rest time

Along with the fulfilling of requests and rules, there are also some circumstances that would be desirable. The time interval in which exams are placed along the day is usually from 9:00 to 21:00,[4] but it is common to leave an interval of one or two hours between 13:00 and 15:00[5] so that both examiners and students can have lunch and rest.

An exam cannot start during this time rest time interval, but it could happen that an exam starts before the initial bound of the interval and ends in it or afterwards. In any case, it is desirable to prioritize solutions in which this time interval is as exam free as possible.

4.2.7 Numerical complexity

It is desired by the school to spread as much as possible the exams that require the biggest study loads. This now is done in a subjective way, and in the end it can be very complex to consider, also taking into account the other constraints.

For such a reason, solutions in which exams with the same numerical complexity, which will be a value provided by the user before the execution, are spread should be prioritized.

4.2.8 Extra time for the exams

It is not within the scope of this project to assign the exams to the physical resources that the school has, but due to the Covid-19 pandemic, some special measurements needed to be taken.

After the on site exams a time lapse of around ten minutes was used to clean the classroom or laboratory. For such a reason each exam can have a given extra time that will be necessary to

[4] This is the typical scenario, but it can be configured for each day individually as stated before.

[5] Note that this interval can also be configured by the user before the execution of the algorithm.

consider as part of its duration.

4.2.9 Deliveries

The schedule is not only used to hold exam dates, but also to state deadlines to some deliveries, like projects or applications. Because of the nature of this kind of event, it is not a problem that a delivery deadline is placed at the same time as other events. In fact, these deliveries are usually stated beforehand the actual exam calendar computing, even out of the day interval time.

4.3 Problem Constraints

There is a well-known classification for constraints that marks them as *hard* or *soft*. The difference is unfulfilling a hard constraint makes the whole candidate solution invalid. This do not happen with the soft ones, whose fulfilment is linked to the quality of the solution.

In our problem, constraints have been identified for both types of that classification, but there was a group of them which was not so clear; these are the *user* constraints. This kind of constraints can be soft or hard, depending on the context, that is, the user input.

Consider the following example: "The user receives a call from the Subject A lecturer in which he says that he has a medical appointment on a day D". In this case Subject A's exam simply cannot be put on day D, because there will be no examiner. This is a *hard* constraint.

Now consider this other example: "The user receives a call from the Subject A lecturer in which he says that he would like not to have the Subject A's exam on day D, but that he could attend even if so.". In this case even though there is a preference for the exam not to be on day D, the exam can take place on day D. This is a *soft* constraint.

Even though this special kind of constraints could be considered twice, in a soft version and in a hard version, that could be misleading because it is in fact the same constraint, the same logic. Because of that it was decided to classify them in their own category, being such *user* constraints.

4.3.1 Hard constraints

1. Two exams of the same year cannot take place on the same day unless there is no other possibility. Under no circumstances can two exams of the same course and semester take place on the same day.
2. Exams cannot start in the specified resting interval.
3. Exams must be scheduled within the available time interval of the day in which they will take place.

4.3.2 Soft constraints

1. All the exams must be scheduled.
2. The resting interval should be as free as possible.

3. Exams with a value assigned to their numerical complexity should be as separated as possible of other exams with the same value assigned.

The first one could be seen as a hard one, thinking that the final solution is not valid since not all the exams were scheduled. In this case, a solution with an unscheduled exam could be highly penalized, but could indeed be used as an indicator that maybe there were too many hard constraints given as input.

4.3.3 User constraints

This category groups constraints that can be considered hard or soft depending on the circumstances. They are "variable". One way of modeling them could be listing them in both categories: soft and hard, but it is clearer to group and address them all by a common name.

1. **Day Banned (DB):** An exam e cannot be on a day d.
2. **Day Interval (DI):** An exam e must be scheduled between day d_1 and day d_2.
3. **Same Day (SD):** The exams e_1 and e_2 must take place on the same day.
4. **Different Day (DD):** The exams e_1 and e_2 must take place on different days.
5. **Order Exams (OE):** The exam e_1 must be scheduled before the exam e_2.
6. **Time Displacement (TD):** The exam e_2 must be scheduled a number of days n_{days} after the exam e_1.

This user constraints were identified in several meetings with the user responsible of making the exams calendar at the School.

4.4 Problem Instance Example

In this section a simple problem instance and a possible solution will be commented. For the sake of simplicity, it will consist in a small pool of exams, days and user constraints. Numerical complexity, and extra times will not be applied, and the resting interval will be the default one, from 13:00 to 15:00.

4.4.1 Calendar

First of all, the days in which the exams need to be scheduled are shown in Table 4.1.

Date	Day Initial Hour	Day Ending Hour
2022-07-01	9:00	16:00
2022-07-02	15:00	20:00
2022-07-02	11:00	17:00

Table 4.1: Problem instance example - calendar.

4.4.2 Exams

In second place the exams that need to be scheduled are shown in Table 4.2. No deliveries are included in this example, to simplify things. For the sake of clarity in the next section, assume that $(ID)_{e_i} = i$, thus we can refer to the first exam which has $ID = 1$ as e_1.

ID	Subject	Duration (h)	Date	Starting hour	Ending hour
1	Subject1	1	-	-	-
2	Subject2	4	2022-07-01	9:00	13:00
3	Subject3	2	-	-	-
4	Subject4	3	2022-07-02	15:00	18:00
5	Subject5	1	-	-	-

Table 4.2: Problem instance example - exams.

4.4.3 User constraints

In addition to the hard and soft constraints stated in Sections 4.3.1 and 4.3.2, in this example the following user constraints will be considered. Note that the ones in bold are marked as hard.

1. **e_2 must be scheduled on the same day as e_5**
2. e_3 must be scheduled two days after than e_5
3. **e_1 cannot be scheduled on 2022-07-02.**

4.4.4 Proposed solution

In Table 4.3 a solution for this problem instance is proposed. This is one of the best solutions that could be provided, since there is no occupation of the resting interval, no soft user constraints unfulfilled, and no exams unscheduled.

As stated previously, a solution is valid when all the hard constraints are fulfilled. Therefore, a solution such as the one in Table 4.4 is also acceptable, but it is not as good as the one in Table 4.3.

ID	Subject	Duration (h)	Date	Starting hour	Ending hour
1	Subject1	1	2022-07-03	15:00	16:00
2	Subject2	4	2022-07-01	9:00	13:00
3	Subject3	2	2022-07-03	11:00	13:00
4	Subject4	3	2022-07-02	15:00	18:00
5	Subject5	1	2022-07-01	15:00	16:00

Table 4.3: Problem instance example - best solution.

ID	Subject	Duration (h)	Date	Starting hour	Ending hour
1	Subject1	1	2022-07-03	11:00	12:00
2	Subject2	4	2022-07-01	9:00	13:00
3	Subject3	2	2022-07-02	18:00	20:00
4	Subject4	3	2022-07-02	15:00	18:00
5	Subject5	1	2022-07-01	15:00	16:00

Table 4.4: Problem instance example - acceptable solution.

Chapter 5

Proposed Solution

As commented previously there are many approaches that can be followed in order to find a solution for the problem. Since most of the timetabling problems are NP-Hard and it is not known an algorithm that always obtains optimal solutions in polynomial time. The goal of our solution will be minimizing certain facts that decrease the quality of the solutions, such as the number of unfulfilled user constraints.

To solve the problem both a Genetic Algorithm and a Greedy Algorithm will be used. The Genetic Algorithm will be in charge of exploring the search space, while the Greedy Algorithm will evaluate the solutions and guide the search towards valid ones.[1]

5.1 Search Space

5.1.1 Exam schedule

As stated before, the goal of the problem is to assign a date, a starting hour, and a ending hour to every exam e_i, within the time availability stated on the set of viable days D, and fulfilling all hard constraints on C.

Considering that, we can represent an exam assignment as a 4-tuple composed by the scheduled exam, the day in which the exam is placed, the starting time of the exam, and the ending time of the exam.

$$(e_i, d_j, h_{start}, h_{end}) \tag{5.1}$$

We can define the exam time interval as $Ti_{e_i} = [h_{start}, h_{end}]$, and $Ti_{e_i} \subseteq Ti_{d_j}$ must hold.

A solution for the problem would be a set containing all the exam schedules made. The process may start with an *empty solution* or with a *partial solution.* In a empty solution all the exam assignments will be incomplete, which means that they do not have defined the three last elements of the tuple. On a partial solution, there will be a mix of complete and incomplete exam assignments.

$$\text{Incomplete exam assignment} = (e_i, -, -, -) \tag{5.2}$$

[1] A *valid* solution fulfils all the hard constraints.

In a complete solution all the exam assignments will be completed, that is all the 4-tuples will have all the values assigned.

5.1.2 State definition

A state is represented as a partial solution of the problem, in which not all the exam assignments are made. The problem is likely to start with a partial solution although an empty solution can also be a starting point.

Either way, there will be several middle steps in which partial solutions of the problem will be evaluated until getting to the final state in which a complete solution is reached.

5.1.3 State expansion

A state is expanded when scheduling an exam in an incomplete exam schedule, that is from a partial solution of the problem. That implies that for an exam schedule $Es = (e_i, d_j, h_{start}, h_{end})$, the number of new states that can be derived from it can be large.

The main reason of that is the hour bounds, because there are many combinations that can be done in just a single day. For instance, it is not the same for an exam to start at 9:00 than at 9:01.

Consider a calendar $D = \{d_1, d_2, ..., d_n\}$, a day d_j such that $d_j \in D$ and that Ti_{d_j} is usually 12 hours long. If we divide such hours in intervals of 5 minutes, that would result in 12 per hour. This interval in division is considered because it is unlikely for an exam to start on a minute or have a duration in minutes which is not a multiple of 5.

Taking that example into account we could approximate the number of possible new states as:

$$\text{Number of new possible states} = 12 * 12 * |D| \tag{5.3}$$

Since typically $|D| \in [15, 25]$, this implies a number of **3600** possible states. However, considering no such intervals, and the typical maximum value of $size(D) = 25$ that would lead us to:

$$\text{Number of new possible states} = 12 * 60 * 25 = 16000 \tag{5.4}$$

A Greedy Algorithm will be used to guide the search deterministically to valid and promising areas of the search space.

As a seed for this process, the Greedy Algorithm will be using the chromosomes of the individuals of a Genetic Algorithm, which would represent the order in which the Greedy Algorithm must schedule the exams according to a heuristic. This way by means of the Genetic Algorithm, the set of seeds will tend along the iterations to better solutions.

5.1.4 Instances complexity and solution existence

Complexity can vary drastically among instances of the problem. There are many factors that affect it which are not as simple as the number of exams. Available time to place the exams can also be seen an important factor, the less time we have the more probabilities that an exam is left unscheduled. However, what truly affects the complexity is not the amount of available time by itself but the proportion of exams and available time that we have, the less time we have to place the same number of exams then the harder it gets to schedule all the exams.

The number of user hard constraints can have a similar impact, making some exams impossible to schedule. Dependencies among the constraints must also be considered, since an inconsistency loop can be generated.

For example, suppose that it is stated that exams e_1 and e_2 must take place on the same day. Later, when writing the constraints associated with another exam e_3, it is stated that e_2 and e_3 must take on place on the same day, and that e_1 and e_3 must take place on different days. This inconsistency will lead directly to an incomplete solution, because one of those three exams will not be scheduled due to the stated hard constraints.

5.1.5 Exam collisions

Although the formulation of the problem does not state that exams collide, that is, that two exams cannot be placed at the same time, in reality that is desired.

Taking into account previous exam calendars of the school, it is always possible with a relevant margin for all the exams to be placed without superposing them.

There is one exception though, being such the deliveries. Deliveries are deadlines that typically do not imply any student attendance. Such particular events can be placed at the same time as other exams if wanted by the user. This will be a configurable option.

5.2 Greedy Algorithm

The Greedy Algorithm is an algorithm that schedules the incomplete exam assignments of a partial or empty provided solution in a deterministic way.

The only assumption that the algorithm makes is that exams must not overlap, with the exception of the deliveries if such option is disabled for them.

The algorithm will receive two inputs: the Individual (chromosome) that is going to be used as seed, and an object with the configuration and data of the execution. This second object has the information about which days are available, as well as the constraints of the particular problem instance.

The ordered list of exams to be scheduled will be obtained from the individual, while all calendar days, as well as their time interval initial hour will be loaded to a data structure, a map with the day as key and the available time interval associated to it as value is used. Once that initialization is done, the algorithm will try to schedule the exams one by one. This whole process can be seen in Algorithm 5. *getOrderedExamList* provides an ordered list of exams getting them from *examSchedule* and sorting them as specified in the chromosome of *individual.* Since there exist only one instance of each exam, and such instances are stored in *examSchedule*, there is no need for a return value in Algorithm 5.

The Greedy Algorithm also uses a repairing process to improve its effectiveness which is detailed in Section5.2.1. In Algorithm 5 the last zero in the call to *scheduleExam* means that in the basic version, that process is disabled.

Algorithm 5 Greedy Algorithm general view.

```
Input: $individual, examSchedule$
  $daysTimes = initializeDays(examsSchedule)$;
  $exams = getOrderedExamList(individual, examSchedule)$;
  for $exam \in exams$ do
    /* The 0 is the current depth of the repairing process. */
    $scheduleExam(exam, daysTimes, 0)$;
  end for
```

To schedule an exam the first thing that is needed to be computed is the set of days in which such an exam can be placed, that is a subset of the available days in which placing this exam will not lead to an unfulfilled hard constraint. To the hard constraints specified in Section 4.3.1, we must add the user constraints that the user has marked as hard for the execution. Once that set of days is computed, the algorithm tries to place the exam on the first valid date and the first possible hour which are tracked at the variable “daysTimes”.

Once an hour is selected the algorithm checks that it is not in the resting interval, where exams cannot start, and that the exam will not collide with another one in case it is placed there. In Algorithm 6 these checks can be seen in the *while* condition. *checkForCollision* method returns an exam that collides with the one that we are trying to place if there exists, which is the reason for which a *null* appears in the condition, to check there was such an exam.

In case the exam cannot be placed at the considered hour, this hour will be updated with the ending hour of the resting interval or the exam that was colliding with ours. The algorithm will try to follow the same process until placing the exam or finding that the exam does not fit in the current day. If the last thing occurs it will try with the following viable day for the exam.

This whole process can be seen in Algorithm 6. For the sake of clarity what is shown is the general process of the algorithm, a number of specific checks such as if the exam is a delivery and if delivery collisions are enabled, are not included in the pseudocode. It is returned *true* if the exam was scheduled, and false if it was not.

5.2.1 Repairing process

The Greedy Algorithm explained until now in Algorithms 5 and 6 is fully functional and can be used in the problem effectively. However, as the number of hard constraints is increased it will tend to leave some exams unscheduled because the small pool of days in which they can be placed is already occupied. In Algorithm 6, this means that *viableDays* is too small. In this scenario in which an exam has a small set of viable days to be placed, the repairing algorithm is crucial, since without it those exams would probably be left as unscheduled.

The repairing algorithm will be built based on a single principle:

If the set of viable days for an exam e_i is already fully occupied, we could try to find if some of the already placed exams in such days can be swapped with e_i. That is, finding an exam e_j in those days that could be scheduled on a different available day and placing e_1 in the position of e_j.

The previous algorithms used a variable called “repDepth” which stands for *repairingDepth.* To implement the commented principle a recursive procedure is used. This approach leads to a tree of temporal swaps whose maximum depth is limited by means of that variable “repDepth”. Users will be able to specify in configuration files how deep can this tree be.

Algorithm 6 Schedule exam method.

```
function scheduleExam($exam, daysTimes, repDepth$)
  $viableDays = computeViableDaysFor(exam)$;
  for $day \in viableDays$ do
    $tempHour = getFirstPossibleHour(daysTimes, day)$;
    while $isHourInRestingInterval(tempHour)$ or $(collidingExam = checkForCollision(exam, day, tempHour)) \neq null$ do

        if $collidingExam \neq null$ then
            $daysTimes = updateDayTime(daysTimes, collidingExam.getFinishingHour())$;

            $tempHour = getFirstPossibleHour(daysTimes, day)$;
        end if

        /* Order matters, the colliding exam could end on the resting interval. */
        if $isHourInRestingInterval(tempHour)$ then
            $daysTimes = updateHourForDay(daysTimes, getRestingTimeFinishingHour())$;
        end if

        $tempHour = getFirstPossibleHour(daysTimes, day)$;
    end while

    if $endsOnTime(day, exam, tempHour)$ then
        $scheduleExam(exam, day, tempHour)$;
        $updateDayTime(daysTimes, exam.getFinishingHour())$;
        $break$;
    end if
  end for

  /* Repairing process (Algorithm 7) would be placed here. */

  return exam.wasScheduled();
end function
```

The repairing process can be seen in Algorithm 7. Note that as stated in Algorithm 6, the following code belongs there, it is not a method by itself.

Algorithm 7 Repairing process.

```
if $\neg exam.wasScheduled$ and $repDepth < getMaxRepDepth()$ then
  $exCandidates = getSwappableExamsOfOver(exam, viableDays)$;
  for $examCandidate \in examCandidates$ do
    $tDate = examCandidate.getDate()$;
    $tHour = examCandidate.getHour()$;
    $exam.scheduleFor(day, tempHour)$;
    $examCandidate.unSchedule()$;
    $daysTimesCopy = cloneMap(daysTimes)$
    if $scheduleExam(examCandidate, daysTimesCopy, repDepth + 1)$ then
      $updateDaysTimes(daysTimes, daysTimesCopy)$;
      $break$;
    else
      $exam.unSchedule()$;
      $examCandidate.scheduleFor(tDate, tHour)$;
    end if
  end for
end if
```

5.2.2 Full view of the algorithm

The whole pseudocode of the schedule exam method is provided in Algorithm 8. This pseudocode those shown in Algorithms 6 and 7, and it is used in Algorithm 5.

5.3 Genetic Algorithm

5.3.1 Introduction

The Greedy Algorithm needs a "seed" to start and come up with a candidate solution. Those seeds are the chromosomes of the individuals handled by a Genetic Algorithm whose purpose is to conduct a population of individuals to a promising area of the search space.

The chromosomes, that is the seeds, are formed by a sequence of numbers which are unique identifiers of exams. That way the meaning of such sequence is the order in which the exams must be scheduled by the Greedy Algorithm.

This Genetic Algorithm receives the following parameters:

- p_m, the mutation probability.
- p_c, the crossover probability.
- Pop_{size}, the size of the population of individuals.
- N_{gen}, the number of generations to be done.

Although the values are fully configurable by the user, some tuning experiments have been done over several scenarios and problem instances to provide a recommended configuration.

Algorithm 8 Schedule exam method (complete).

```
function scheduleExam($exam, daysTimes, repDepth$)
  $viableDays = computeViableDaysFor(exam)$;
  for $day \in viableDays$ do
    $tempHour = getFirstPossibleHour(daysTimes, day)$;
    while $isHourInRestingInterval(tempHour)$ or $(collidingExam = checkForCollision(exam, day, tempHour)) \neq null$ do

      if $collidingExam \neq null$ then
        $daysTimes = updateDayTime(daysTimes, collidingExam.getFinishingHour())$;

        $tempHour = getFirstPossibleHour(daysTimes, day)$;
      end if

      /* Order matters, the colliding exam could end on the resting interval. */
      if $isHourInRestingInterval(tempHour)$ then
        $daysTimes = updateHourForDay(daysTimes, getRestingTimeFinishingHour())$;
      end if

      $tempHour = getFirstPossibleHour(daysTimes, day)$;
    end while

    if $endsOnTime(day, exam, tempHour)$ then
      $exam.scheduleFor(day, tempHour)$;
      $updateDayTime(daysTimes, exam.getFinishingHour())$;
      $break$;
    end if
  end for

  /* Repairing algorithm code */
  if $\neg exam.wasScheduled$ and $repDepth < getMaxRepDepth()$ then
    $exCandidates = getSwappableExamsOfOver(exam, viableDays)$;
    for $examCandidate \in examCandidates$ do
      $tDate = examCandidate.getDate()$;
      $tHour = examCandidate.getHour()$;
      $exam.scheduleFor(day, tempHour)$;
      $examCandidate.unSchedule()$;
      $daysTimesCopy = cloneMap(daysTimes)$
      if $scheduleExam(examCandidate, daysTimesCopy, repDepth + 1)$ then
        $updateDaysTimes(daysTimes, daysTimesCopy)$;
        $break$;
      else
        $exam.unSchedule()$;
        $examCandidate.scheduleFor(tDate, tHour)$;
      end if
    end for
  end if

  return exam.wasScheduled();
end function
```

5.3.2 General behaviour

The Genetic Algorithm generates the initial population randomly considering permutations of the set of exams to be scheduled.

The stopping criteria for the algorithm is the number of generations N_{gen}. Until reached, the algorithm will repeat the following process.

In this solution children are generated until reaching Pop_{size}. Only one child is generated from each pair of parents, so at least Pop_{size} pairs will be selected by the Selection Operator to do the crossover by means of the Crossover Operator. When selecting a pair of parents p_c represents the probability of such selected pair to proceed to crossover.

When the Crossover Operator generates the child, the Mutation Operator may perform a mutation to the child depending on p_m.

The Replacement Operator consists in the children substituting the parents in the new generation. However, *elitism* is applied.

To evaluate how good each individual is, the Greedy Algorithm is executed to get the candidate solution. Once we have such solution the fitness function is computed giving as a result the fitness of the individual.

5.3.3 Enconding

As stated before, each individual is a chromosome with a particular order of exam identifiers, which states in which order must the Greedy Algorithm attempt to schedule them.

Initially the provided exam assignments list is considered. As stated before this is normally a partial solution although it could be an empty one. The encoding function must transform what can be seen in Equation 5.5, to one of the options presented in Equation 5.6.

$$[(e_1, d_1, h_{start_{e_1}}, h_{end_{e_1}}), (e_2, -, -, -), (e_3, -, -, -)] \tag{5.5}$$

Note that the first exam, e_1, is already scheduled. Since it is already scheduled by input we can change nothing about it, and so we must not encode it to the chromosomes. This is because it would be irrelevant data for the Genetic Algorithm. Since we have two exams to be scheduled $\{e_2, e_3\}$, in this case we could have two different chromosomes.The two possibilities can be seen in Equation 5.6, assuming $(ID)_{e_i} = i$.

$$[2, 3], [3, 2] \tag{5.6}$$

5.3.4 Evolutionary operators

Selection operator

The proposed selection operator selects an individual of the population with a roulette selection based on the individuals fitness.

Due to how the algorithm is built it will select at least $2 * Pop_{size}$ individuals. That is guaranteed because only one individual is generated from each pair of individuals by the chosen crossover operator.

Crossover operator

The crossover operator used is the Order Crossover operator (OX). This a well-known crossover operator which has given good results when working with chromosomes that represent permutations.

In the proposed implementation two random positions of the chromosome are selected to execute the operator logic, which was shown in Figure 3.2.

Mutation operator

The mutation operator implementation performs a value swap between two positions of the chromosome. Suppose that we have the chromosome shown in Equation 5.7.

$$[1, 4, 7, 6, 5, 3, 2] \tag{5.7}$$

One of the possible results after executing the mutation operator over chromosome in Equation 5.7, can be seen in Equation 5.8.

$$[1, \mathbf{3}, 7, 6, 5, \mathbf{4}, 2] \tag{5.8}$$

5.3.5 Replacement operator

The replacement policy was explained in Section 5.3.2. As commented there, children substitute their parents, but considering elitism.

This implementation even if it was the chosen one, was not the only one considered. The full replacement of children was tested but it led to random results, the algorithm was not able to converge towards a good area of the search space.

Other option was considered, which was joining the previous generation with the children of such generation, sorting that list by fitness and choosing the best fitness individuals. This alternative was pretty decent, but it set a too high selective pressure in the algorithm. The results were not bad, but were not as good as with the chosen alternative.

5.3.6 Fitness function

Some quality factors can be derived from Section 4.3. The key factors that need to be considered to evaluate the quality of the solutions are:

- **Number of unscheduled exams:** The aim of the algorithm is to find complete solutions, that is, solutions with all the exam schedules completed. Since this cannot be always achievable, solutions with less unscheduled exams must be strongly prioritized.

- **Resting interval free time:** It is given by the problem definition that no exam can start in the specified resting interval, however they can end in that interval. Since it is desirable for such interval to be as free as possible, the occupation of that time will be penalized in the fitness function.
- **Number of unfulfilled soft user constraints:** The fitness function must evaluate as better solutions the ones with fewer unfulfilled user constraints. Each type of user constraint will be measured separately, since a different weight may be assigned to each type.
- **Numerical complexity distance:** As a new functionality, it was desired to separate exams with a given complexity value. The fitness function must penalize that those exam are close to each other.

These factors can be evaluated in several different ways, which implies that there are many fitness functions that could be considered here.

In this case it has been decided to use a weighted linear function with those criteria as variables and being the user able to provide in a configuration file the weights to assign to each of them, that is the coefficients of the function. The fitness function can be seen in Equation 5.9.

An experimental study has been performed to get a recommended value for each of those coefficients, which are used in the default configuration. This configuration can be seen in Table 5.1.

$$\begin{aligned}\text{Fitness} = {} & \omega_{ue} * \text{numberOfUnscheduledExams} \\ & + \ \omega_{uc} * \text{numberOfUnfulfilledUserSoftConstraints} \\ & + \ \omega_{ri} * \text{numberOfOccupiedMinutesInRestingInterval} \\ & + \ \omega_{nc} * \text{computedNumericalComplexityFunction}\end{aligned} \tag{5.9}$$

Coefficient	Value
ω_{ue}	100000
ω_{uc}	140
ω_{ri}	3.5
ω_{nc}	2.5

Table 5.1: Default configuration of weights.

Normally Genetic Algorithms tend to maximize the fitness, but in this case it is going to be minimized. This is because the presented function consists in the addition of penalization values to the solution. Therefore, the less penalizations the individual has the better it is.

Single-objective-optimization

As can be seen in the presented function the project is considered a single-objective optimization. It would be possible to approach the Pareto Front with a multi-objective algorithm (5), but this would introduce complexity for user maintenance and extension.

Furthermore, as commented before, other Final Degree Projects (4) in the domain of the School of Computer Science Engineering have successfully achieved good solution by using single-objective optimization algorithms. Because of that, single-objective optimization was considered the best approach for the problem.

Numerical complexity function

Even though most of the factors commented in Section 5.3.6 are self-explanatory, numerical complexity is not. What is wanted to be achieved is that the fitness function penalize exams with the same complexity value for being close in the calendar.

The process followed to compute the value of this function is simple. First, all the exams are grouped according to their numerical complexity. This grouping is done because only exams with a complexity number A affect to exams with the same complexity number A. If the numerical complexity value is not provided for an exam, it will be treated as 0, which means that this functionality will be disabled for that exam.

After that each group is processed, meaning that every pair of exams within a group is evaluated to add its penalization value depending on the dates of that exam pair. The process until now can be seen in Algorithm 9.

Algorithm 9 Numerical Complexity Function. General iteration process.

```
function numericalComplexity($exams$)
  $accumulator = 0$;
  $examsGroupedByComplexity = groupExamsByNumericalComplexity(exams)$;
  for $(numComp_i, exams_{numComp_i}) \in examsGroupedByComplexity$ do
    $pairs = getPairs(exams_{numComp_i})$;
    for $pair \in pairs$ do
      $accumulator = accumulatePenalization(numComp_i, pair)$;
    end for
  end for
  return accumulator;
end function
```

Finally, *accumulatePenalization* checks if both exams of the pair are scheduled. If so, it counts the distance in days between both exams and uses the numerical complexity to compute something similar to the function shown in Equation 5.10. This function establishes more weight the higher the complexity and the lower the distance. However, this function is not defined when both exams are scheduled for the same day, which is the worst scenario possible and the one that should be highly penalized. Because of this, the piecewise function shown in Equation 5.11 will be considered to cover that particular scenario.

$$f(numComp_i, distanceInDays) = numComp_i/distanceInDays \tag{5.10}$$

$$f(a,b) = \begin{cases} a/b & \text{if b} > 0 \\ a^2 & \text{if b} = 0 \end{cases} \tag{5.11}$$

With that, *accumulatePenalization* can be seen in Algorithm 10.

Algorithm 10 Numerical Complexity Function. accumulatePenalization method.

```
function accumulatePenalization($c, examPair$)
  $penalization = 0$;
  if $isScheduledPair(examPair)$ then
    $distanceInDays = computeDistanceInDays(examPair)$;
    if $distanceInDays == 0$ then
      $penalization = nc^2$;
    else
      $penalization = nc/distanceInDays$;
    end if
  end if
  return $penalization$;
```

Chapter 6

Project scheduling and budget

6.1 Temporal schedule

This section shows the followed schedule. The project was started on September and it ended on April. In total, it took 720 hours of work.

The project was divided in ten phases, which are listed below. In addition to that, Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8 show the project schedule by means of Gantt Diagrams.

1 **Project planification (2021-08-21 / 2021-09-02):** In this phase the schedule of the project, the identification of the phases and estimation of times is done.

   1.1 First meeting with the Tutor (2021-08-21 / 2021-09-02)

   1.2 Definition of the project phases (2021-08-30 / 2021-08-31)

   1.3 Design of the documentation (2021-09-01 / 2021-08-02)

2 **Initial research (2021-09-03 / 2021-09-17):** Initially some knowledge about the topics linked to the project was necessary. This phase was aimed to obtain necessary knowledge before talking to the client.

   2.1 Gonzalo De La Cruz project review (2021-09-03 / 2021-09-06)

   2.2 Intelligent Systems Subject Metaheuristic materials review (2021-09-07 / 2021-09-08)

   2.3 Literature review (2021-09-09 / 2021-09-17)

      2.3.1 Metaheuristics (2021-09-09 / 2021-09-09)

      2.3.2 Evolutionary algorithms and Genetic Algorithms (2021-09-10 / 2021-09-13)

      2.3.3 Scheduling and Timetabling problems (2021-09-14 / 2021-09-15)

      2.3.4 University Exam Timetabling problems (2021-09-16 / 2021-09-17)

3 **Problem analysis (2021-09-21 / 2021-10-08):** A meeting with the client and the tutor is held. The problem is analysed, and the scope of the project is fixed.

   3.1 Requirements getting (2021-09-20 / 2021-09-22)

      3.1.1 Meeting with the Tutor and the client (2021-09-20 / 2021-09-20)

      3.1.2 Analysis and clarification of the taken notes during the meeting (2021-09-21 / 2021-09-22)

   3.2 Scope definition (2021-09-23 / 2021-09-23)

- **3.2.1** Meeting with the Tutor to talk about the scope of the system (2021-09-23 / 2021-09-23)
- **3.2.2** Analysis of the meeting conclusions (2021-09-23 / 2021-09-23)
- **3.3** System Requirements identification (2021-09-24 / 2021-09-28)

4 Proposed Solution (2021-09-29 / 2021-10-12): In this phase the problem is formalized and a theoretical resolution is proposed.

- **4.1** Problem formalization (2021-09-29 / 2021-09-30)
- **4.2** Genetic Algorithm design (2021-10-01 / 2021-10-01)
- **4.3** Greedy Algorithm design (2021-10-04 / 2021-10-06)
- **4.4** Repairing algorithm design (2021-10-07 / 2021-10-08)
- **4.5** Fitness function design (2021-10-11 / 2021-10-11)
- **4.6** Numerical Complexity function design (2021-10-12 / 2021-10-12)

5 Solution implementation (2021-10-13 / 2021-11-11): The theoretical problem solution designed in the previous phase is now put into a prototype application.

- **5.1** General system architecture definition (2021-10-13 / 2021-11-14)
- **5.2 Domain subsystem (2021-10-14 / 2021-11-10):** All the domain logic is done in the phase.
  - **5.2.1 Class design** (2021-10-15 / 2021-11-11)
    - **5.2.1.1** General structure (2021-10-15 / 2021-10-15)
    - **5.2.1.2** Constraints hierarchy (2021-10-18 / 2021-10-19)
    - **5.2.1.3** Parser structure (2021-10-20 / 2021-10-22)
  - **5.2.2 Implementation** (2021-10-25 / 2021-11-08)
    - **5.2.2.1** Exam Parser (2021-10-28 / 2021-11-02)
    - **5.2.2.2** Constraint Parser (2021-11-03 / 2021-11-08)
  - **5.2.3** Code documentation (2021-11-09 / 2021-11-11)
- **5.3 Configuration subsystem (2021-11-12 / 2021-11-24):** All the configuration options that the algorithm will have are identified, distributed among the configuration files, and the system to parse them is implemented.
  - **5.3.1** Identification of all configuration options (2021-10-12 / 2021-11-24)
  - **5.3.2** Class design (2021-11-12 / 2021-11-18)
  - **5.3.3** Implementation (2021-11-18 / 2021-11-22)
  - **5.3.4** Code documentation (2021-11-22 / 2021-11-24)
- **5.4 Genetic subsystem (2021-11-24 / 2021-12-08):** During this phase the genetic algorithm structure previously defined theoretically is transformed into an actual implementation.
  - **5.4.1 Class design (2021-11-24 / 2021-13-01)**
    - **5.4.1.1** Operators (2021-11-24 / 2021-11-26)
    - **5.4.1.2** Unfulfilled constraints counter (2021-11-26 / 2021-11-30)
    - **5.4.1.3** Fitness function (2021-11-30 / 2021-12-01)
  - **5.4.2** Implementation (2021-12-01 / 2021-12-06)
  - **5.4.3** Code documentation (2021-12-07 / 2021-12-08)
- **5.5 Greedy subsystem (2021-12-08 / 2021-12-14):** During this phase the greedy algorithm structure previously defined theoretically is transformed into an actual implementation.

5.5.1 Implementation (2021-12-08 / 2021-12-13)

5.5.2 Code documentation (2021-12-14 / 2021-12-14)

5.6 Logger subsystem (2021-12-15 / 2021-12-16): In this phase the files to be used to log, their structure and contents were designed.

5.6.1 General design (2021-12-15 / 2021-12-15)

5.6.2 Implementation (2021-12-08 / 2021-12-16)

5.6.3 Code documentation (2021-12-16 / 2021-12-16)

5.7 Console subsystem (2021-12-27 / 2021-12-28): During this phase the simple CLI (Console Line Interface) of the prototype is designed and implemented. Initially this phase was linked to the previous one, but due to the need to meet with the client, it was postponed after that.

5.7.1 User prompts implementation (2021-12-27 / 2021-12-27)

5.7.2 Error message implementation (2021-12-28 / 2021-12-28)

6 Client Alfa review (2021-12-17 / 2021-12-24): A meeting is held with the client to show the actual state of the prototype and ask for feedback. This phase also includes the necessary changes that needed to be performed.

6.1 Meeting with the client to review the prototype (2021-12-17 / 2021-12-17)

6.2 Identification of new changes to be done due to the meeting (2021-12-17 / 2021-12-17)

6.3 Rounds input change (2021-12-20 / 2021-12-22): This task was added due to the meeting with the client.

6.4 New input options in data file (2021-12-23 / 2021-12-24): This task was added due to the meeting with the client.

7 Real-shape instances generator (2021-12-28 / 2022-01-05): A tool to generate instances to test the prototype is designed and implemented.

7.1 Analysis of previous real-life instances (2021-12-29 / 2021-12-30)

7.2 Design of the tool (2021-12-21 / 2022-01-04)

7.3 Implementation of the tool (2022-01-05 / 2022-01-06)

7.4 Manual testing (2022-01-07 / 2022-01-07)

8 System tests (2022-01-06 / 2022-01-12): The tests that can be found in chapter 10 are designed and executed.

8.1 Test conditions study (2022-01-10 / 2022-01-10)

8.2 Identification of interface paths (2022-01-10 / 2022-01-10)

8.3 Tests design, and test cases generation (2022-01-11 / 2022-01-12)

8.4 Tests cases execution and bug fixes (2022-01-13 / 2022-01-14)

9 Experimental study (2022-01-17 / 2022-02-08): The prototype configurations are tuned, and experiments are performed over different expected scenarios to prove the effectiveness of the tool. This phase is composed of four different sub-phases which are the following:

9.1 Instances Scenarios (2022-01-17 / 2022-01-19): The scenarios to be used in the study are designed, and then generated by means of the implemented tool.

9.2 Genetic Algorithm parameter tuning Scenarios (2022-01-20 / 2022-01-28): Several combinations of the genetic parameters are tested. The best configuration is set as the default one in the algorithm.

- **9.3 Fitness Function coefficients (2022-01-28 / 2022-02-04):** Several combinations of coefficients are tested for the fitness function. The best configuration is set as the default one in the algorithm.
- **9.4 Comparative study (2022-02-04 / 2022-02-08):** The effectiveness of the tool is tested over the generated scenarios. The results are kept to be shown in the final documentation.

10 Final documentation (2022-02-08 / 2022-04-21): This phase groups all the tasks that needed to produce the final documentation of the project. A LaTeX template was provided and adapted to this project.

- **10.1 LaTeX Project Structure (2022-02-08 / 2022-02-15)**
  - **10.1.1** Base template Study (2022-02-08 / 2022-02-09)
  - **10.1.2** Title page design (2022-02-09 / 2022-02-11)
  - **10.1.3** Adapt template structure (2022-02-11 / 2022-02-15)
- **10.2 Chapters writing (2022-02-15 / 2022-04-21)**
  - **10.2.1** Project memory justification (2022-02-15 / 2022-02-16)
  - **10.2.2** Introduction (2022-02-16 / 2022-02-18)
  - **10.2.3 Theoretical explanation (2022-02-18 / 2022-02-28)**
    - **10.2.3.1** General alternatives to solve University Exam Scheduling problems (2022-02-18 / 2022-02-21)
    - **10.2.3.2** Metaheuristics (2022-02-21 / 2022-02-23)
    - **10.2.3.3** Evolutionary algorithms (2022-02-23 / 2022-02-24)
    - **10.2.3.4** Genetic Algorithms (2022-02-24 / 2022-02-28)
  - **10.2.4 Problem definition (2022-02-28 / 2022-03-04)**
    - **10.2.4.1** General motivation (2022-02-28 / 2022-02-28)
    - **10.2.4.2** Verbose description (2022-02-28 / 2022-02-03)
    - **10.2.4.3** Constraints identification and example (2022-03-03 / 2022-03-04)
  - **10.2.5 Problem proposed solution (2022-03-07 / 2022-03-17)**
    - **10.2.5.1** Formalization of the search space (2022-03-07 / 2022-03-07)
    - **10.2.5.2** Formalization of the state expansion (2022-03-07 / 2022-03-07)
    - **10.2.5.3** Greedy algorithm (2022-03-08 / 2022-03-14)
    - **10.2.5.4** Genetic algorithm (2022-03-15 / 2022-03-17)
  - **10.2.6 Scheduling and Budget (2022-03-18 / 2022-03-22)**
    - **10.2.6.1** Scheduling (2022-03-18 / 2022-03-18)
    - **10.2.6.2** Budget (2022-03-21 / 2022-03-22)
  - **10.2.7 System analysis (2022-03-23 / 2022-03-29)**
    - **10.2.7.1** Scope of the system (2022-03-23 / 2022-03-23)
    - **10.2.7.2** System actors (2022-03-23 / 2022-03-23)
    - **10.2.7.3** Requirements table (2022-03-24 / 2022-03-25)
    - **10.2.7.4** Subsystems identification (2022-03-28 / 2022-03-28)
    - **10.2.7.5** Preliminary class diagram (2022-03-28 / 2022-03-29)
    - **10.2.7.6** Subsystems description (2022-03-29 / 2022-03-29)
  - **10.2.8 System design (2022-03-30 / 2022-04-07)**
    - **10.2.8.1** Package diagram (2022-03-30 / 2022-03-30)
    - **10.2.8.2** Components diagram (2022-03-31 / 2022-03-31)
    - **10.2.8.3** General Class diagram (2022-04-01 / 2022-04-01)
    - **10.2.8.4** Genetic Algorithm Class diagram (2022-04-04 / 2022-04-04)
    - **10.2.8.5** Domain Class diagram (2022-04-04 / 2022-04-04)

10.2.8.6 Constraints Class diagram (2022-04-05 / 2022-04-05)

10.2.8.7 Fitness Function and Constraints Counter Class diagram (2022-04-06 / 2022-04-06)

10.2.8.8 Logger Class diagram (2022-04-07 / 2022-04-07)

10.2.8.9 User Interface design description (2022-04-07 / 2022-04-07)

10.2.9 System implementation (2022-04-08 / 2022-04-08)

10.2.9.1 Used tools description (2022-04-08 / 2022-04-08)

10.2.9.2 Difficulties found in the project (2022-04-08 / 2022-04-08)

10.2.10 System tests (2022-04-11 / 2022-04-13)

10.2.10.1 Genetic design and explanation (2022-04-11 / 2022-04-11)

10.2.10.2 Test cases details (2022-04-12 / 2022-04-12)

10.2.10.3 Auxiliary figures (2022-04-13 / 2022-04-13)

10.2.11 Experimental study (2022-04-14 / 2022-04-18)

10.2.11.1 Genetic tuning (2022-04-14 / 2022-04-14)

10.2.11.2 Fitness coefficients tuning (2022-04-14 / 2022-04-15)

10.2.11.3 Comparative study (2022-04-15 / 2022-04-18)

10.2.12 User manual (2022-04-18 / 2022-04-21)

10.2.12.1 Installation manual (2022-04-18 / 2022-04-28)

10.2.12.2 Configuration files (2022-04-19 / 2022-04-20)

10.2.12.3 Input data file (2022-04-20 / 2022-04-20)

10.2.12.4 Output files (2022-04-21 / 2022-04-21)

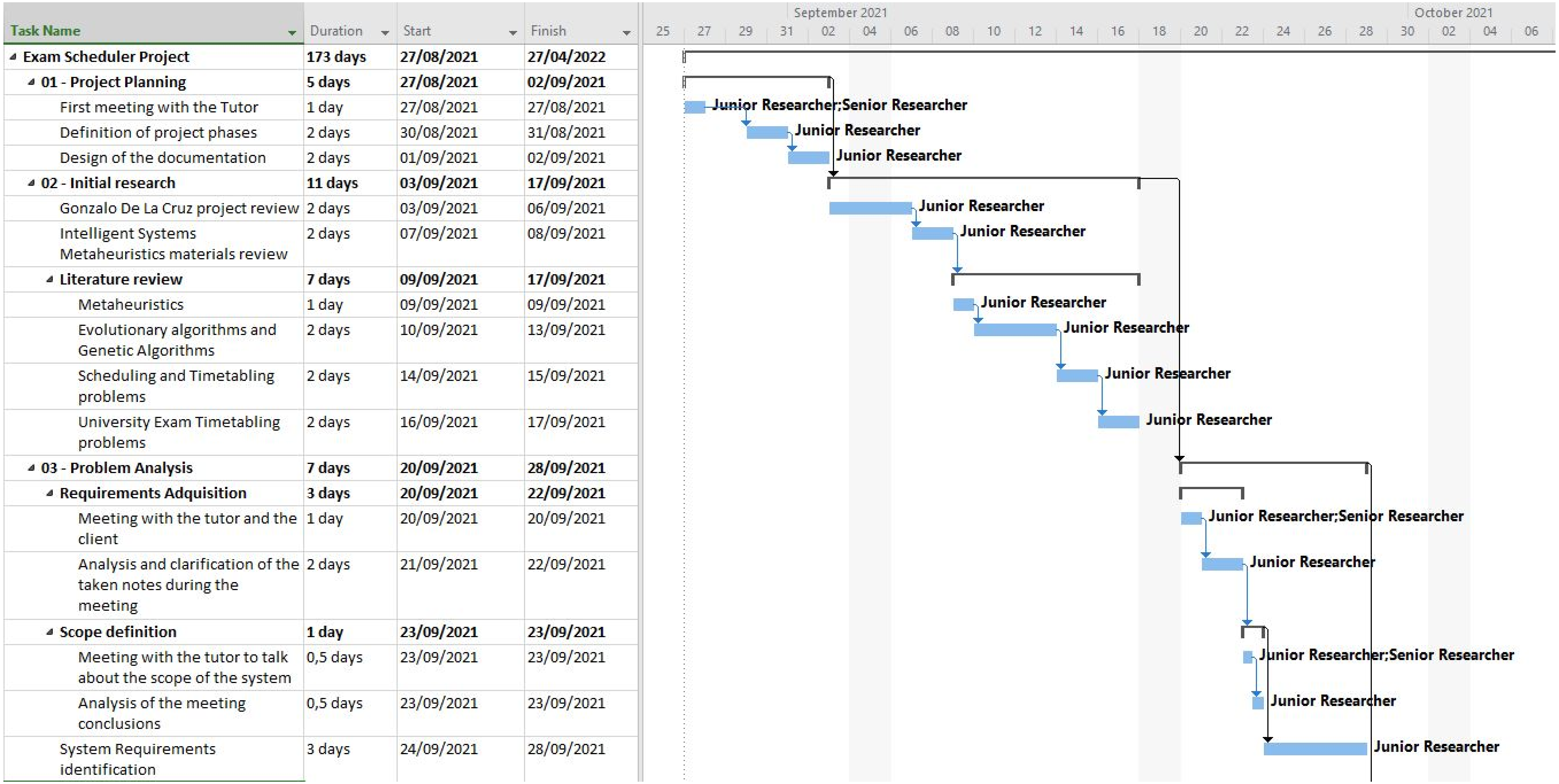
10.2.13 Conclusions (2022-04-21 / 2022-04-21)

11 Project Closure (2022-04-21 / 2022-04-21)

11.1 Final Documentation Review (2022-04-22 / 2022-04-26)

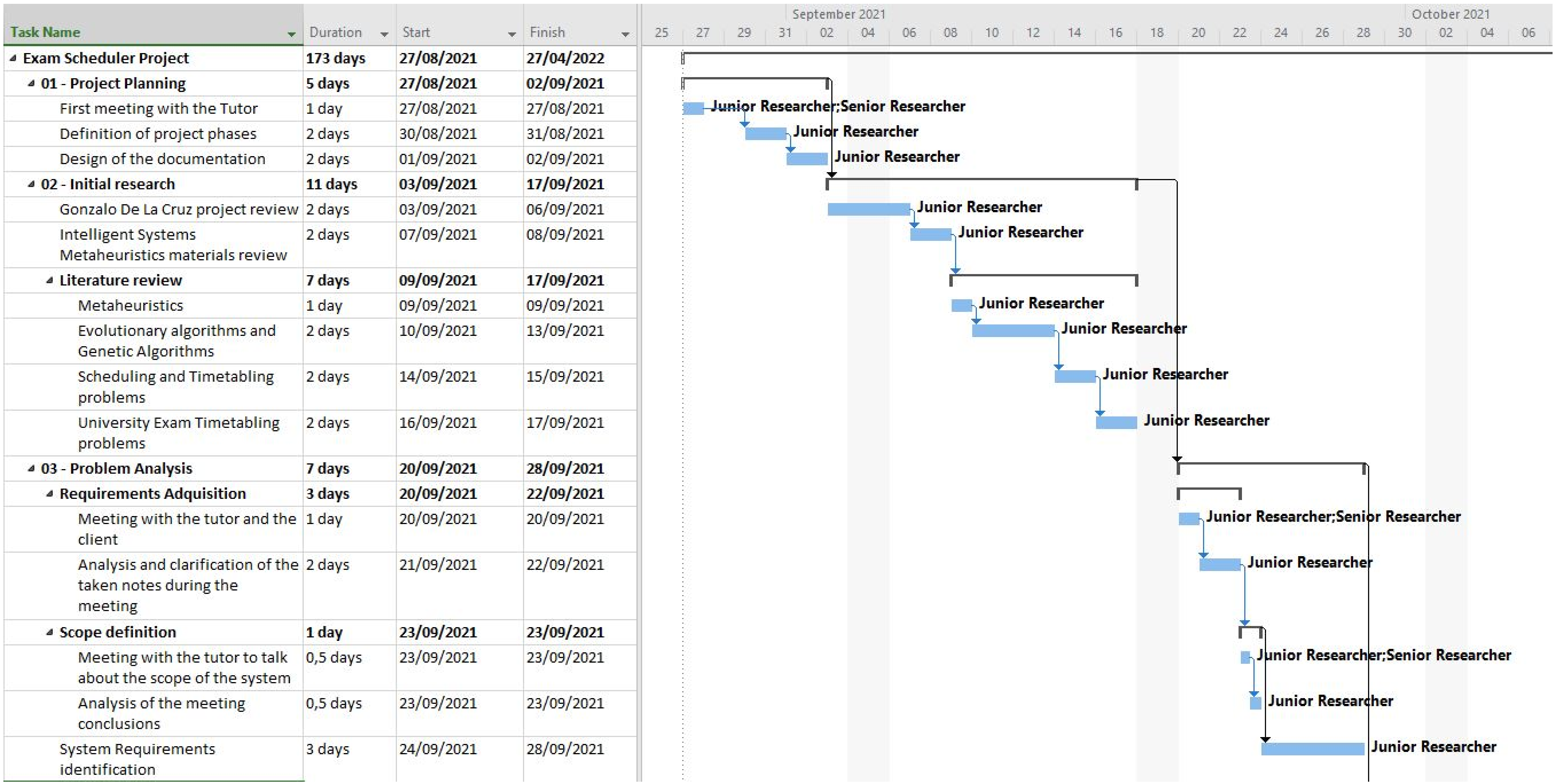
11.2 Final delivery (2022-04-27 / 2022-04-27)

The phases tasks breakdown are the following:



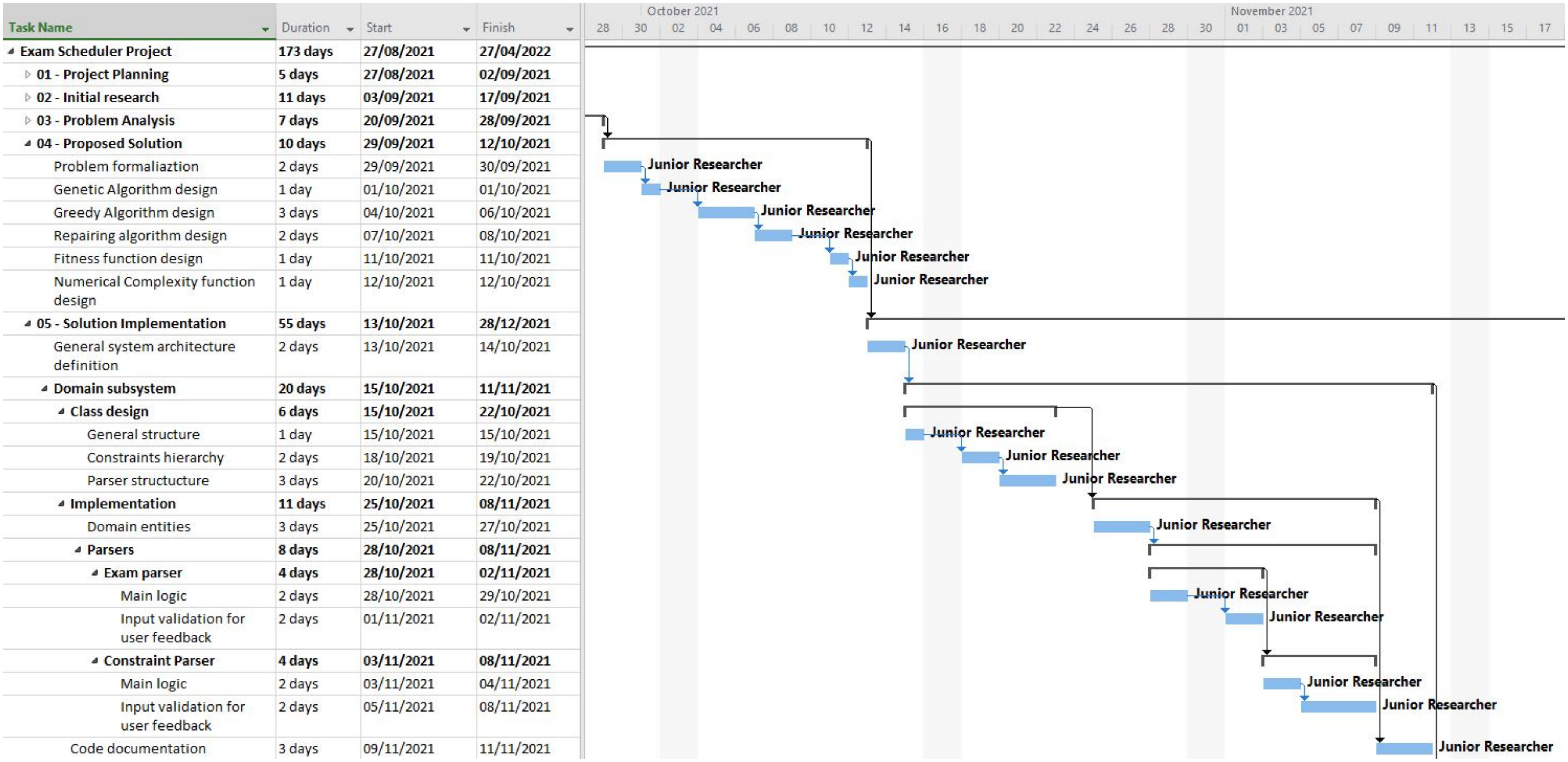
Task Name	Duration	Start	Finish
◢ Exam Scheduler Project	**173 days**	**27/08/2021**	**27/04/2022**
◢ 01 - Project Planning	**5 days**	**27/08/2021**	**02/09/2021**
First meeting with the Tutor	1 day	27/08/2021	27/08/2021
Definition of project phases	2 days	30/08/2021	31/08/2021
Design of the documentation	2 days	01/09/2021	02/09/2021
◢ 02 - Initial research	**11 days**	**03/09/2021**	**17/09/2021**
Gonzalo De La Cruz project review	2 days	03/09/2021	06/09/2021
Intelligent Systems Metaheuristics materials review	2 days	07/09/2021	08/09/2021
◢ Literature review	**7 days**	**09/09/2021**	**17/09/2021**
Metaheuristics	1 day	09/09/2021	09/09/2021
Evolutionary algorithms and Genetic Algorithms	2 days	10/09/2021	13/09/2021
Scheduling and Timetabling problems	2 days	14/09/2021	15/09/2021
University Exam Timetabling problems	2 days	16/09/2021	17/09/2021
◢ 03 - Problem Analysis	**7 days**	**20/09/2021**	**28/09/2021**
◢ Requirements Adquisition	**3 days**	**20/09/2021**	**22/09/2021**
Meeting with the tutor and the client	1 day	20/09/2021	20/09/2021
Analysis and clarification of the taken notes during the meeting	2 days	21/09/2021	22/09/2021
◢ Scope definition	**1 day**	**23/09/2021**	**23/09/2021**
Meeting with the tutor to talk about the scope of the system	0,5 days	23/09/2021	23/09/2021
Analysis of the meeting conclusions	0,5 days	23/09/2021	23/09/2021
System Requirements identification	3 days	24/09/2021	28/09/2021

Figure 6.1: Exam Scheduler - Gantt Chart (1/7)



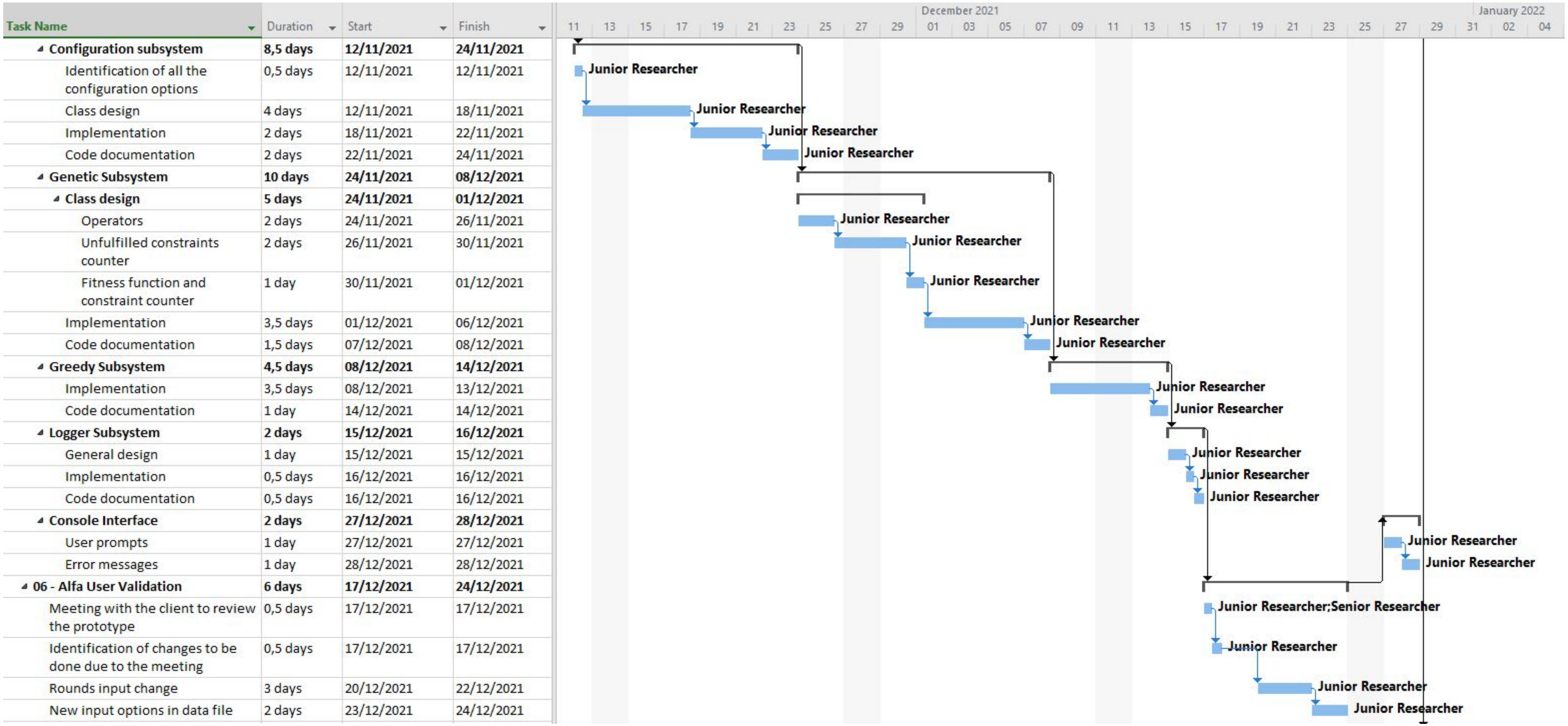
Task Name	Duration	Start	Finish
◢ **Exam Scheduler Project**	**173 days**	**27/08/2021**	**27/04/2022**
◢ **01 - Project Planning**	**5 days**	**27/08/2021**	**02/09/2021**
First meeting with the Tutor	1 day	27/08/2021	27/08/2021
Definition of project phases	2 days	30/08/2021	31/08/2021
Design of the documentation	2 days	01/09/2021	02/09/2021
◢ **02 - Initial research**	**11 days**	**03/09/2021**	**17/09/2021**
Gonzalo De La Cruz project review	2 days	03/09/2021	06/09/2021
Intelligent Systems Metaheuristics materials review	2 days	07/09/2021	08/09/2021
◢ **Literature review**	**7 days**	**09/09/2021**	**17/09/2021**
Metaheuristics	1 day	09/09/2021	09/09/2021
Evolutionary algorithms and Genetic Algorithms	2 days	10/09/2021	13/09/2021
Scheduling and Timetabling problems	2 days	14/09/2021	15/09/2021
University Exam Timetabling problems	2 days	16/09/2021	17/09/2021
◢ **03 - Problem Analysis**	**7 days**	**20/09/2021**	**28/09/2021**
◢ **Requirements Adquisition**	**3 days**	**20/09/2021**	**22/09/2021**
Meeting with the tutor and the client	1 day	20/09/2021	20/09/2021
Analysis and clarification of the taken notes during the meeting	2 days	21/09/2021	22/09/2021
◢ **Scope definition**	**1 day**	**23/09/2021**	**23/09/2021**
Meeting with the tutor to talk about the scope of the system	0,5 days	23/09/2021	23/09/2021
Analysis of the meeting conclusions	0,5 days	23/09/2021	23/09/2021
System Requirements identification	3 days	24/09/2021	28/09/2021

Figure 6.2: Exam Scheduler - Gantt Chart (2/8)



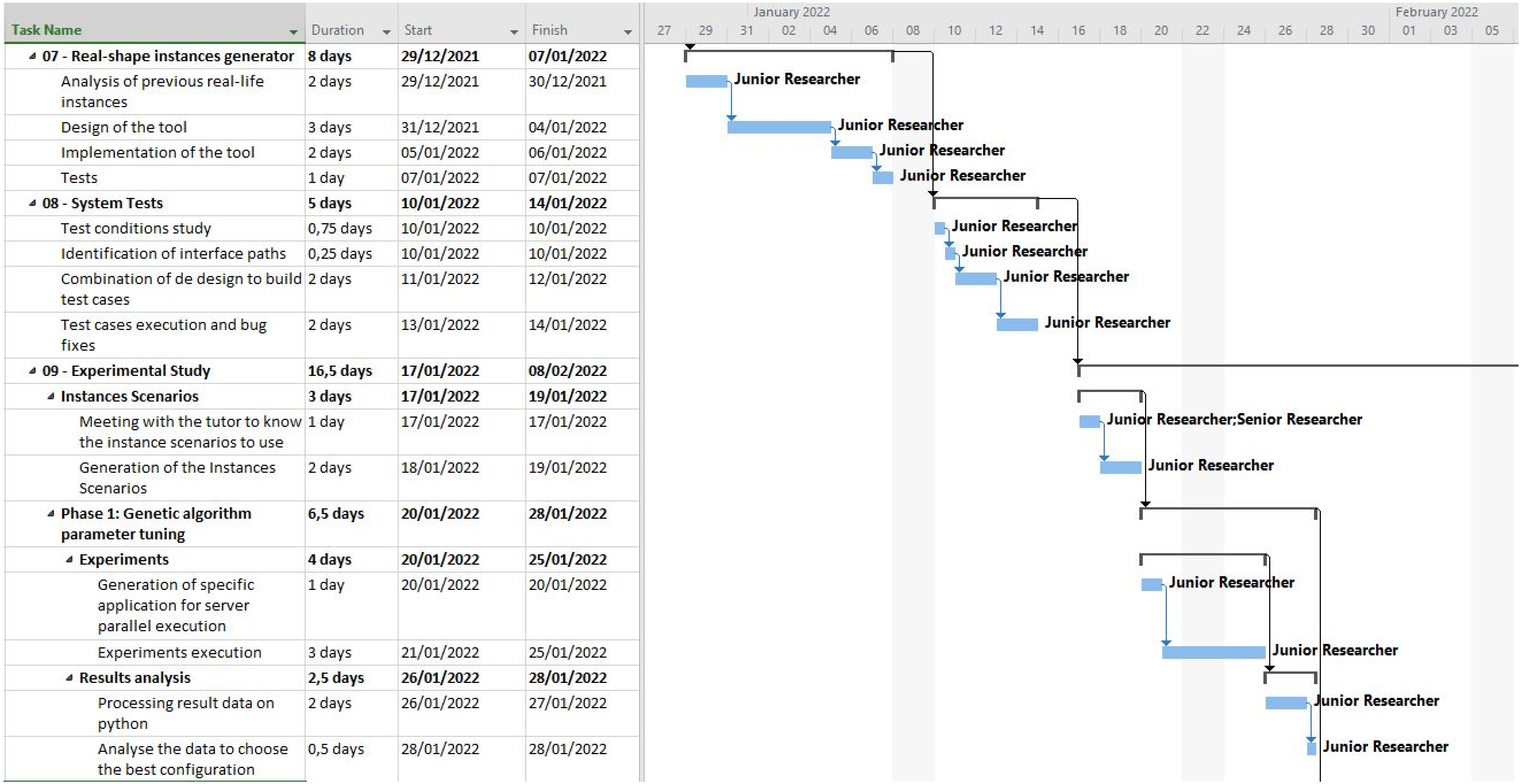
Task Name	Duration	Start	Finish
Exam Scheduler Project	173 days	27/08/2021	27/04/2022
01 - Project Planning	5 days	27/08/2021	02/09/2021
02 - Initial research	11 days	03/09/2021	17/09/2021
03 - Problem Analysis	7 days	20/09/2021	28/09/2021
04 - Proposed Solution	10 days	29/09/2021	12/10/2021
Problem formaliaztion	2 days	29/09/2021	30/09/2021
Genetic Algorithm design	1 day	01/10/2021	01/10/2021
Greedy Algorithm design	3 days	04/10/2021	06/10/2021
Repairing algorithm design	2 days	07/10/2021	08/10/2021
Fitness function design	1 day	11/10/2021	11/10/2021
Numerical Complexity function design	1 day	12/10/2021	12/10/2021
05 - Solution Implementation	55 days	13/10/2021	28/12/2021
General system architecture definition	2 days	13/10/2021	14/10/2021
Domain subsystem	20 days	15/10/2021	11/11/2021
Class design	6 days	15/10/2021	22/10/2021
General structure	1 day	15/10/2021	15/10/2021
Constraints hierarchy	2 days	18/10/2021	19/10/2021
Parser structucture	3 days	20/10/2021	22/10/2021
Implementation	11 days	25/10/2021	08/11/2021
Domain entities	3 days	25/10/2021	27/10/2021
Parsers	8 days	28/10/2021	08/11/2021
Exam parser	4 days	28/10/2021	02/11/2021
Main logic	2 days	28/10/2021	29/10/2021
Input validation for user feedback	2 days	01/11/2021	02/11/2021
Constraint Parser	4 days	03/11/2021	08/11/2021
Main logic	2 days	03/11/2021	04/11/2021
Input validation for user feedback	2 days	05/11/2021	08/11/2021
Code documentation	3 days	09/11/2021	11/11/2021

Figure 6.3: Exam Scheduler - Gantt Chart (3/8)



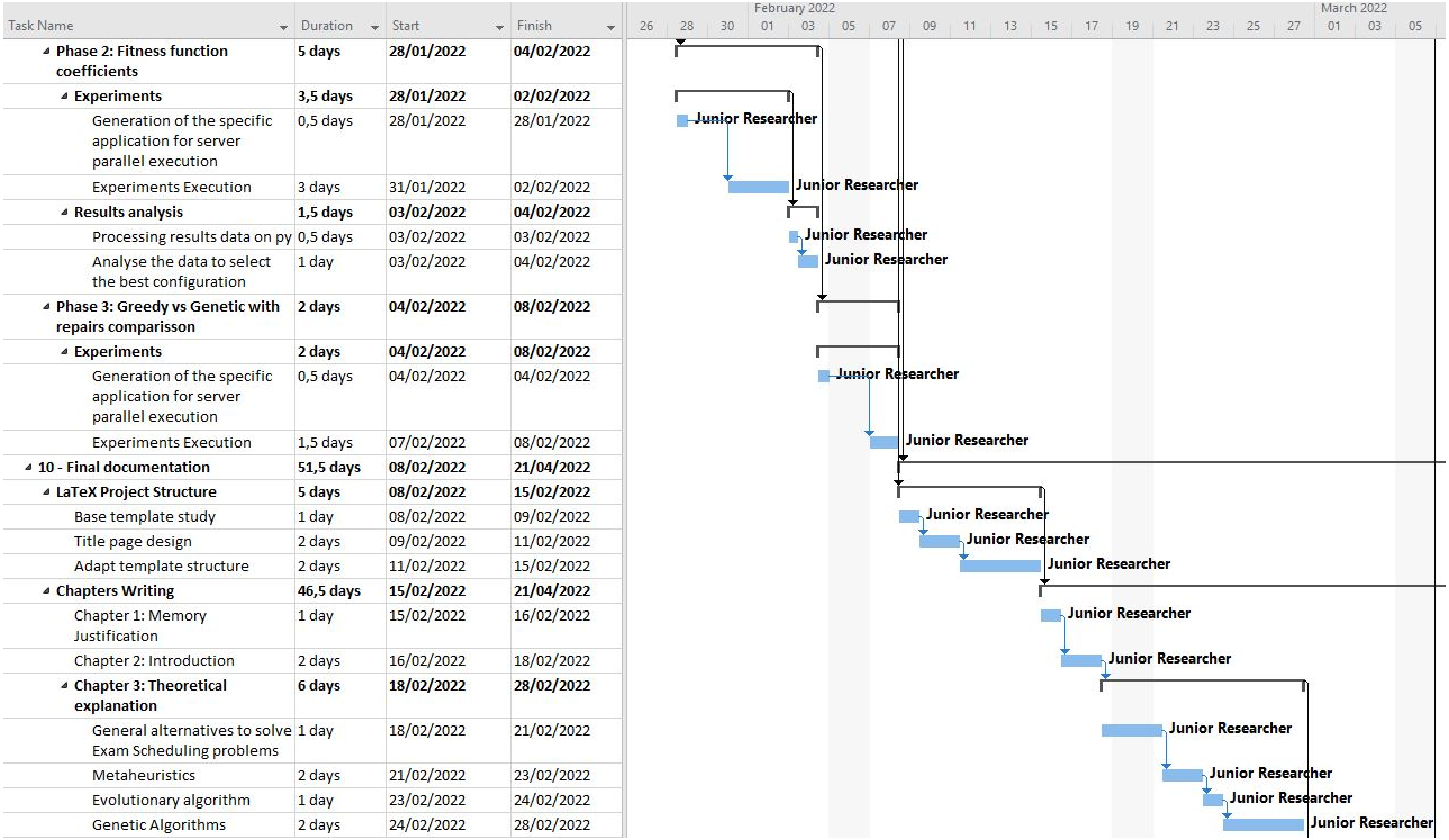
Task Name	Duration	Start	Finish
Configuration subsystem	**8,5 days**	**12/11/2021**	**24/11/2021**
Identification of all the configuration options	0,5 days	12/11/2021	12/11/2021
Class design	4 days	12/11/2021	18/11/2021
Implementation	2 days	18/11/2021	22/11/2021
Code documentation	2 days	22/11/2021	24/11/2021
Genetic Subsystem	**10 days**	**24/11/2021**	**08/12/2021**
Class design	**5 days**	**24/11/2021**	**01/12/2021**
Operators	2 days	24/11/2021	26/11/2021
Unfulfilled constraints counter	2 days	26/11/2021	30/11/2021
Fitness function and constraint counter	1 day	30/11/2021	01/12/2021
Implementation	3,5 days	01/12/2021	06/12/2021
Code documentation	1,5 days	07/12/2021	08/12/2021
Greedy Subsystem	**4,5 days**	**08/12/2021**	**14/12/2021**
Implementation	3,5 days	08/12/2021	13/12/2021
Code documentation	1 day	14/12/2021	14/12/2021
Logger Subsystem	**2 days**	**15/12/2021**	**16/12/2021**
General design	1 day	15/12/2021	15/12/2021
Implementation	0,5 days	16/12/2021	16/12/2021
Code documentation	0,5 days	16/12/2021	16/12/2021
Console Interface	**2 days**	**27/12/2021**	**28/12/2021**
User prompts	1 day	27/12/2021	27/12/2021
Error messages	1 day	28/12/2021	28/12/2021
06 - Alfa User Validation	**6 days**	**17/12/2021**	**24/12/2021**
Meeting with the client to review the prototype	0,5 days	17/12/2021	17/12/2021
Identification of changes to be done due to the meeting	0,5 days	17/12/2021	17/12/2021
Rounds input change	3 days	20/12/2021	22/12/2021
New input options in data file	2 days	23/12/2021	24/12/2021

Figure 6.4: Exam Scheduler - Gantt Chart (4/8)



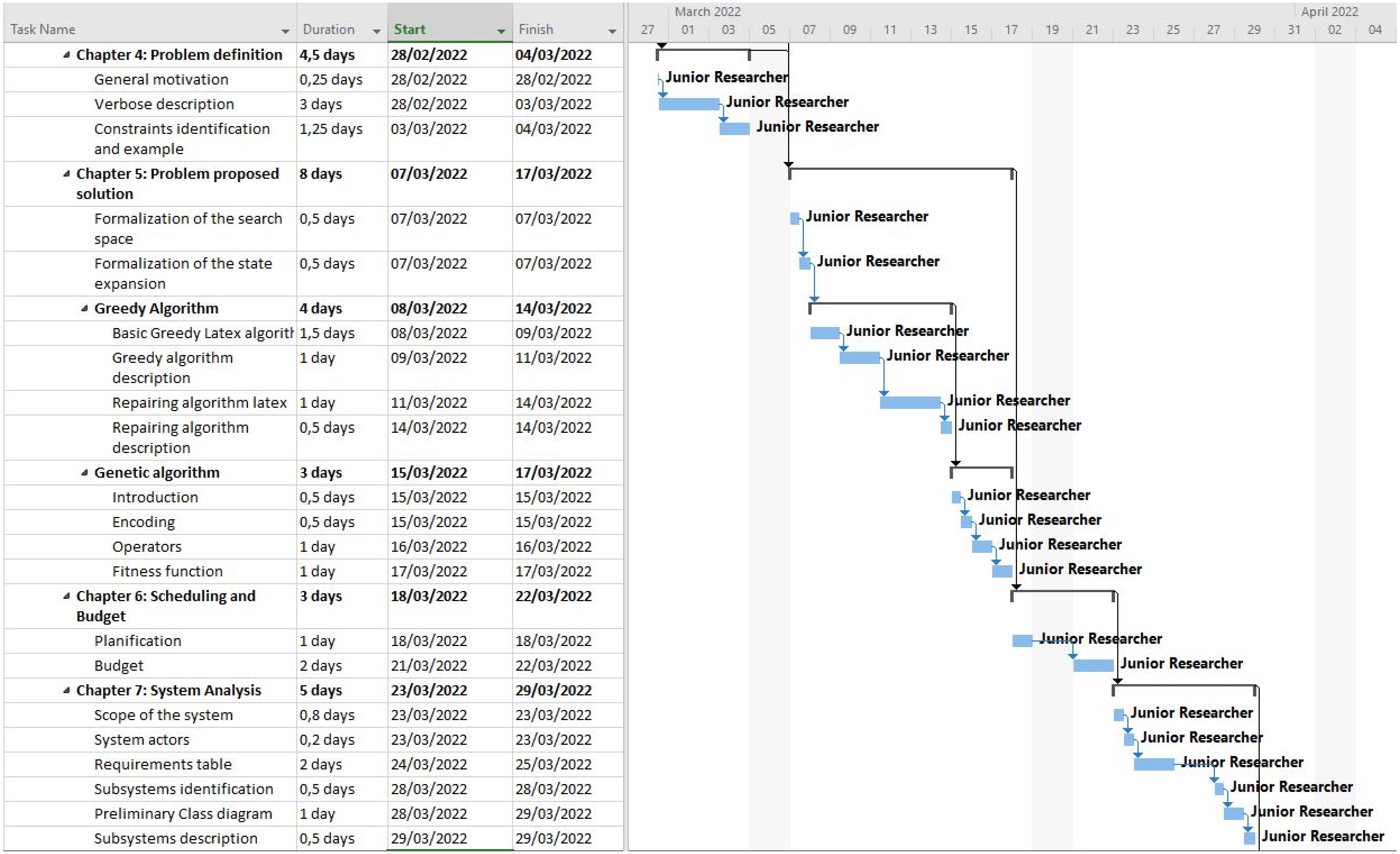
Task Name	Duration	Start	Finish
◢ **07 - Real-shape instances generator**	**8 days**	**29/12/2021**	**07/01/2022**
Analysis of previous real-life instances	2 days	29/12/2021	30/12/2021
Design of the tool	3 days	31/12/2021	04/01/2022
Implementation of the tool	2 days	05/01/2022	06/01/2022
Tests	1 day	07/01/2022	07/01/2022
◢ **08 - System Tests**	**5 days**	**10/01/2022**	**14/01/2022**
Test conditions study	0,75 days	10/01/2022	10/01/2022
Identification of interface paths	0,25 days	10/01/2022	10/01/2022
Combination of de design to build test cases	2 days	11/01/2022	12/01/2022
Test cases execution and bug fixes	2 days	13/01/2022	14/01/2022
◢ **09 - Experimental Study**	**16,5 days**	**17/01/2022**	**08/02/2022**
◢ **Instances Scenarios**	**3 days**	**17/01/2022**	**19/01/2022**
Meeting with the tutor to know the instance scenarios to use	1 day	17/01/2022	17/01/2022
Generation of the Instances Scenarios	2 days	18/01/2022	19/01/2022
◢ **Phase 1: Genetic algorithm parameter tuning**	**6,5 days**	**20/01/2022**	**28/01/2022**
◢ **Experiments**	**4 days**	**20/01/2022**	**25/01/2022**
Generation of specific application for server parallel execution	1 day	20/01/2022	20/01/2022
Experiments execution	3 days	21/01/2022	25/01/2022
◢ **Results analysis**	**2,5 days**	**26/01/2022**	**28/01/2022**
Processing result data on python	2 days	26/01/2022	27/01/2022
Analyse the data to choose the best configuration	0,5 days	28/01/2022	28/01/2022

Figure 6.5: Exam Scheduler - Gantt Chart (5/8)



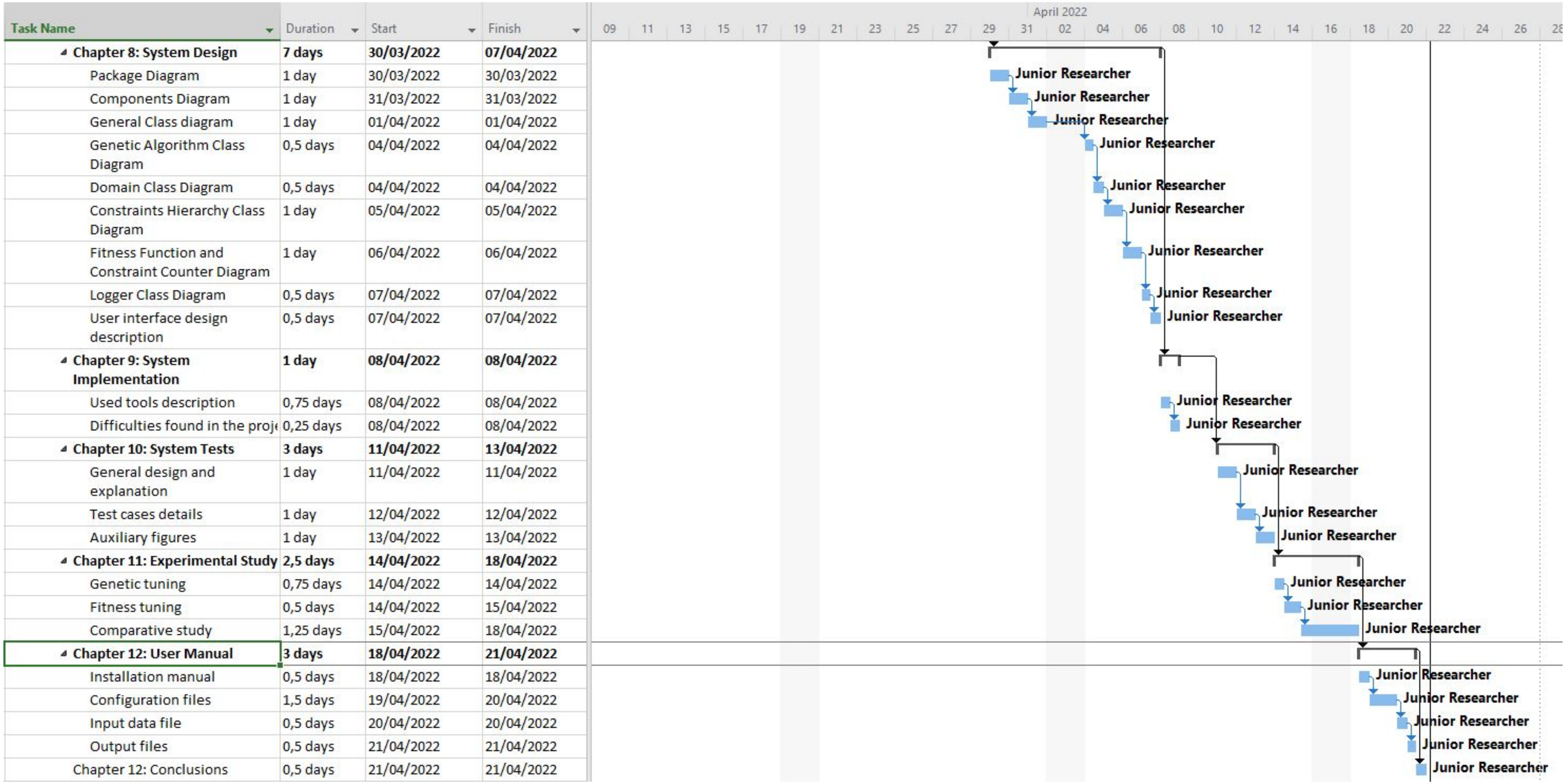
Task Name	Duration	Start	Finish
Phase 2: Fitness function coefficients	**5 days**	**28/01/2022**	**04/02/2022**
Experiments	**3,5 days**	**28/01/2022**	**02/02/2022**
Generation of the specific application for server parallel execution	0,5 days	28/01/2022	28/01/2022
Experiments Execution	3 days	31/01/2022	02/02/2022
Results analysis	**1,5 days**	**03/02/2022**	**04/02/2022**
Processing results data on py	0,5 days	03/02/2022	03/02/2022
Analyse the data to select the best configuration	1 day	03/02/2022	04/02/2022
Phase 3: Greedy vs Genetic with repairs comparisson	**2 days**	**04/02/2022**	**08/02/2022**
Experiments	**2 days**	**04/02/2022**	**08/02/2022**
Generation of the specific application for server parallel execution	0,5 days	04/02/2022	04/02/2022
Experiments Execution	1,5 days	07/02/2022	08/02/2022
10 - Final documentation	**51,5 days**	**08/02/2022**	**21/04/2022**
LaTeX Project Structure	**5 days**	**08/02/2022**	**15/02/2022**
Base template study	1 day	08/02/2022	09/02/2022
Title page design	2 days	09/02/2022	11/02/2022
Adapt template structure	2 days	11/02/2022	15/02/2022
Chapters Writing	**46,5 days**	**15/02/2022**	**21/04/2022**
Chapter 1: Memory Justification	1 day	15/02/2022	16/02/2022
Chapter 2: Introduction	2 days	16/02/2022	18/02/2022
Chapter 3: Theoretical explanation	**6 days**	**18/02/2022**	**28/02/2022**
General alternatives to solve Exam Scheduling problems	1 day	18/02/2022	21/02/2022
Metaheuristics	2 days	21/02/2022	23/02/2022
Evolutionary algorithm	1 day	23/02/2022	24/02/2022
Genetic Algorithms	2 days	24/02/2022	28/02/2022

Figure 6.6: Exam Scheduler - Gantt Chart (6/8)



Task Name	Duration	Start	Finish
Chapter 4: Problem definition	**4,5 days**	**28/02/2022**	**04/03/2022**
General motivation	0,25 days	28/02/2022	28/02/2022
Verbose description	3 days	28/02/2022	03/03/2022
Constraints identification and example	1,25 days	03/03/2022	04/03/2022
Chapter 5: Problem proposed solution	**8 days**	**07/03/2022**	**17/03/2022**
Formalization of the search space	0,5 days	07/03/2022	07/03/2022
Formalization of the state expansion	0,5 days	07/03/2022	07/03/2022
Greedy Algorithm	**4 days**	**08/03/2022**	**14/03/2022**
Basic Greedy Latex algorit	1,5 days	08/03/2022	09/03/2022
Greedy algorithm description	1 day	09/03/2022	11/03/2022
Repairing algorithm latex	1 day	11/03/2022	14/03/2022
Repairing algorithm description	0,5 days	14/03/2022	14/03/2022
Genetic algorithm	**3 days**	**15/03/2022**	**17/03/2022**
Introduction	0,5 days	15/03/2022	15/03/2022
Encoding	0,5 days	15/03/2022	15/03/2022
Operators	1 day	16/03/2022	16/03/2022
Fitness function	1 day	17/03/2022	17/03/2022
Chapter 6: Scheduling and Budget	**3 days**	**18/03/2022**	**22/03/2022**
Planification	1 day	18/03/2022	18/03/2022
Budget	2 days	21/03/2022	22/03/2022
Chapter 7: System Analysis	**5 days**	**23/03/2022**	**29/03/2022**
Scope of the system	0,8 days	23/03/2022	23/03/2022
System actors	0,2 days	23/03/2022	23/03/2022
Requirements table	2 days	24/03/2022	25/03/2022
Subsystems identification	0,5 days	28/03/2022	28/03/2022
Preliminary Class diagram	1 day	28/03/2022	29/03/2022
Subsystems description	0,5 days	29/03/2022	29/03/2022

Figure 6.7: Exam Scheduler - Gantt Chart (7/8)



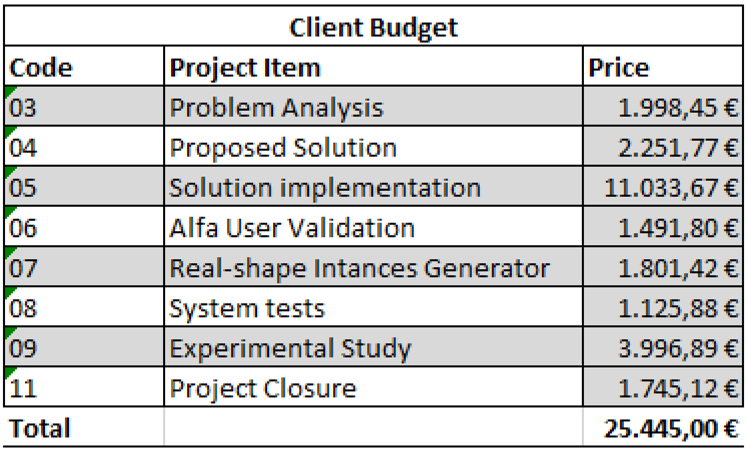
Task Name	Duration	Start	Finish
◢ **Chapter 8: System Design**	**7 days**	**30/03/2022**	**07/04/2022**
Package Diagram	1 day	30/03/2022	30/03/2022
Components Diagram	1 day	31/03/2022	31/03/2022
General Class diagram	1 day	01/04/2022	01/04/2022
Genetic Algorithm Class Diagram	0,5 days	04/04/2022	04/04/2022
Domain Class Diagram	0,5 days	04/04/2022	04/04/2022
Constraints Hierarchy Class Diagram	1 day	05/04/2022	05/04/2022
Fitness Function and Constraint Counter Diagram	1 day	06/04/2022	06/04/2022
Logger Class Diagram	0,5 days	07/04/2022	07/04/2022
User interface design description	0,5 days	07/04/2022	07/04/2022
◢ **Chapter 9: System Implementation**	**1 day**	**08/04/2022**	**08/04/2022**
Used tools description	0,75 days	08/04/2022	08/04/2022
Difficulties found in the proj	0,25 days	08/04/2022	08/04/2022
◢ **Chapter 10: System Tests**	**3 days**	**11/04/2022**	**13/04/2022**
General design and explanation	1 day	11/04/2022	11/04/2022
Test cases details	1 day	12/04/2022	12/04/2022
Auxiliary figures	1 day	13/04/2022	13/04/2022
◢ **Chapter 11: Experimental Study**	**2,5 days**	**14/04/2022**	**18/04/2022**
Genetic tuning	0,75 days	14/04/2022	14/04/2022
Fitness tuning	0,5 days	14/04/2022	15/04/2022
Comparative study	1,25 days	15/04/2022	18/04/2022
◢ **Chapter 12: User Manual**	**3 days**	**18/04/2022**	**21/04/2022**
Installation manual	0,5 days	18/04/2022	18/04/2022
Configuration files	1,5 days	19/04/2022	20/04/2022
Input data file	0,5 days	20/04/2022	20/04/2022
Output files	0,5 days	21/04/2022	21/04/2022
Chapter 12: Conclusions	0,5 days	21/04/2022	21/04/2022

Figure 6.8: Exam Scheduler - Gantt Chart (8/8)

6.2 Budget

In this section the client budget is presented based on the planning previously exposed. As can be seen in Figure 6.9 the project has been divided in eight items.

For further reference on how this client budget was done check Chapter 13.



Client Budget		
Code	Project Item	Price
03	Problem Analysis	1.998,45 €
04	Proposed Solution	2.251,77 €
05	Solution implementation	11.033,67 €
06	Alfa User Validation	1.491,80 €
07	Real-shape Intances Generator	1.801,42 €
08	System tests	1.125,88 €
09	Experimental Study	3.996,89 €
11	Project Closure	1.745,12 €
Total		**25.445,00 €**

Figure 6.9: Client Budget

Chapter 7

Analysis

In this chapter the prototype to be built will be analyzed and modelled. The scope of the system will be initially provided, followed by a detailed view of the subsystems by means of the requirements table, as well as use cases and scenarios.

7.1 System Specification

7.1.1 Scope of the system

The functional goal of the project is to create a prototype implementing the algorithm detailed in Chapter 5, which is our proposed solution for solving the exam scheduling problem of the School of Computer Science.

The prototype must be a command line application which receives the inputs by means of files, being such the necessary input data of the problem along with some other configurations for the algorithm.

Once that information is provided, the system will start computing the best solution until completing a certain number of iterations specified by the user. For the sake of feedback, during that computing process the algorithm must be informing the user about the actual best and average fitness, the expected time to end, and the completion percentage.

After finishing the computation, the system must report an specified number of different solutions, being them the best ones. That output will be done in the same format as it is in the input, so that it is possible for the user to always work with the same file format.

7.2 System Requirements

7.2.1 Requirements analysis

In this section the requirements of the system will be detailed. There will be a differentiation between functional and non-functional requirements. Each of those categories will have groups of requisites for the different functionalities of the prototype. For the sake of clarity, the sub-requisites will be tabbed with respect to their parents.

Now the functional requirements will be listed. Their identifiers meet the following legend:

- **IDR:** Input Data Requirements.
- **ICR:** Input Configuration Requirements.
- **IEVR:** Input Error Validation Requirements.
- **ER:** Execution Requirements.
- **OFR:** Output Files Requirements.

IDR1 The user must provide the exams to be handled by the algorithm.

IDR1.1 The user must specify the exams to be scheduled providing:

IDR1.1.1 Course.
IDR1.1.2 Semester.
IDR1.1.3 Code.
IDR1.1.4 Acronym.
IDR1.1.5 Subject.
IDR1.1.6 Order.
IDR1.1.7 Content type.
IDR1.1.8 Modality.
IDR1.1.9 Number of students.
IDR1.1.10 Duration.
IDR1.1.11 Unique numeric identifier.

IDR1.2 The user can specify already scheduled exams providing everything stated in **IDR1.1** and also:

IDR1.2.1 Date of the exam.
IDR1.2.2 Day of the week of the exam.
IDR1.2.3 Hour to start the exam.
IDR1.2.4 Hour to end the exam.

IDR2 The user may provide to any exam stated on **IDR1** any of the following extra data:

IDR2.1 Extra time.

IDR2.2 Numerical complexity.

IDR2.3 Round identifier.

IDR3 The user must be able to input the constraints to the system.

IDR3.1 The user must be able to specify user constraints.

IDR3.1.1 The user must be able to specify Day Banned Constraints (DB) providing:
IDR3.1.1.1 The exam id affected by the constraint.
IDR3.1.1.2 The day in which the exam cannot be placed.

IDR3.1.2 The user must be able to specify Time Displacement Constraints (TD) providing:
IDR3.1.2.1 The exam id of the first exam.
IDR3.1.2.2 The exam id of the second exam.
IDR3.1.2.3 The distance in calendar day positions that the second exam must have with respect to the first one.

IDR3.1.3 The user must be able to specify Order Exam Constraints (OE) providing:

IDR3.1.3.1 The exam id of the first exam.

IDR3.1.3.2 The exam id of the second exam.

IDR3.1.4 The user must be able to specify Same Day Constraints (SD) providing two exam ids.

IDR3.1.5 The user must be able to specify Different Day Constraints (DD) providing two exam ids.

IDR3.1.6 The user must be able to specify Day Interval Constraints (DI) providing:

IDR3.1.6.1 The exam id of the exam affected by the constraint.

IDR3.1.6.2 The time interval, bounded by two dates, in which the exam can be placed

IDR3.2 All user constraints can be marked as "hard".

IDR4 The user must be able to specify the calendar of days in which exams can be placed.

IDR4.1 Each day must have assigned an interval in which exams can be placed.

IDR5 All the input data of **IDR1.1**, **IDR1.2**, **IDR2**, **4.3.3** and **IDR4** must be specified in a single Excel file.

ICR1 The system must allow the user to configure the genetic algorithm parameters.

ICR1.1 Size of the population.

ICR1.2 Number of generations to be performed.

ICR1.3 Mutation probability.

ICR1.4 Crossover probability.

ICR1.5 Repairing algorithm recursive tree of changes depth.

ICR1.6 Number of schedules to be reported.

ICR1.7 Number of times that the algorithm must be run.

ICR2 The system must allow the user to set scheduling time configurations.

ICR2.1 The user must be able to set the default extra time for the exams.

ICR2.2 The user must be able to enable or disable the default extra time functionality.

ICR2.2.1 If enabled, then **ICR2.1** must be set as extra time to all exams declared on **IDR1** which do not have specified an extra time by means of **IDR2**.

ICR2.3 The user must be able to set the resting interval.

ICR2.4 The user must be able to specify a modality value, by which the application will identify if the exam is a delivery..

ICR2.5 The user must be able to disable exam collisions for exams which are identified as deliveries by means of **ICR2.4**.

ICR2.5.1 If disabled exam collisions for an exam, that exam can be placed at the same time as other exams.

ICR3 The user must be able to change the weights of the fitness function.

ICR3.1 The user must be able to change the weight of The user constraints individually.

ICR3.2 The user must be able to change the weight of the resting time penalization constraint.

ICR3.3 The user must be able to change the weight of the numerical complexity penalization.

ICR3.4 The user must be able to change the weight of the number of unclassified exams.

ICR4 Any error detected during parsing must be notified to the user.

ICR5 An error report will be generated when the parsing is complete.

ICR5.1 The report must inform of input parsing troubleshooting.

ICR5.1.1 Exams skipped due to bad parsing.
ICR5.1.2 Constraints skipped due to bad parsing.
ICR5.1.3 Calendar days skipped due to bad parsing.
ICR5.1.4 Configurations options

ICR5.2 If not empty, the user will be notified.

ICR5.2.1 The execution of the application will be stopped.
ICR5.2.2 The user will be asked if execution must proceed.
ICR5.2.2.1 On a yes, the application will proceed.
ICR5.2.2.2 On a no, the application will end.

IEVR1 Any error parsing any piece of data on **IDRx** will imply to skip that piece data.

IEVR2 Any error parsing any piece of data on **ICRx** will imply to assign it to a default value of **CONF_DEFAULT_VALUE**.

IEVR3 An error report will be generated when the input parsing is complete.

IEVR3.1 The report must inform of input parsing troubleshooting.

IEVR3.1.1 Exams skipped due to bad parsing.
IEVR3.1.2 Constraints skipped due to bad parsing.
IEVR3.1.3 Calendar days skipped due to bad parsing.
IEVR3.1.4 Configurations options set to default value.

IEVR3.2 If not empty, the user will be notified.

IEVR3.2.1 The application execution will be stopped.
IEVR3.2.2 The user will be asked if execution must proceed.
IEVR3.2.2.1 On a yes, the application will proceed.
IEVR3.2.2.2 On a no, the application will end.

IEVR3.3 If the input file formats are not the ones specified in Chapter **12**, the application execution will be aborted.

IEVR3.3.1 User will be notified in the console of the error.

ER1 During the execution the system must inform about the current status in each generation.

ER1.1 The best fitness must be shown.

ER1.2 The average fitness must be shown.

ER1.3 The current generation.

ER1.4 The maximum number of generations.

ER1.5 The percentage of completion.

ER1.6 The time that the algorithm has been being executed.

ER1.7 The expected time in which the algorithm will end.

OFR1 When the algorithm ends its execution, a maximum number of different schedules (specified on **ICR1.6**) will be generated.

OFR2 The output schedule files must be in the same format as the input file **IDR5**.

OFR2.1 It must be possible to use an output file directly as an input file.

OFR3 The exams which were initially unscheduled and were successfully scheduled by the algorithm must have specified data on **IDR1.2**.

OFR4 The output schedule files must detail which constraints have been fulfilled.

Now non-functional requirements will be listed. Their identifiers meet the following legend:

- **TR:** Technological Requirements.
- **PR:** Performance Requirements.

TR1 The system must be a command-line application.

TR2 The system must be configured by means of files.

TR3 The system must not require more than **PARAM_NUMBER_LIMIT** arguments on the running script.

PR1 The algorithm must compute a solution in less than **EXEC_TIME_LIMIT**.

Since there was a need to use variables on the requirements, here are the assigned values for them:

Variable name	Value
CONF_DEFAULT_VALUE	0
PARAM_NUMBER_LIMIT	6
EXEC_TIME_LIMIT	6 hours

Table 7.1: Requirements variables

7.2.2 System actors

Due to the simplicity of the system, it is not connected to any other system. Therefore, there is only one user, which is the person currently in charge of making the exam schedule in the School of Computer Science.

After the system report a group of solutions as a result, it will be that user the one that will select the final schedule to be used.

7.3 Subsystems Identification and Description

In this section the subsystems that compose the prototype will be described.

- **Domain:** This subsystem holds all the domain logic. It will be in charge of handling the exam instances, of providing logic to check for collisions, etc. One of the main functionalities will be parsing the input files to domain instances that can be handled by the other subsystems.
- **Genetic Algorithm:** This subsystem covers all the Genetic Algorithm functionality, including the operators, previously commented in Chapter 5. It is composed of other subsystems:
  - **Encoder Subsystem:** This subsystem is the one that encodes the candidate solutions to chromosomes, and so creates the individuals to be used by the Genetic Algorithm.
  - **Selection Operators:** This subsystem contains all the selection operators that were evaluated in the algorithm.
  - **Crossover Operators:** This subsystem contains all the crossover operators that were evaluated in the algorithm.
  - **Mutation Operators:** This subsystem contains all the mutation operators that were evaluated in the algorithm.
  - **Replacement Operators:** This subsystem contains all the replacement operators that were evaluated in the algorithm.
  - **Fitness Subsystem:** This subsystem contains all the fitness functions that are used or were tested in the Genetic Algorithm.
- **Greedy Algorithm:** This subsystem contains the logic of the Greedy Algorithm.
- **Logger:** This subsystem generates the logging files both at the beginning and end of the algorithm's execution.

There is also a special subsystem which was used to build the testing environment for the algorithm:

- **Test Samples Generator:** This is an extra subsystem which was used to generate test instances for the problem. It depends on the domain subsystem to do so.

7.4 Preliminary Classes Identification

This section is devoted to expose and explain the identified classes during the analysis phase. They intend to model an approximate idea of how the final system will look like.

In this section a diagram will be shown and the entities appearing in such a diagram will be described.

7.4.1 Preliminary class diagram

A diagram modelling the relations amongst the preliminary classes can be seen in Figure 7.1.

In such diagram, the classes are grouped in the briefly described subsystems in Section 7.3.

7.4.2 Classes description

In this section the core classes of each subsystem are described in more detail. This is an initial analysis, and so, what is detailed here, even if similar to the final design, it is not exactly equal.

Domain Subsystem

ExamSchedule

Description

This is the core class of the Domain subsystem. It is the one that holds the exam instances, the constraints, and logic to check for collisions.

Responsibilities

This class will act as a façade for all the domain subsystem. It will be in charge of holding the current list of exams, as well as the constraints. It will provide logic to schedule and unschedule exams, and check for collisions.

Proposed attributes

- **Exams**: The list of exams that were parsed from the input files, and over which the scheduled will be made.
- **Constraints**: The list of soft constraints that need to be considered.

Proposed methods

- **ScheduleExam**: Schedules a given exam on a given date and time.
- **UnscheduleExam**: Unschedules a given exam.
- **GetSwappableExamsOver**: Takes an exam and a list of days in which it can be placed and returns all the exams that have a time gap assigned in which the provided exam can fit. This method will be mainly used by the repair algorithm of the Genetic Algorithm.
- **VerifyConstraints**: Checks the fulfillment of the constraints.

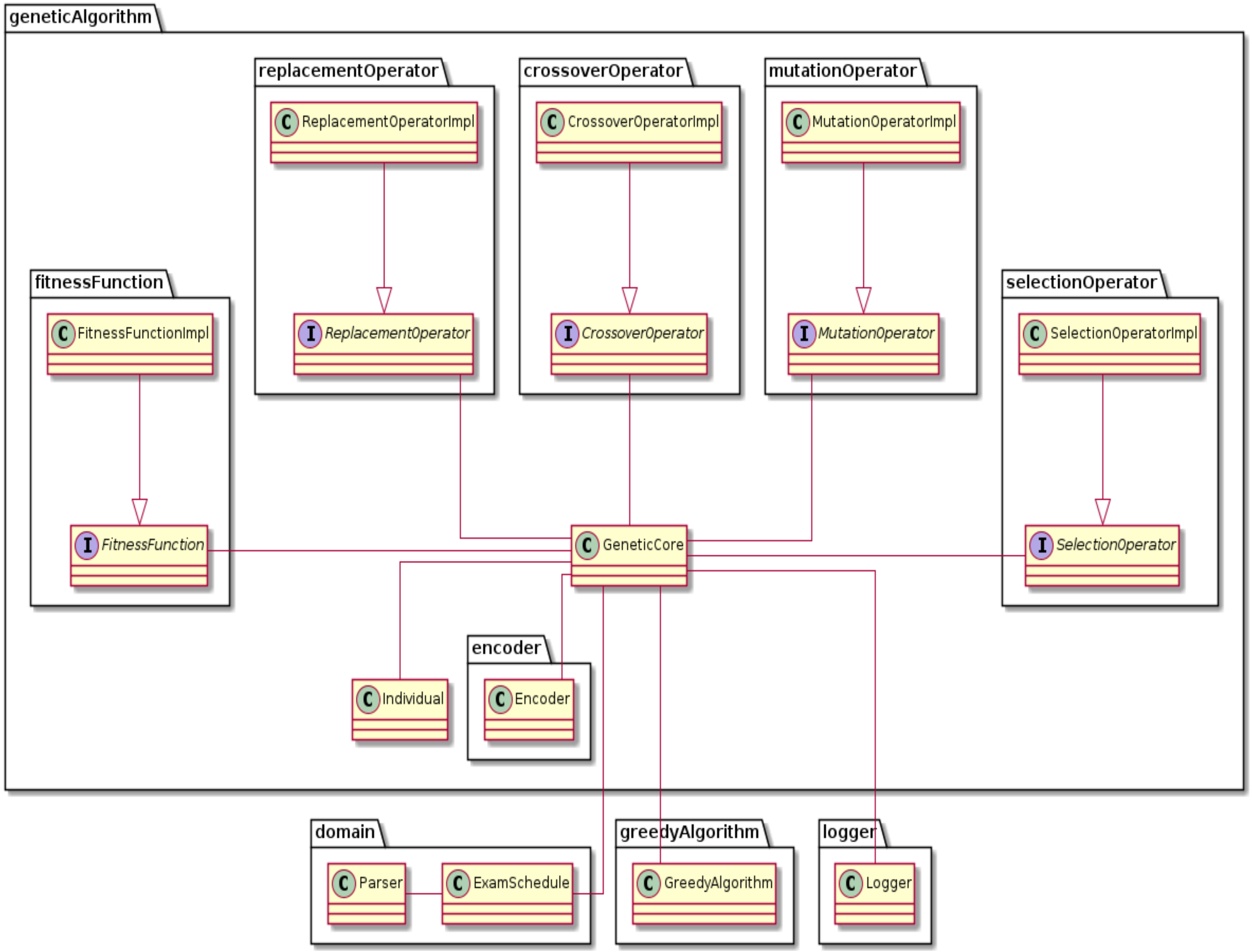


Figure 7.1: Preliminary class diagram.

Parser

DESCRIPTION

This will not be a concrete class itself. Particular parsers will be used for each input (exams, constraints, days). This “class” is a representation of the functionalities of all of them.

RESPONSIBILITIES

Parsers will be in charge of both translating the input into domain objects that can be handled by the algorithm and to translate these objects to the output files maintaining the format of the input files.

PROPOSED ATTRIBUTES

-

PROPOSED METHODS

There will be many specific methods since we will be parsing an Excel file, however the core functionalities are the following:

- **ParseExams**: Parses the list of exams from the Excel file.
- **ParseConstraints**: Parses the list of user constraints from the Excel file.
- **ParseDates**: Parses the list of dates from the Excel file.
- **ParseIndividualToExcel**: Takes an individual of the Genetic Algorithm population, transforms it into a solution by means of the Greedy Algorithm, and the writes the whole schedule (exams, constraints and calendar) into Excel format.

Greedy Algorithm Subsystem

Greedy Algorithm

DESCRIPTION

This is in charge of guiding the search process towards valid solutions according to the chromosome sequence in the individuals handled by the Genetic Algorithm.

RESPONSIBILITIES

This class must schedule the exams taking as seed the chromosome sequence of the individuals handled by the Genetic Algorithm. It must also compute some quality factors, such as the number of unscheduled exams, for the fitness function to evaluate them.

PROPOSED ATTRIBUTES

- **LimitDepth**: The Greedy Algorithm will have implemented a repair algorithm to increase its efficiency. This algorithm will come into place when an exam cannot be placed due to hard constraints or time issues. What it will try to achieve is to swap this exam with another placed one that would still be able to be placed again in other place. This attribute would limit the depth of the tree of swaps of such algorithm.

PROPOSED METHODS

- **Decode**: This method will take as seed the chromosome of the individuals of the Genetic Algorithm and reach deterministically a candidate solution of the problem.

Genetic Algorithm Subsystem

Genetic Algorithm

DESCRIPTION

This class is in charge of guiding the search to a promising area of the search space using a Genetic Algorithm. In this case, of getting fitter individuals along the generations to be used as seeds by the Greedy Algorithm.

RESPONSIBILITIES

This class will hold the current population and make it evolve by means of evolutionary algorithm techniques. It will call all the evolutionary operators and also inform the logger about how the fitness is changing along the iterations.

PROPOSED ATTRIBUTES

- **Population**: List of individuals that the Genetic Algorithm is trying to evolve.
- **SelectionOperator**: Selection operator to be used by the algorithm.
- **MutationOperator**: Mutation operator to be used by the algorithm.
- **CrossoverOperator**: Crossover operator to be used by the algorithm.
- **ReplacementOperator**: Replacement operator to be used by the algorithm.
- **Logger**: Link to the logger subsystem which needs to be informed about the status evolution of the population.
- **Encoder**: Encoder to be used during the execution of the algorithm.
- **GreedyAlgorithm**: Genetic Algorithm that will process the individuals to reach deterministically candidate solutions of the problem. It can be considered in this context the decoder of the Genetic Algorithm.
- **SelectionOperator**: Selection operator to be used by the algorithm.
- **FitnessFunction**: Fitness function selected, that will be used to compute the fitness value of the individuals.

PROPOSED METHODS

- **GeneticAlgorithm**: This methods hold the whole evolution process and will be parameterized with the input configurations.

Selection Operator Subsystem

Selection Operator
Description
This operator is in charge of selecting an individual among the ones in the population received as parameter. To do so, the fitness function is taken into account.
Responsibilities
Perform the selection of individuals from the population.
Proposed attributes
-
Proposed methods

- **Selection(population, fitnessFunction)**: This method selects an individual from the population.

Mutation Operator Subsystem

Mutation Operator
Description
This operator is in charge of performing mutations over new generated individuals.
Responsibilities
Perform the mutation of new created individuals.
Proposed attributes
-
Proposed methods

- **Mutation(Individual)**: Creates a mutated version of the provided individual.

Crossover Operator Subsystem

Crossover Operator
Description
This operator is in charge of generating a new individual giving two that will act as its parents.
Responsibilities
Perform the crossover over two given individuals.
Proposed attributes
-
Proposed methods

- **Crossover(Individual1, Individual2)**: Creates a new individual by combining the chromosomes of the individuals provided as parents.

Replacement Operator Subsystem

Replacement Operator
Description
This operator is in charge of selecting the final set of individuals that will be part of the population of a new generation. It will do so taking into account the previous generations as well as the generated children by the crossover operator.
Responsibilities
Select the final set of individuals that will be part of the population of a new generation.
Proposed attributes
-
Proposed methods

- **Replacement(PrevGeneration, Childs, FitnessFunction)**: Selects the final population of individuals for a new generation.

Encoder Subsystem

Encoder

DESCRIPTION

Encodes a given list of exams into an individual that the Genetic Algorithm can use.

RESPONSIBILITIES

Generate an individual from a provided candidate solution.

PROPOSED ATTRIBUTES

-

PROPOSED METHODS

- **EncodeListOfExams(ExamSchedule)**: This method takes the list of exams in the ExamSchedule class and generates an individual. It is needed to use the ExamSchedule as an intermediary because there are already placed exams that must not be rescheduled by the algorithm and so, that must not appear encoded in the individuals of the Genetic Algorithm.

Fitness function Subsystem

Fitness Function

DESCRIPTION

The Fitness function will compute the fitness value for each individual in the Genetic Algorithm population.

RESPONSIBILITIES

Compute the fitness function for each individual in the Genetic Algorithm population.

PROPOSED ATTRIBUTES

- **GreedyAlgorithm**: Genetic Algorithm instance that will translate a given individual from the Genetic Algorithm population into a candidate solution that can be evaluated by the fitness function.

PROPOSED METHODS

- **Apply(Individual)**: Computes the fitness value associated to a given individual.

Logger Subsystem

Logger
Description
This will log all fitness evolution related data. It will also compute some statistics at the end of the execution of the algorithm. It is mainly used for testing, tuning and statistic purposes. Furthermore it will also log the errors that could happen during the input files parsing.
Responsibilities
Track the fitness evolution and compute statistics about the provided solution at the end of the algorithm's execution. Generate the error logs to inform the user about strange inputs.
Proposed attributes
-
Proposed methods

- **AddError(String error)**: Adds an error to be displayed to the user in the application stopping points. (See Sections 8.3 and 10.1.2)
- **writeFitnessGraphData**: Writes a file with the necessary data to plot a fitness graph using iterations or lapsed time.
- **writeStatisticsToFile**: Writes a file with some quality factors about the provided solution such as the number of unscheduled exams, the number of unfulfilled constraints, etc.

7.5 Use Cases and Scenarios Analysis

The prototype is a straightforward application so there is only one use case to be commented.

Prototype execution	
Preconditions	To execute the algorithms, the following preconditions must be met: • An Excel file must be provided in the specified format. It will have at least three tabs in the right format. • A set of configuration files with no missing files or properties inside them must be provided.
Postconditions	Upon the termination of the execution the system will have not modifierd any of the input files. Instead it will have generated the following files: • An number of Excel schedule files in the same format as the one provided, but with the assignments done by the algorithm. They will contain different schedules. • A file containing the errors detected during the parsing of the input files. • A log of file parsing process. Additionally, depending on the configuration, some extra files can be generated, being such: **A fitness graph file**, with the necessary data to plot a graph with the fitness evolution during the execution; and **a statistics file**, that will contain some quality values such as the number of unplaced exams, among others.
Actors	As previously stated in Section 7.2.2, there is only one user, and so one actor. This is expected to be the person currently in charge of making the exam schedule in the School of Computer Science. This actor will execute the algorithm and evaluate the results to state the final schedule.
Description	The user launches the prototype by providing a "master file" in which the filepath to all the necessary files for the execution is specified. This is file is referred as **filesFile** in the project. Those files are well-parsed and the algorithm ends its execution providing the set of output schedules.
Secondary Scenarios	The following scenarios are considered by the algorithm, and in each of them it will provide an error message and specify the origin of the error, and, if possible what that error may be: • An exam has a mandatory field omitted. • A constraint has a mandatory field omitted. • A configuration has an invalid value.
Exceptions	Execution will be aborted if: the input Excel file does not exist, a configuration file is missing, a configuration value is missing or there are no exams to schedule.

7.6 User Interface Analysis

The system provides a simple CLI (Command Line Interface) to be used. As the user interaction is very limited, the main interaction of the user with the system will be at the moment of its initialization.

It will be at that moment when the user must provide a path to the configuration files to run the algorithm.

During the parsing of the configuration and input files, the user will be notified in case errors are detected during the parsing. User prompt will be required to proceed with the execution or end it prematurely. This will happen twice: at the end of the exam parsing, and at the end of the constraint parsing.

Once the algorithm starts computing the solution, the user system will be constantly updating the status of the population as well as the progress until completion of the execution.

Some output files with data of the execution will be generated at the end in the output directory.

Chapter 8

System Design

In this chapter the prototype will be described. Its architecture, classes hierarchies, user interfaces and some design factors will be detailed along the chapter.

8.1 System Architecture

8.1.1 Package diagram

Due to the large size of the system, the package diagram that can be seen in Figure 8.1 is provided with a description of each packages functionalities.

ExamScheduler

This package contains the main applications that coordinate the components to obtain the desired functionality.

It contains the main class of the prototype, as well as other main classes used during the experimental study.

Domain

This package contains all the entities of the problem: Exams, Constraints, TimeIntervals, etc. It provides the tools to model the problem properly.

Additionally, it provides the input/output mechanisms to obtain such entities from the input files, and write them to the output files.

Logger

The Logger package has two main functionalities. The first one is to provide as output status data of the algorithm such as quality factors of the solutions or the data to build the fitness graph of the execution.

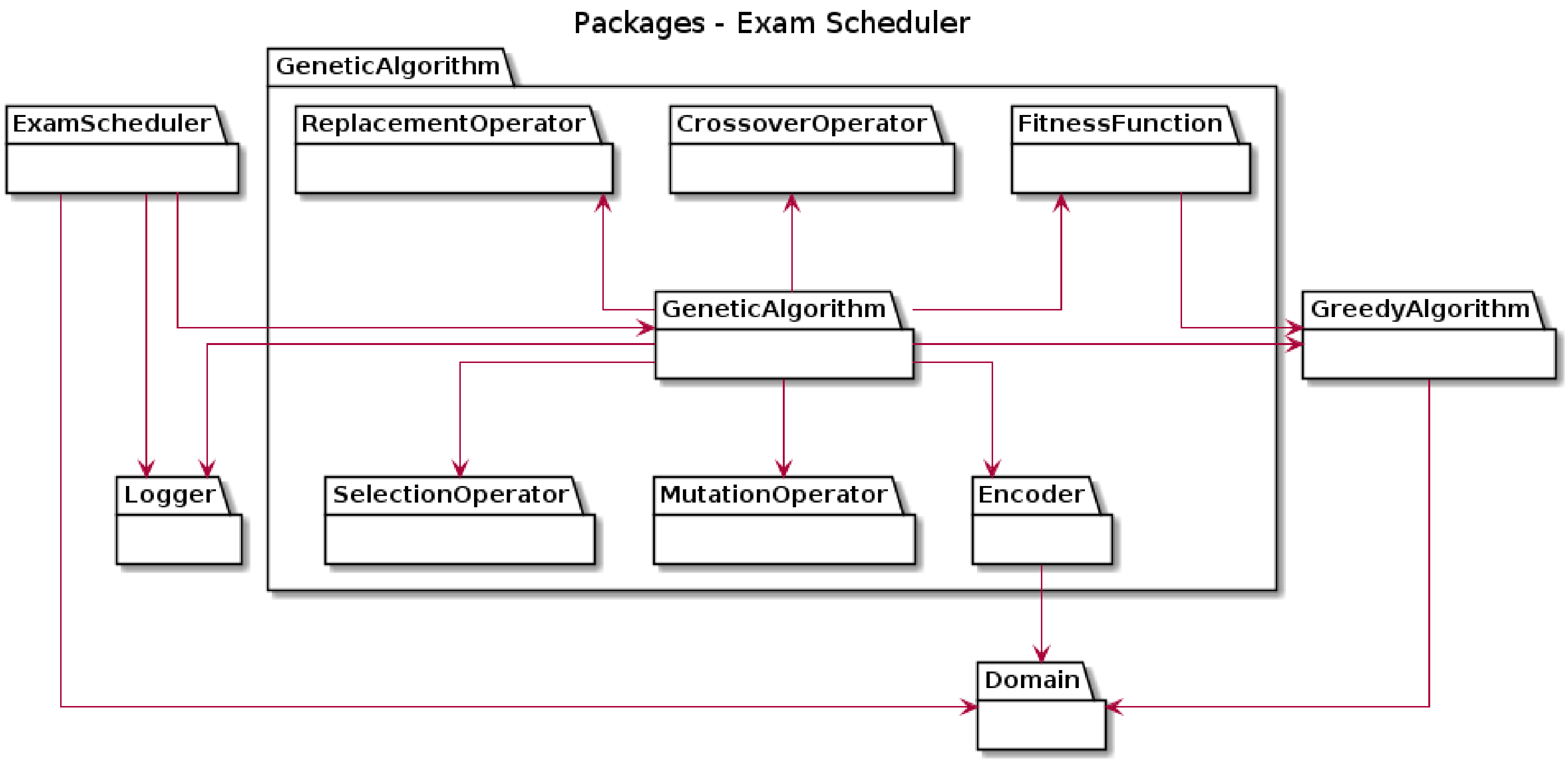


Figure 8.1: Exam Scheduler - Package Diagram

Additionally it also handles the errors that happened during the execution. This is mainly used at the beginning of the execution when parsing the input files to inform the user of possible errors.

GreedyAlgorithm

This package contains the functionality to transform the individuals handled by the Greedy Algorithm into actual solutions of the problem that can be evaluated by a fitness function.

GeneticAlgorithm

The GeneticAlgorithm package contains the implementation of the Genetic Algorithm. Operators have been designed with a Strategy design pattern, so each of the resulting hierarchies will be in its own package.

FitnessFunction

This package contains all the implementations of the fitness function used by the Genetic Algorithm.

Encoder

This package contains the logic to do the encoding of the partial solutions to individuals that can be handled by the Genetic Algorithm.

SelectionOperator

This package is in charge of containing the hierarchy and all the implementations of the selection operators.

MutationOperator

This package is in charge of containing the hierarchy and all the implementations of the mutation operators.

CrossoverOperator

This package is in charge of containing the hierarchy and all the implementations of the crossover operators.

ReplacementOperator

This package is in charge of containing the hierarchy and all the implementations of the replacement operators.

8.1.2 Component diagram

In Figure 8.2 the basic interaction between components in the system is shown. An explanation of each component shown is detailed in this section.

Genetic algorithm

The Genetic Algorithm, previously detailed in Section 5.3 is in charge of guiding the exploration of the search spaces to promising areas, that is, to find fitter individuals to be used as seeds by the Greedy Algorithm.

In Figure 8.2, the Genetic Algorithm interacts mainly with the Fitness Function to evaluate the individuals and provides periodically data to the Logger.

Logger

This component is in charge of generating the output log files, as well as the error files. It is called by the Genetic Algorithm to log status data and by the parser to log errors.

Fitness function

The fitness function works as previously detailed in Section 5.3.6. To do so, as can be seen in Figure 8.2, it calls the Greedy Algorithm to decode the individuals, and the Exam Schedule to check for the fulfilled constraints.

Greedy algorithm

The Greedy Algorithm works as detailed in Section 5.2. The decoding process is done over an Exam Schedule instance, which is why they are related in Figure 8.2.

Exam schedule

The Exam Schedule component is in charge on storing the actual exam schedule, as well of handling the constraints of the execution. The Greedy Algorithm decodes the individuals of the Genetic Algorithm into a single ExamSchedule instance, which is the one that will provide the data to the fitness function to compute the fitness.

On its creation, this component will call the parser in order to load the exams, constraints of the problem instance and configurations of the particular execution in progress.

Parser

The Parser component that can be seen in Figure 8.2, cannot be directly related to a physical component. In reality, this component is divided into: Configurer, ExamParser, and ConstraintParser; each of which is in charge of one of the calls done by the Exam Schedule component. All of them can find errors during the parsing, so all of them are related to the Logger.

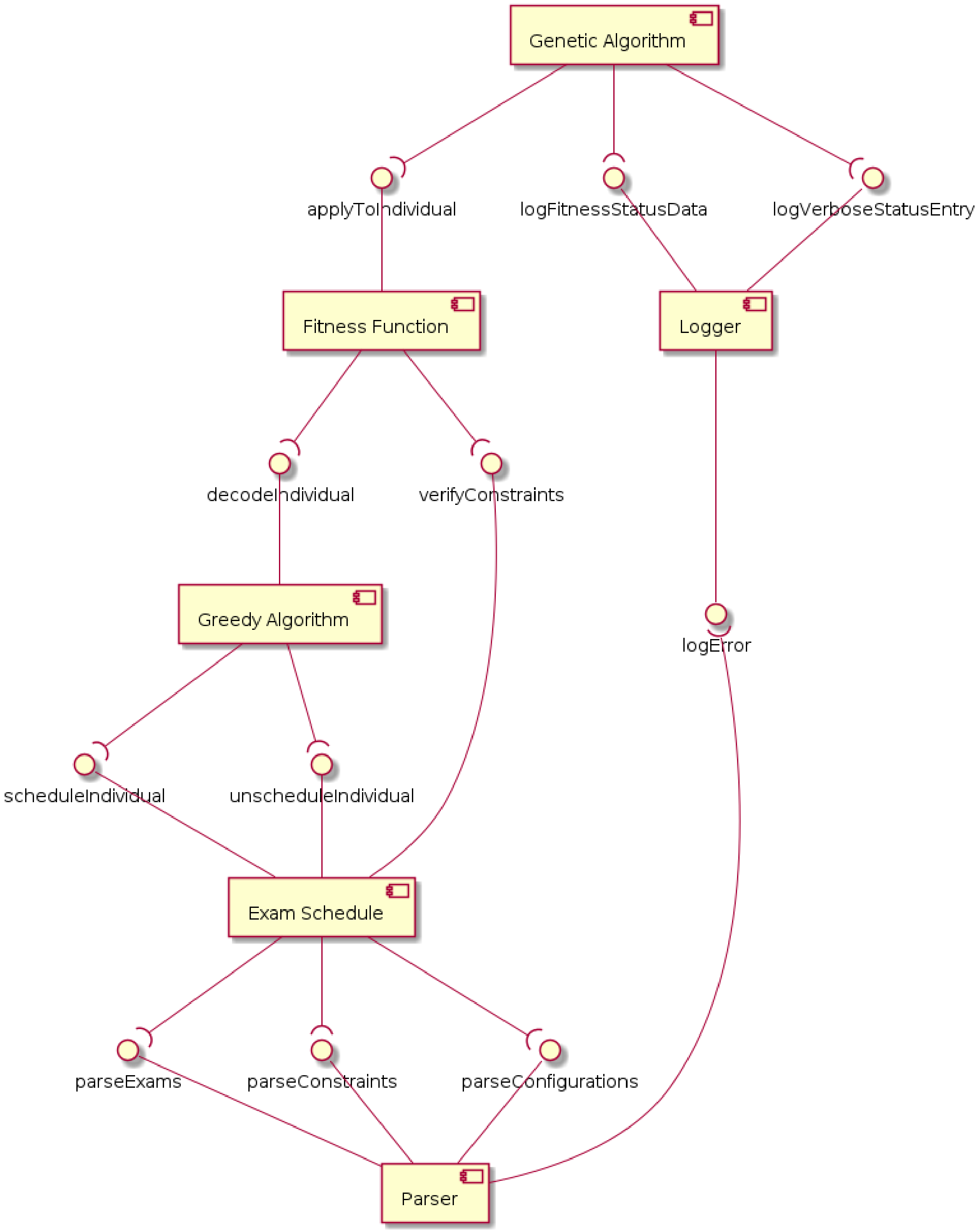


Figure 8.2: Exam Scheduler - Component Diagram

8.2 Class Design

In this section several UML diagrams which describe the architecture and design of the system will be presented. Some of them, the ones that refer to specific design decisions such the one in Figure 8.8, will be also explained in the text.

8.2.1 Class diagram

Due to the size of the class diagram, it was necessary to divide it into small diagrams. However it was possible to fit in one diagram the general architecture of the system without methods signatures. It can be seen in Figure 8.3.

The design of the Genetic Algorithm can be seen in Figure 8.4. Both the operators and the fitness function were designed in a modular way, to make testing with different combinations easier.

The structure of the Domain package can be seen in Figure 8.5. It was not possible to represent all the entities that are in such package, but the diagram provides a clear view of the general structure. This diagram is completed with diagrams in Figures: 8.6, 8.8 and 8.9.

The structure of the Logger package can be seen in Figure 8.7.

8.2.2 Design decisions

This section is devoted to explain parts of the system which required a proper design to achieve the desired extensibility of the system.

Constraints hierarchy

Designing the constraints hierarchy was one of the most complicated design decisions. As previously commented in Section 4.3, we have three types of constraints: hard, soft, and user constraints (being the last ones constraints that can be either soft or hard.)

As can be seen in Figure 8.3 hard and soft constraints are handled in a different way. The problem was how to parse from the Excel file the user constraints in such a way that we do not have duplicated code.

The chosen approach to model this situation can be seen in Figure 8.8. User constraints were modeled as soft constraints, which if marked as hard in the Excel file would imply the creation of a *hardifiedUserConstraint* object, which would encapsulate all the logic of the user constraint and will be used as a hard constraint.

This process is done by the constraint parser which was implemented using the *State* design pattern. This was needed to create each particular type of user constraint when finding the lists in the Excel[1].

[1] This constraints list format are detailed in Section 12.2.2.

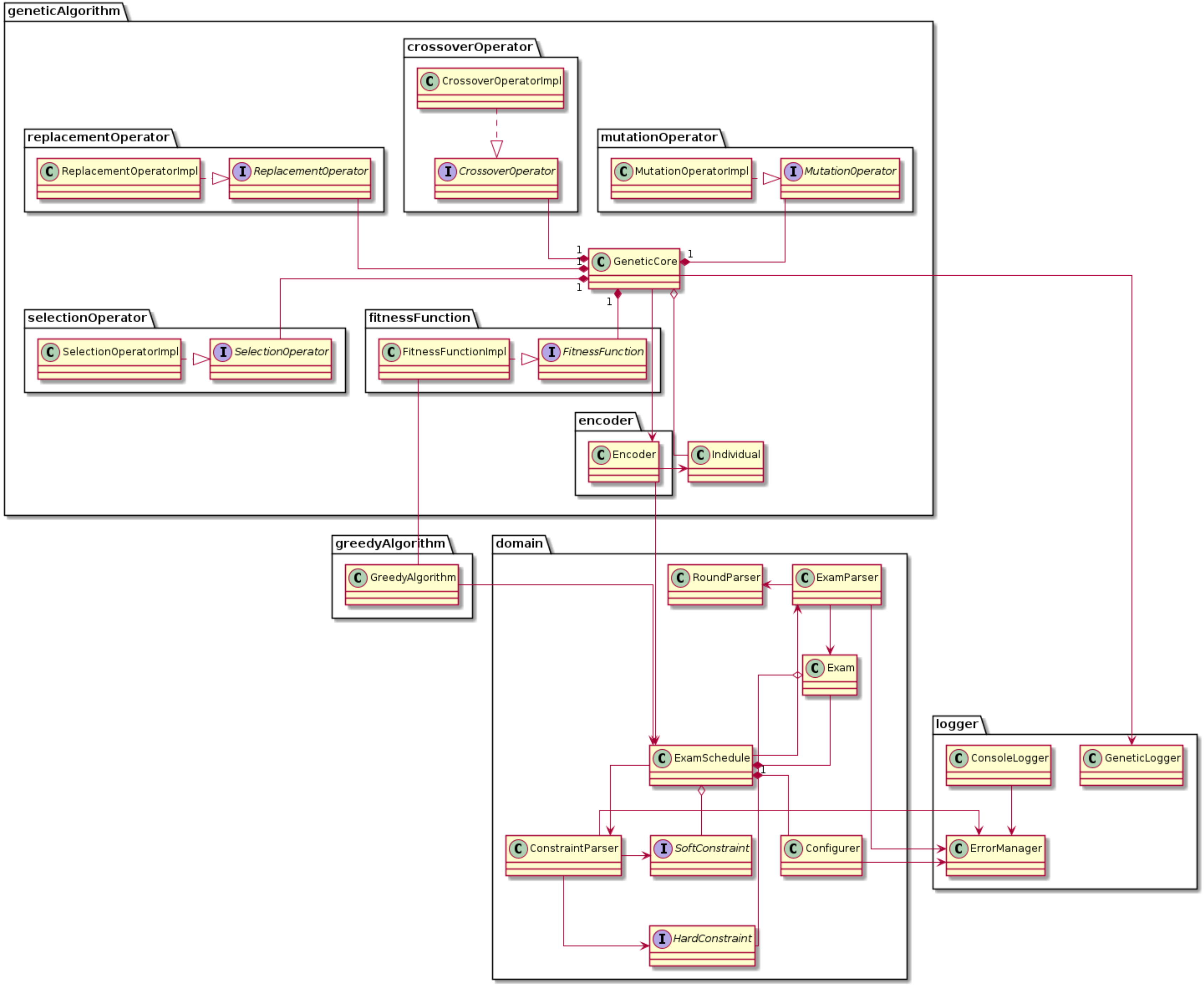


Figure 8.3: Exam Scheduler - Class Diagram

Hard constraints

In Figure 8.5 can be seen that the exams store a list of hard constraints. This list contains the hard constraints that affect the exam. Storing hard constraints that way allows to obtain the set of days in which an exam can be scheduled on easily.

The premise is to iterate that list providing the whole set of days in which exams can be placed. From that set of days the first constraint will return a subset of days which lead to its fulfillment, meaning, that any day in which this constraint would be unfulfilled is eliminated. The following constraints take the output subset of their predecessors. This way, the initial calendar will be reduced to the subset of available days of the exam in a straightforward way.

Viable days iteration

Considering an scenario in which there are no constraints, all exams would have the same set of available days. A linear iteration from beginning to end of such sets for all exams would lead to a timetable in which all exams are packaged in the first days.

To prevent this scenario, the set of viable days in which an exam can be scheduled is randomly shuffled before starting to iterate it. This results in the exams spread along the whole calendar.

Constraint counter for fitness function

From the beginning a linear fitness function was the chosen option, because it was proven acceptably effective in this kind of problems. However, to ensure the extensibility of the system and to make it easier to test different fitness functions, a *Visitor* design pattern has been used.

A fitness function, is a sequence of variables with their coefficients. The coefficients will be provided by the user as it is detailed in Section 12.2.1, so if any entity holds the value of such variables, any fitness function could be easily built just asking that entity.

The problem was that each type of unfulfilled user constraint needed to be counted separately, while other soft constraint such as the *NumericalComplexityPenalization* was the result of a function to be computed. All soft constraints are in a list in *ExamSchedule*, and so the concrete type of the constraint is unknown. This is the typical *Visitor* design pattern scenario which is what is presented as solution in Figure 8.9.

8.2.3 Diagrams

This section shows the diagrams that are linked in the previous sections.

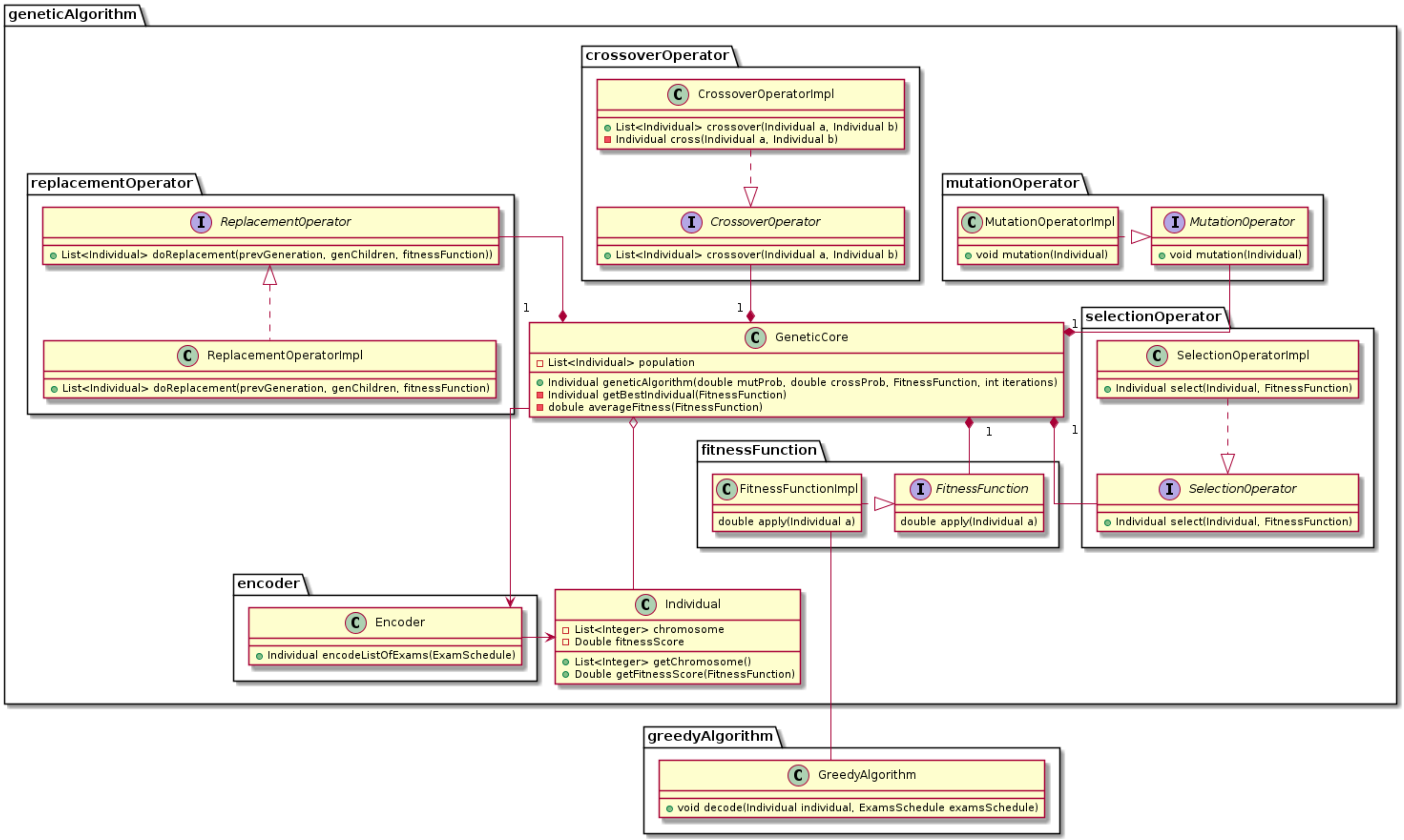

geneticAlgorithm
crossoverOperator
CrossoverOperatorImpl
List<Individual> crossover(Individual a, Individual b)
Individual cross(Individual a, Individual b)
CrossoverOperator
List<Individual> crossover(Individual a, Individual b)
replacementOperator
ReplacementOperator
List<Individual> doReplacement(prevGeneration, genChildren, fitnessFunction))
ReplacementOperatorImpl
List<Individual> doReplacement(prevGeneration, genChildren, fitnessFunction)
mutationOperator
MutationOperatorImpl
void mutation(Individual)
MutationOperator
void mutation(Individual)
selectionOperator
SelectionOperatorImpl
Individual select(Individual, FitnessFunction)
SelectionOperator
Individual select(Individual, FitnessFunction)
GeneticCore
List<Individual> population
Individual geneticAlgorithm(double mutProb, double crossProb, FitnessFunction, int iterations)
Individual getBestIndividual(FitnessFunction)
dobule averageFitness(FitnessFunction)
1
1
1
1
1
fitnessFunction
FitnessFunctionImpl
double apply(Individual a)
FitnessFunction
double apply(Individual a)
encoder
Encoder
Individual encodeListOfExams(ExamSchedule)
Individual
List<Integer> chromosome
Double fitnessScore
List<Integer> getChromosome()
Double getFitnessScore(FitnessFunction)
greedyAlgorithm
GreedyAlgorithm
void decode(Individual individual, ExamsSchedule examsSchedule)

Figure 8.4: Exam Scheduler - Genetic Algorithm Design Diagram

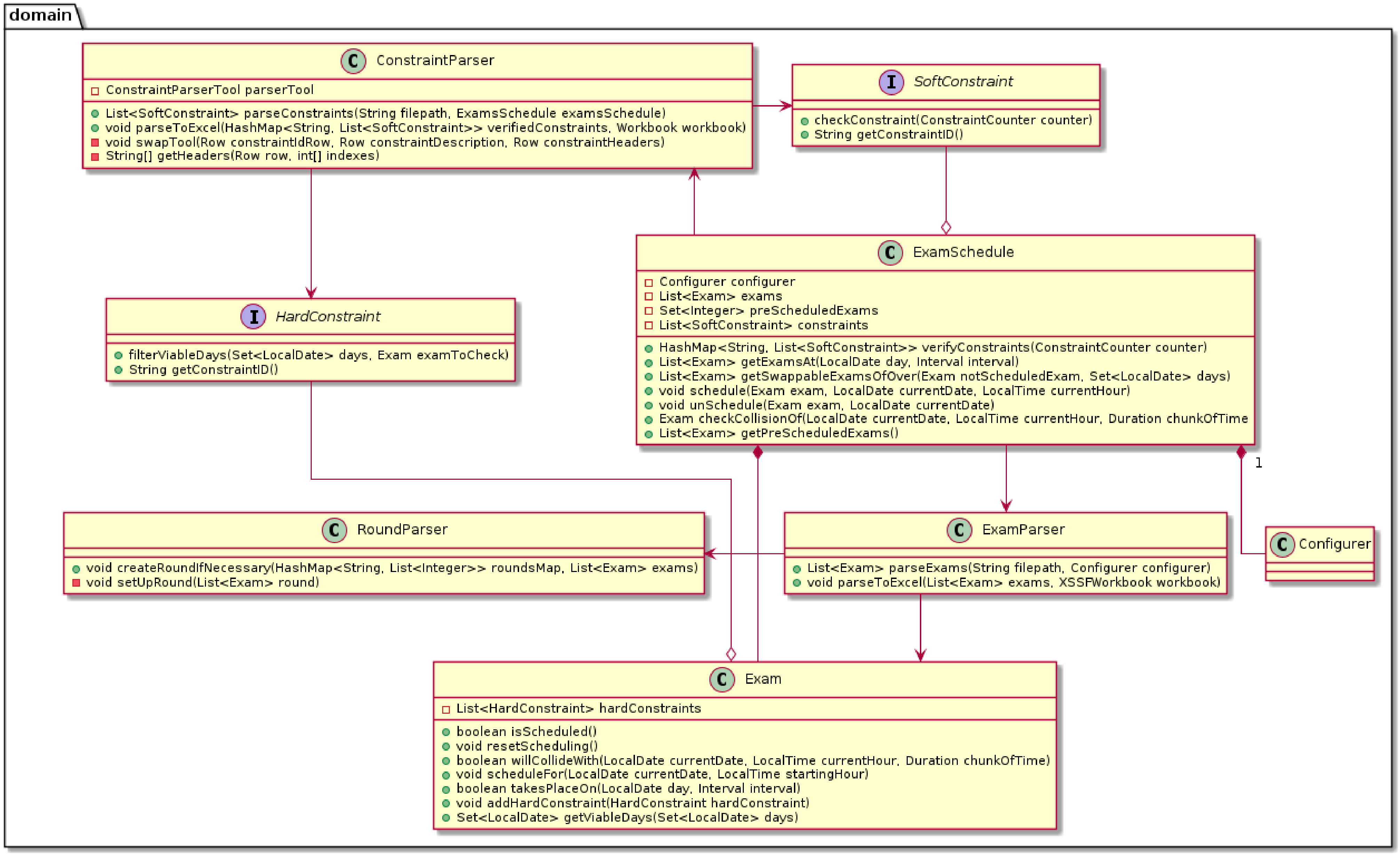

domain
ConstraintParser
□ ConstraintParserTool parserTool
● List<SoftConstraint> parseConstraints(String filepath, ExamsSchedule examsSchedule)
● void parseToExcel(HashMap<String, List<SoftConstraint>> verifiedConstraints, Workbook workbook)
■ void swapTool(Row constraintIdRow, Row constraintDescription, Row constraintHeaders)
■ String[] getHeaders(Row row, int[] indexes)
SoftConstraint
● checkConstraint(ConstraintCounter counter)
● String getConstraintID()
HardConstraint
● filterViableDays(Set<LocalDate> days, Exam examToCheck)
● String getConstraintID()
ExamSchedule
□ Configurer configurer
□ List<Exam> exams
□ Set<Integer> preScheduledExams
□ List<SoftConstraint> constraints
● HashMap<String, List<SoftConstraint>> verifyConstraints(ConstraintCounter counter)
● List<Exam> getExamsAt(LocalDate day, Interval interval)
● List<Exam> getSwappableExamsOfOver(Exam notScheduledExam, Set<LocalDate> days)
● void schedule(Exam exam, LocalDate currentDate, LocalTime currentHour)
● void unSchedule(Exam exam, LocalDate currentDate)
● Exam checkCollisionOf(LocalDate currentDate, LocalTime currentHour, Duration chunkOfTime
● List<Exam> getPreScheduledExams()
1
RoundParser
● void createRoundIfNecessary(HashMap<String, List<Integer>> roundsMap, List<Exam> exams)
■ void setUpRound(List<Exam> round)
ExamParser
● List<Exam> parseExams(String filepath, Configurer configurer)
● void parseToExcel(List<Exam> exams, XSSFWorkbook workbook)
Configurer
Exam
□ List<HardConstraint> hardConstraints
● boolean isScheduled()
● void resetScheduling()
● boolean willCollideWith(LocalDate currentDate, LocalTime currentHour, Duration chunkOfTime)
● void scheduleFor(LocalDate currentDate, LocalTime startingHour)
● boolean takesPlaceOn(LocalDate day, Interval interval)
● void addHardConstraint(HardConstraint hardConstraint)
● Set<LocalDate> getViableDays(Set<LocalDate> days)

Figure 8.5: Exam Scheduler - Domain Design Diagram

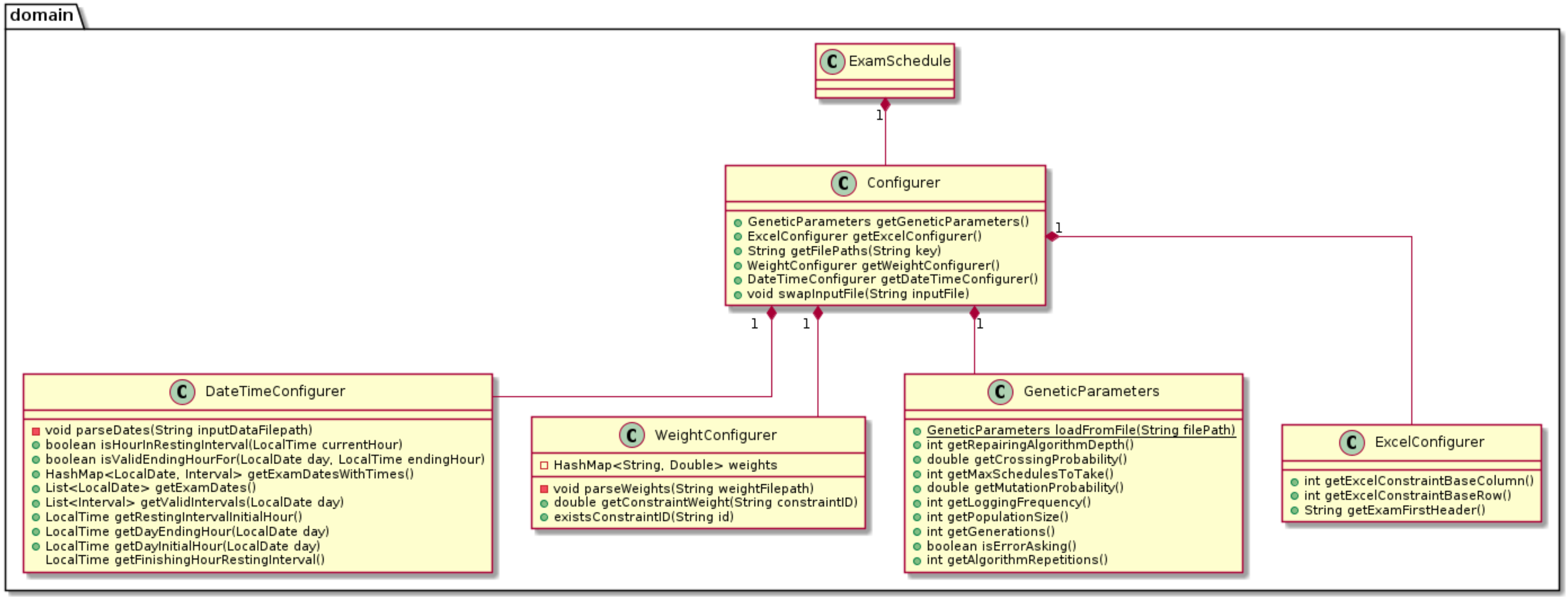

domain
ExamSchedule
1
Configurer
GeneticParameters getGeneticParameters()
ExcelConfigurer getExcelConfigurer()
String getFilePaths(String key)
WeightConfigurer getWeightConfigurer()
DateTimeConfigurer getDateTimeConfigurer()
void swapInputFile(String inputFile)
1
1
1
1
DateTimeConfigurer
void parseDates(String inputDataFilepath)
boolean isHourInRestingInterval(LocalTime currentHour)
boolean isValidEndingHourFor(LocalDate day, LocalTime endingHour)
HashMap<LocalDate, Interval> getExamDatesWithTimes()
List<LocalDate> getExamDates()
List<Interval> getValidIntervals(LocalDate day)
LocalTime getRestingIntervalInitialHour()
LocalTime getDayEndingHour(LocalDate day)
LocalTime getDayInitialHour(LocalDate day)
LocalTime getFinishingHourRestingInterval()
WeightConfigurer
HashMap<String, Double> weights
void parseWeights(String weightFilepath)
double getConstraintWeight(String constraintID)
existsConstraintID(String id)
GeneticParameters
GeneticParameters loadFromFile(String filePath)
int getRepairingAlgorithmDepth()
double getCrossingProbability()
int getMaxSchedulesToTake()
double getMutationProbability()
int getLoggingFrequency()
int getPopulationSize()
int getGenerations()
boolean isErrorAsking()
int getAlgorithmRepetitions()
ExcelConfigurer
int getExcelConstraintBaseColumn()
int getExcelConstraintBaseRow()
String getExamFirstHeader()

Figure 8.6: Exam Scheduler - Configurers Design Diagram

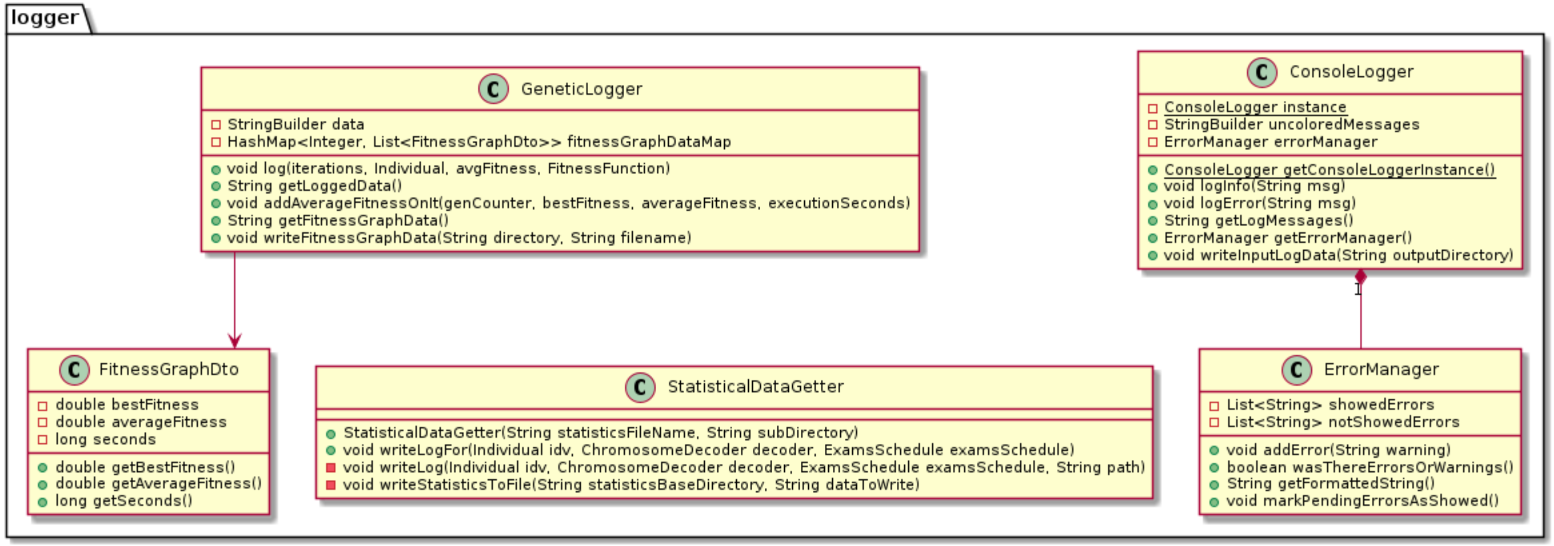

logger
GeneticLogger
□ StringBuilder data
□ HashMap<Integer, List<FitnessGraphDto>> fitnessGraphDataMap
● void log(iterations, Individual, avgFitness, FitnessFunction)
● String getLoggedData()
● void addAverageFitnessOnIt(genCounter, bestFitness, averageFitness, executionSeconds)
● String getFitnessGraphData()
● void writeFitnessGraphData(String directory, String filename)
ConsoleLogger
□ ConsoleLogger instance
□ StringBuilder uncoloredMessages
□ ErrorManager errorManager
● ConsoleLogger getConsoleLoggerInstance()
● void logInfo(String msg)
● void logError(String msg)
● String getLogMessages()
● ErrorManager getErrorManager()
● void writeInputLogData(String outputDirectory)
1
FitnessGraphDto
□ double bestFitness
□ double averageFitness
□ long seconds
● double getBestFitness()
● double getAverageFitness()
● long getSeconds()
StatisticalDataGetter
● StatisticalDataGetter(String statisticsFileName, String subDirectory)
● void writeLogFor(Individual idv, ChromosomeDecoder decoder, ExamsSchedule examsSchedule)
■ void writeLog(Individual idv, ChromosomeDecoder decoder, ExamsSchedule examsSchedule, String path)
■ void writeStatisticsToFile(String statisticsBaseDirectory, String dataToWrite)
ErrorManager
□ List<String> showedErrors
□ List<String> notShowedErrors
● void addError(String warning)
● boolean wasThereErrorsOrWarnings()
● String getFormattedString()
● void markPendingErrorsAsShowed()

Figure 8.7: Exam Scheduler - Logger Design Diagram

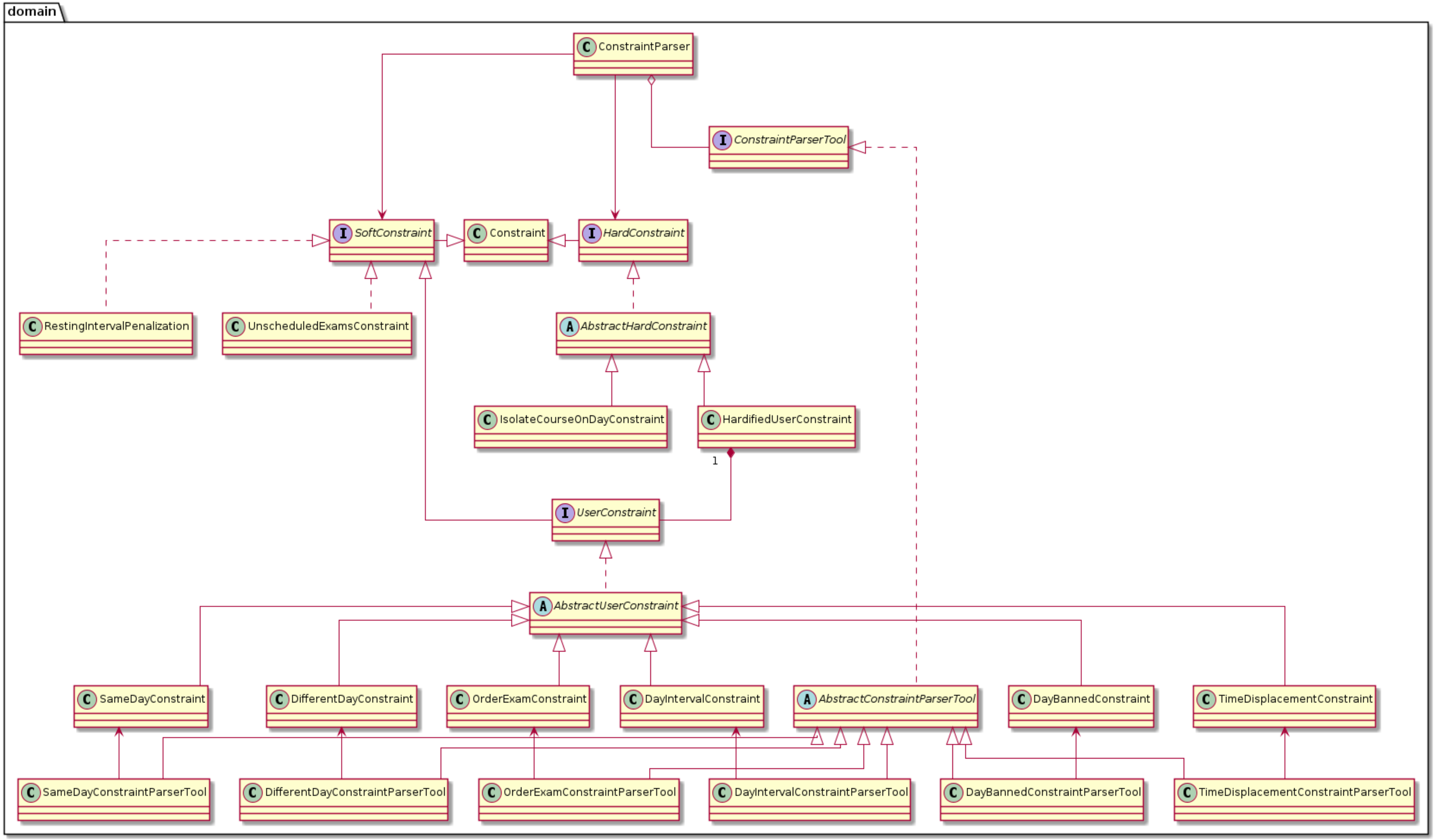

domain
ConstraintParser
ConstraintParserTool
SoftConstraint
Constraint
HardConstraint
RestingIntervalPenalization
UnscheduledExamsConstraint
AbstractHardConstraint
IsolateCourseOnDayConstraint
HardifiedUserConstraint
1
UserConstraint
AbstractUserConstraint
SameDayConstraint
DifferentDayConstraint
OrderExamConstraint
DayIntervalConstraint
AbstractConstraintParserTool
DayBannedConstraint
TimeDisplacementConstraint
SameDayConstraintParserTool
DifferentDayConstraintParserTool
OrderExamConstraintParserTool
DayIntervalConstraintParserTool
DayBannedConstraintParserTool
TimeDisplacementConstraintParserTool

Figure 8.8: Exam Scheduler - Constraint Hierarchy Diagram

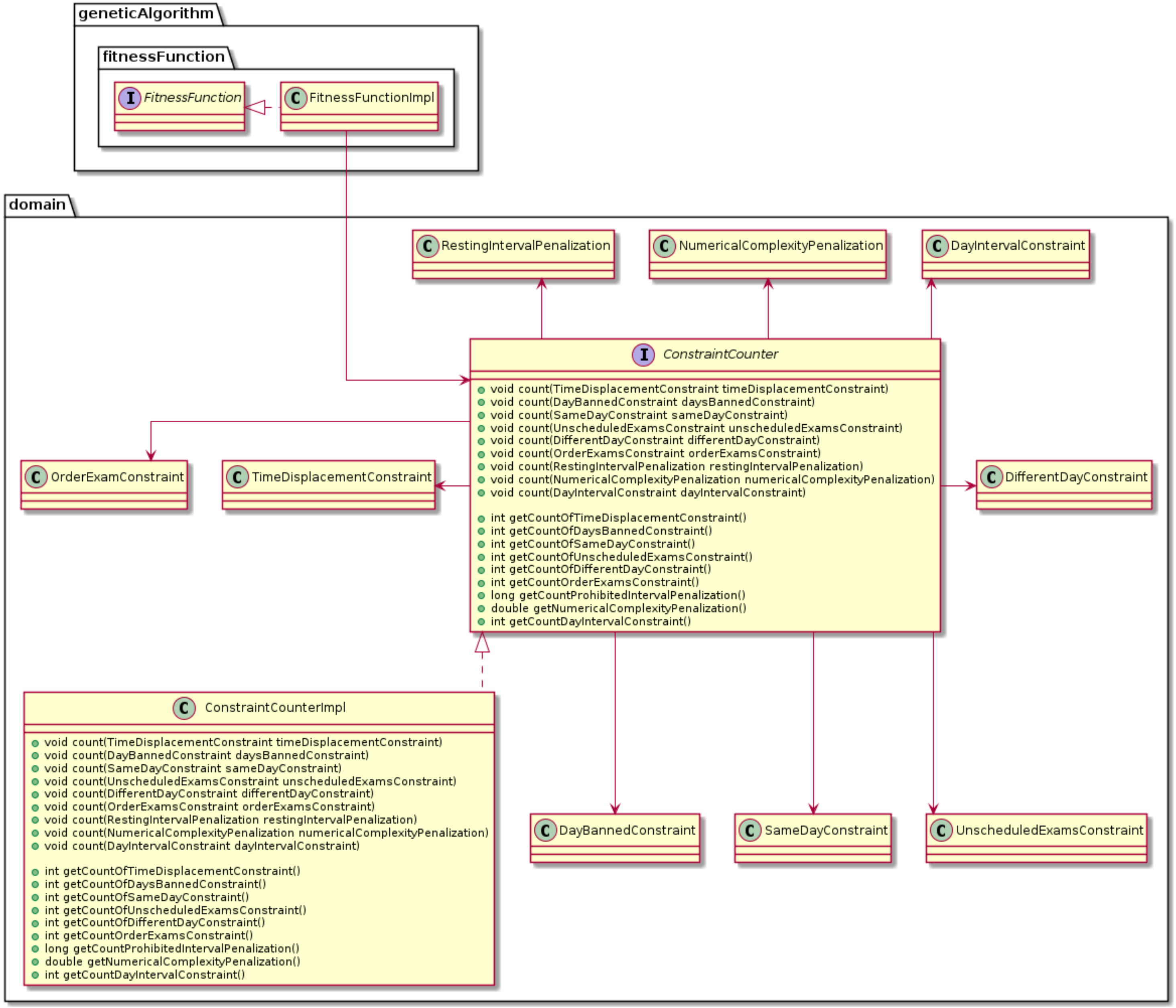

geneticAlgorithm
fitnessFunction
FitnessFunction
FitnessFunctionImpl
domain
RestingIntervalPenalization
NumericalComplexityPenalization
DayIntervalConstraint
ConstraintCounter
void count(TimeDisplacementConstraint timeDisplacementConstraint)
void count(DayBannedConstraint daysBannedConstraint)
void count(SameDayConstraint sameDayConstraint)
void count(UnscheduledExamsConstraint unscheduledExamsConstraint)
void count(DifferentDayConstraint differentDayConstraint)
void count(OrderExamsConstraint orderExamsConstraint)
void count(RestingIntervalPenalization restingIntervalPenalization)
void count(NumericalComplexityPenalization numericalComplexityPenalization)
void count(DayIntervalConstraint dayIntervalConstraint)
int getCountOfTimeDisplacementConstraint()
int getCountOfDaysBannedConstraint()
int getCountOfSameDayConstraint()
int getCountOfUnscheduledExamsConstraint()
int getCountOfDifferentDayConstraint()
int getCountOrderExamsConstraint()
long getCountProhibitedIntervalPenalization()
double getNumericalComplexityPenalization()
int getCountDayIntervalConstraint()
OrderExamConstraint
TimeDisplacementConstraint
DifferentDayConstraint
ConstraintCounterImpl
void count(TimeDisplacementConstraint timeDisplacementConstraint)
void count(DayBannedConstraint daysBannedConstraint)
void count(SameDayConstraint sameDayConstraint)
void count(UnscheduledExamsConstraint unscheduledExamsConstraint)
void count(DifferentDayConstraint differentDayConstraint)
void count(OrderExamsConstraint orderExamsConstraint)
void count(RestingIntervalPenalization restingIntervalPenalization)
void count(NumericalComplexityPenalization numericalComplexityPenalization)
void count(DayIntervalConstraint dayIntervalConstraint)
int getCountOfTimeDisplacementConstraint()
int getCountOfDaysBannedConstraint()
int getCountOfSameDayConstraint()
int getCountOfUnscheduledExamsConstraint()
int getCountOfDifferentDayConstraint()
int getCountOrderExamsConstraint()
long getCountProhibitedIntervalPenalization()
double getNumericalComplexityPenalization()
int getCountDayIntervalConstraint()
DayBannedConstraint
SameDayConstraint
UnscheduledExamsConstraint

Figure 8.9: Exam Scheduler - Constraint Counter (Visitor) Diagram

8.3 User Interface Design

The developed prototype is a simple command line application, therefore it does not have a complex user interface. This section is devoted to explain the console interface of the application.

The unique interaction that the user has with the system is to launch it, which is the process commented in this section. Further information about input file formats is given in Chapter 12.

The command to launch the application is the following:

```
$ java -jar examScheduler.jar <pathToFilesFile> <nameOfOutputFile>
        <nameOfTheStaticticsFile>
```

The arguments will be explained in detail in Chapter 12. The first one is a path to a file containing the paths of other configuration files. The second parameter is optional and it generates a file with some quality measures of each generated solution.

The execution of the command could lead to one of the following scenarios:

1. One whole configuration file is missing, one configuration within that file is missing, or the input data Excel has an invalid format.
2. A configuration option content is invalid. For instance, writing an alphanumeric string in a numeric configuration.
3. Everything has the right format.

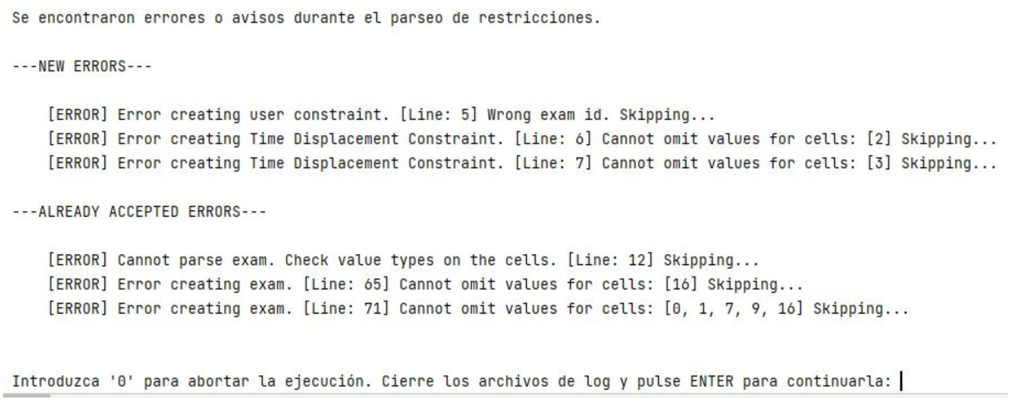
In the first scenario the application will end abruptly showing an error message. Extra information about the error will be provided afterwards, some examples are:

```
Cause: Missing properties in weights configuration file.
The following properties are mandatory: [DB, TD, SD, DD, OE,
DI, RIP, NCP, UE, SCDD]

Cause: Could not find dates and times configuration file.

Cause: Could not parse input Excel file:
files/pruebas/v6_sF_numericalComplexityBad.xlsx
```

In the second scenario, the application will not end, but it will inform the user. In case the conflicting value is in a configuration file the application will assign "0" as default value, and in case it is within an exam or constraint it will skip such exam or constraint. Note that since exams are parsed before constraints an error in an exam will make that every constraint linked to that exam will raise an error stating that the provided exam id does not exist.

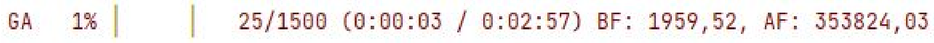
The user will be asked if the execution must continue a maximum of two times, when finishing parsing the exam and when finishing parsing the constraints (in case there were any errors). When doing so, both in a file and in the console, an output such as the the one shown in Figure 8.10 will be displayed.

Finally in the third scenario (or in case the errors of the second scenario were accepted by the user) the application will inform of the parameters of the algorithm and begin the computation. While doing so a progress bar will be updated in each iteration. Such progress bar, which can be seen in Figure 8.11, contains the following information:



```
Se encontraron errores o avisos durante el parseo de restricciones.

---NEW ERRORS---

    [ERROR] Error creating user constraint. [Line: 5] Wrong exam id. Skipping...
    [ERROR] Error creating Time Displacement Constraint. [Line: 6] Cannot omit values for cells: [2] Skipping...
    [ERROR] Error creating Time Displacement Constraint. [Line: 7] Cannot omit values for cells: [3] Skipping...

---ALREADY ACCEPTED ERRORS---

    [ERROR] Cannot parse exam. Check value types on the cells. [Line: 12] Skipping...
    [ERROR] Error creating exam. [Line: 65] Cannot omit values for cells: [16] Skipping...
    [ERROR] Error creating exam. [Line: 71] Cannot omit values for cells: [0, 1, 7, 9, 16] Skipping...


Introduzca '0' para abortar la ejecución. Cierre los archivos de log y pulse ENTER para continuarla:
```

Figure 8.10: Exam Scheduler - Errors after constraint parsing



```
GA   1% |      |   25/1500 (0:00:03 / 0:02:57) BF: 1959,52, AF: 353824,03
```

Figure 8.11: Exam Scheduler - Progress Bar

- Percentage of completion.
- Computed generations.
- Generations to be computed.
- Time of execution.
- Expected ending time of execution.
- Best fitness of the population.
- Average fitness of the population.

Chapter 9

System implementation

In this chapter tools and methods used in the project will be described, also enumerating other considered possibilities.

9.1 Programming Languages

9.1.1 Prototype

Three programming languages were considered for implementing the application:

1. **Java:** Java is the core programming language taught at the Degree in Software Engineering. It is a multi-platform programming language due to the middle compilation to a Java Virtual Machine (JVM) Byte-code. Even though the inclusion of the JVM as an intermediary implies a performance loss, it also brings some advantages as the garbage collector, which automatically frees not in use allocated memory.
2. **C#:** C# provides a higher performance that Java. It is also one of the most used programming languages and provides ”linq”, a library that facilitates the management of collections.
3. **Python:** It is a widely used programming language by researchers, and so it has plenty of libraries to cover many functionalities related with optimization problems. However, for this specific problem where design and performance were crucial, Python was not suitable.

In the end, the selected language was Java. The determinant factors for such decision where the following:

- Users can run the same executable on different operating systems.
- Java is not only taught at the Degree in Software Engineering, but also one of the most used programming languages.
- Java allows an acceptable performance for the solution.

9.1.2 Data analysis

Python was the straightforward decision in this aspect. As stated before it has plenty of libraries among which we can point out: graph generation and large data structures management. Those two aspects are provided by: *matplotlib* and *pandas*.

9.1.3 Documentation

- **LaTeX:** It is a document preparation language which is widely used by researchers. The other considered option was *Microsoft Word*, but it did not provide as many facilities as LaTeX did. Cites and cross-refereces were a must in this project and they are easier to handle in LaTeX than in *Word*.
- **PFCEutio(V1.4):** This is the base template provided by the tutors for the final documentation. Despite many changes were applied to it, it was the base of this document.

9.2 Development Tools

9.2.1 IntelliJ IDEA

This is an IDE (Integrated Development Enviroment) developed by *JetBrains*. *JetBrains*' IDEs provide programming conventions analysis, support for packaging tools such as *Maven*, integrated VCS (Version Control Systems) like *Git*, and many others.

Is requires a license to be used, but in this case being a student in the University of Oviedo allowed for getting it free.

9.2.2 Spyder

This is a simple Python IDE included in *Conda*. It provides a simple file view and tools to write and debug code.

Any Python IDE would be fine for the analysis part, Spyder was the chosen one because I have being using it for some years. Other options such as Visual Studio Code or PyCharm would have not made a difference in this aspect.

9.2.3 PlantUML

This is an easy way to create UML diagrams by means of text. It was used to model the system and to create all the diagrams in Chapter 8.

9.3 System Development

9.3.1 Difficulties

In this section the main difficulties of the prototype implementation will be described.

Performance

Initially, executions of the prototype took really long times even for problem instances with small size. There were many reasons for this: large data structures, complex computations, etc.

The decisive improvement to performance came when using *memoization*. What was happening was that the computation of some values that took low execution times individually was taking a large amount when considering all together. This particularly happened when computing the fitness of the individuals in each generation.

Taking into account that individuals from the previous generation can go to the new generation implies that some fitness of a generation were already computed before. By storing into the individual its fitness the first time it is computed and just checking that value afterwards,[1] performance increased considerably.

Extensibility

One of the main desired qualities of the prototype was to be easily extensible. This means that it must not tedious to extend it when wanting to implement new functionalities, operators, fitness functions, etcetera.

Every subsystem needed to be studied and understood in order to model them properly as a whole. Several design decisions were made to reach the objective. For that, some design patters were used as they are well-tested solutions to common programming design problems.

9.3.2 Code classes description

In Section A.1 of Appendix A additional documentation delivered with this document is referenced to detail this topic.

[1] Note that since mutations are only performed over new generated children, there is no way in which a previously created individual can change its chromosome. Thus an individual's fitness can be computed just once.

Chapter 10

System tests

To test the provided solution two main aspects need to be considered separately:

- **Input/Output.** It is necessary to test that the systems behaviour over good and bad input files, and to check that the generated output files are the expected in each case.
- **Algorithm efficacy.** Some comparative tests are needed to showcase the quality of the results provided by the algorithm.

The Algorithm efficacy will be tested in Section 11.5.

The input and output tests will be done considering the single use case detailed in section 7.5. First a list of test coverage items will be provided, and from them a list of test cases will be built.

10.1 Test Design

Since the main user interaction are the input it is necessary to ensure that the algorithm works as expected when receiving incorrect input files. What follows is the process of creation of the test cases. The list of test coverage items is the following:

1 Configuration (Input)

  1.1 Configuration files.

    1.1.1 Date and time.
    1.1.2 Excel configuration.
    1.1.3 Files file.
    1.1.4 Genetic parameters.
    1.1.5 Weights.

  1.2 Valid format.

  1.3 Invalid format (Invalid).

    1.3.1 Wrong path.
    1.3.2 At least one missing configuration option.
    1.3.3 Al least one empty configuration option.

1.3.4 At least one invalid format for a configuration option.

2 Exams tab (Excel data file) (Input).

2.1 Valid format.

2.1.1 Exam type.
2.1.1.1 Scheduled exam.
2.1.1.2 Unscheduled exam.
2.1.2 Missing exam mandatory field.
2.1.2.1 Course.
2.1.2.2 Semester.
2.1.2.3 ID.
2.1.2.4 Modality.
2.1.2.5 Duration.
2.1.3 Invalid datatype for mandatory field.
2.1.3.1 Course.
2.1.3.2 Semester.
2.1.3.3 ID.
2.1.3.4 Modality.
2.1.3.5 Duration.
2.1.4 Not unique IDs for exams.

2.2 Invalid tab format.

2.3 Exams generated.

2.3.1 0 exams (Invalid).

2.3.2 1+ exams.

3 Constraints tab (Excel data file) (Input).

3.1 Valid format.

3.1.1 Constraint type.
3.1.1.1 Soft.
3.1.1.2 Hard.
3.1.2 User Constraint type.
3.1.2.1 Time Displacement.
3.1.2.2 Day Banned.
3.1.2.3 Same Day.
3.1.2.4 Different Day.
3.1.2.5 Order Exams.
3.1.2.6 Day Interval.
3.1.3 Invalid field for constraint.
3.1.3.1 Incorrect format type.
3.1.3.2 Missing value.
3.1.4 Implicit contradiction between constraints.

3.2 Invalid tab format (The program directly ends).

4 Calendar tab (Excel data file) (Input).

4.1 Valid format.

4.2 Invalid tab format.

5 Output

5.1 Number of unscheduled exams.

5.1.1 0.

5.1.2 1+.

10.1.1 Input tests design

There are two groups of inputs: the configuration files, and the Excel data file. First of all let us describe which possible inputs could happen. There are five different configuration files and each of them have a valid or invalid format.

Testing all possible combinations, in addition to the necessary tests for the Excel data file is not viable, so a combination technique will be used. In this, *each choice* is considered. Initially a base scenario will have all files in a valid format. From that a variation will be done over a file to invalidate it, following *each choice*, that is, using the minimum number of test cases.[1]

Even though all files work exactly the same, there is an exception, which forces to consider an additional test case. The Weights file works in a different way from its peers. When an invalid value is assigned, it assigns a default value and warns the user.

The Excel data file has three tabs: exam, user constraints and calendar; which require different tests. This part of the test design will follow a similar approach to the previous part. An initial base case in which all files have a valid format will be considered. From that, test cases for each test coverage item that would result in the application ending abruptly will be tested satisfying the one-to-one approach to avoid error masking. Since the input is a file, many possible errors for exams and constraints can be tested in different rows. That way all of them will be tested with a single test case.

10.1.2 Output test design

As a result of the execution several files are generated. They are detailed in Chapter 12, but in this chapter it will be specified the conditions under which each file is generated.

To fully understand the explanation, it is necessary to comment the simple CLI design of the application which can be seen in Figure 10.1. The application will pause twice before starting the computations of the algorithm. The user will be able to cancel or proceed the execution while seeing the detected errors. If no errors were detected the application will skip that stopping point.

The stopping points are the following:

- **The end of the exam parsing.** This includes both the exams and the configuration parsing.
- **The end of the constraint parsing.**

[1] Even if this could be seen as a pseudo base-choice combination technique, truly applying it would imply many more test cases.

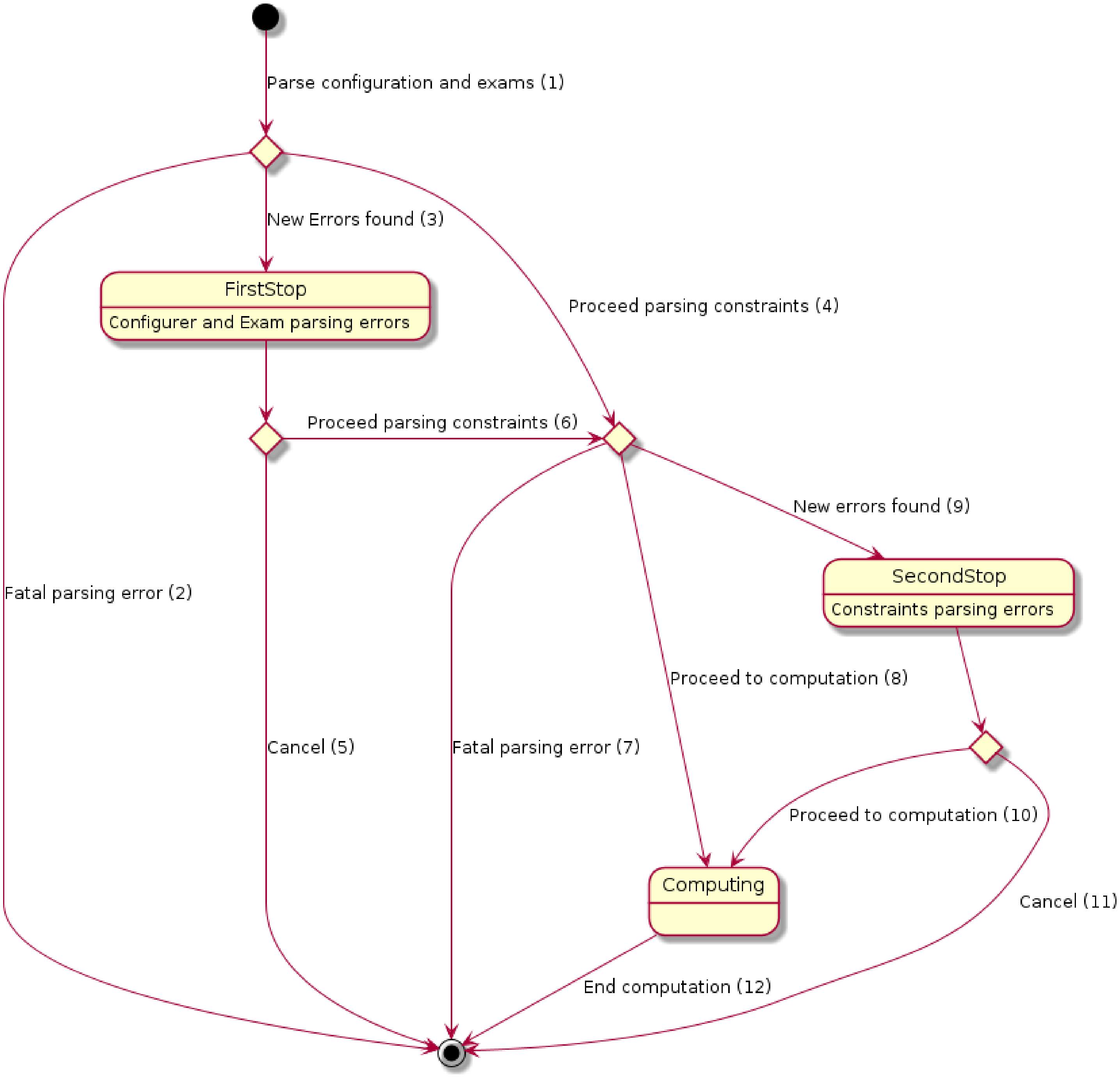

Parse configuration and exams (1)
New Errors found (3)
FirstStop
Configurer and Exam parsing errors
Proceed parsing constraints (4)
Proceed parsing constraints (6)
New errors found (9)
SecondStop
Constraints parsing errors
Fatal parsing error (2)
Proceed to computation (8)
Cancel (5)
Fatal parsing error (7)
Proceed to computation (10)
Computing
Cancel (11)
End computation (12)

Figure 10.1: Stopping points CLI - Exam Scheduler.

At those points the following output files need to be updated:

- **errorLog.txt:** This will have the errors detected at the stopping moment.
- **inputUncoloredLog.txt:** This will track the parsing log at the stopping moment.

The other output files will all be generated when completing the computation of the algorithm. Therefore, their right generation can be checked in any valid input test case.

The output exam schedule file of the algorithm have also some test conditions, such as the number of unscheduled exams, or the number of unfulfilled user constraints. Furthermore the contents of other files could also be considered test conditions.

It is necessary to focus which test conditions are going to be tested because it is not feasible to test everything. This project is focused to schedule exams, and because of that the only output test condition to be tested is the number of unscheduled exams.

10.1.3 Interface tests

As stated in the previous section, even if simple, there is an interface which has some transitions to be considered as shown in Figure 10.1. In the mentioned figure an identifier is also assigned to each possible transition in the application state.

In parallel with the rest of the valid scenarios design, the *simple path* technique will also be followed to test that this transitions work as expected.

The paths to be considered are the following; each of them will have an identifier so that it can be linked in the trazability matrix:

- **Path01:** Start - (1) - (2) End
- **Path02:** Start - (1) - (4) - (7) - End
- **Path03:** Start - (1) - (3) - (5) - End
- **Path04:** Start - (1) - (4) - (9) - (11) - End
- **Path05:** Start - (1) - (3) - (6) - (9) - (10) - (12) - End
- **Path06:** Start - (1) - (4) - (8) - End

10.2 Test Cases

From the previous design two groups of test cases will be built. The ones referring to the invalid test coverage items, in which it is globally tested that the right error handling is done; and the ones referring to valid test coverage items, in which the application is tested under normal circumstances.

ID	Description	Expected Result	Status
TC1	An invalid path is provided for the “Date and time” configuration file.	The application ends its execution showing an error specifying that such file was not found.	Passed
TC2	“constraintsFirstRow” option erased on the Excel configuration file.	The application ends its execution showing an error specifying that an option is missing in the Excel configuration file and stating the options that must be present.	Passed
TC3	Number of generations left as empty property in GeneticParameters file.	The application ends its execution showing an error specifying that there was a format problem in the file	Passed
TC4	“13:00asa” configured as the initial hour of the resting interval in the “Date and time” configuration file.	The application ends its execution showing an error that there was a format problem in the file.	Passed
TC5	An invalid path is provided by console for the “filesFile” configuration file.	The application ends its execution showing an error specifying that such file was not found.	Passed
TC6	An invalid format is provided in the data Excel exam tab. Headers row is erased.	The application ends its execution showing an error indicating that no exam headers row was found.	Passed
TC7	An invalid format is provided for the constraints tab. A description row is erased.	The application ends execution showing an error indicating that it could not proceed parsing the constraints tab.	Passed
TC8	An invalid format is provided for the calendar tab. All data was moved one column to the right.	The application ends its execution showing an error indicating that it could not proceed parsing the calendar tab.	Passed
TC9	No exams were provided in a valid format exams tab.	The application ends its execution showing an error indicating that there were no exams.	Passed

Table 10.1: Test cases over invalid test coverage items

10.2.1 Test cases for invalid executions

In Table 10.1 every test case checks that the output files that show the log and the errors are updated at the right moment, that is, when reaching an stop point of the application. Furthermore the contents of such files are also checked.

10.2.2 Test cases for valid situation

Because these tests imply a detailed explanation of the input provided, this section will be structured differently and each test case will be described in a subsection. Each of the following test cases will contain the following details:

- **Description:** General explanation of the aim of the test case.
- **Input details:** Concrete inputs that relate to the input test coverage items that this test covers.
- **Procedure:** The steps to be followed in the test case.
- **Expected output:** The expected output in each of the steps of the test case.
- **First execution:** (Optional) In case the test detected an error in its first execution, such error detection will be described here.
- **Status:** The result obtained.

TC10

Description In this test case it is wanted to test all the possible wrong inputs that would lead to skip an exam in the exams tab of the Excel data file. Since we provide a list of exams as input we can cover all these test coverage items in a single test case.

Input details The following incorrect formed exams will be considered:

- Exam with no course.
- Exam with no course or seminar.
- Exam with no course, seminar or modality.
- Exam with no course, seminar, modality or duration.
- Exam with no course, seminar, modality, duration or id.
- Exam with a string as course.
- Exam with a string as seminar and duration.
- Exam with a repeated id of a previously well formed exam.

The input file also includes 38 valid format exams.

Procedure The application is executed with valid inputs but the specified ones. Once it reaches the first stopping point, a “0” is introduced to terminate the execution.

In conclusion, we follow *Path03*.

Expected output When reaching the first stopping point the application must show detailed errors both in “errorLog.txt” and in the console.

“inputUncoloredLog.txt” must show that 38 exams were generated.

First execution The log file showed a higher amount of generated exams even if such was impossible. Repeated id exams even if the error was shown were still parsed and used in the application.

Status Solved and passing.

TC11

Description In this test case it is wanted to test all the possible wrong inputs that would lead to skip a constraint in the constraints tab of the Excel data file. Since we provide a list of exams as input we can cover all these test coverage items in a single test case.

Input details This input includes:

- A single valid exam.
- A TD (Time Displacement) constraint with the first exam id missing.
- A DB (Day Banned) constraint with the date missing.
- A DB (Day Banned) constraint not specifying whether it is hard or not.

The input contains also nine constraints but all of them will be invalid since they will reference non-existent exam ids.

Procedure The application is executed with valid inputs but the specified ones. It directly proceeds to the second stopping point at the end of the constraint parsing. At that point, the execution will be canceled.

In conclusion, we follow *Path04*.

Expected output When reaching the second stopping point the application must show detailed errors both in “errorLog.txt” and in the console.

“inputUncoloredLog.txt” must show that one exams was generated, and that there were no constraints generated.

Status Passing.

TC12

Description In this test case we provided a well formed set of exams and we considered well formed constraints to test their right parsing. Additionally this test will cover the particular case of the weights configuration file.

Input details The input for this test case is the same as the one in TC11 in Section 10.2.2. However more well-formed exams, a total of 62 exams will be considered so that the constraints that were invalid due to non-existent exams ids are now valid.

Procedure The previous input will be provided to the application. The execution will proceed in every stop point until reaching the computing phase. When reaching it, we will wait until the application ends.

In conclusion, we follow *Path05*.

Expected output When reaching the first stopping point an error will be shown stating that a default value was considered for constraint weight “dd”. On the second stopping point there will be the same errors as in TC11 in Section 10.2.2.

Therefore “inputUncoloredLog.txt” must show that 62 exams were generated, and that nine constraints were generated. This nine constraints were the ones that could not be created in TC11 due to link to non-existent exam ids.

In the final computation, all the output files must have a right format. Furthermore, no exam is expected to be unscheduled.

Status Passing.

TC13

Description This case will present a normal scenario with no detected errors, but with an implicit error in the specified constraint. That is, two contradictory hard constraints will be considered.

Input details This input considers a set of exams containing just two exams A and B. Knowing that, the following input constraints are considered:

- A Same Day constraint over exams A and B. (Hard)
- A Different Day constraint over exams A and B. (Hard)

All other configuration options are set to a valid value.

Procedure The application will be executed with the provided input, which results in path *Path06*. At the end of the computation we must check that one of both exams must be unscheduled, whereas the other is scheduled.

Expected output The application will not pause at any of the stopping point, it will compute the scheduling and end the execution.

Therefore "inputUncoloredLog.txt" must show that 2 exams were generated, and that 2 constraints were generated.

It the output is expected that either A or B is unscheduled, just one of them. The other must be scheduled.

Status Passing.

10.3 Traceability of the Test Cases

The traceability can be seen in Table 10.2. In such table the identifiers used along the design are used used. For the sake of clarity some special syntax will be used. It is the following:

- **Identifiers with ".X".** This represents all the items below in the hierarchy. For example: 2.3.X, is equivalent to: 2.3.1, 2.3.2

Even if not the test aim, some items of the test design can appear in a test. For instance, while testing that constraints work properly, the calendar must be on a valid format, so must be the exams tab, etc. Enumerating everything in every test would be messy, so instead some variables will be considered:

- **conf_default.** 1.1.X, 1.2
- **exam_default.** 2.3.2, 2.1.1.X
- **constraints_default** 3.1.1.X, 3.1.2.X
- **calendar_default.** 4

Furthermore, the key points of every test case will be in **bold**.

ID	Test case Traceability
TC1	**Path01 - 1.1.1, 1.3.1** - conf_default, exam_default, constraints_default, calendar_default
TC2	**Path01 - 1.1.2, 1.3.2** - conf_default, exam_default, constraints_default, calendar_default
TC3	**Path01 - 1.1.4, 1.3.3** - conf_default, exam_default, constraints_default, calendar_default
TC4	**Path01 - 1.1.1, 1.3.4** - conf_default, exam_default, constraints_default, calendar_default
TC5	**Path01 - 1.1.3, 1.3.1** - conf_default, exam_default, constraints_default, calendar_default
TC6	**Path01 - 2.2** - conf_default, constraints_default, calendar_default
TC7	**Path02 - 3.2** - conf_default, exam_default, calendar_default
TC8	**Path01 - 4.2** - conf_default, exam_default, constraints_default
TC9	**Path01 - 2.3.1** - conf_default, exam_default, constraints_default, calendar_default
TC10	**Path03 - 2.1.2.X, 2.1.3.X, 2.1.4** - conf_default, **exam_default**, constraints_default, calendar_default
TC11	**Path04 - 3.1.3.X** - conf_default, exam_default, **constraints_default**, calendar_default
TC12	**Path05 - 5.1.1, 3.1.3.X** - conf_default, **exam_default**, constraints_default, calendar_default
TC13	**Path06 - 5.1.2, 3.1.4** - conf_default, exam_default, constraints_default, calendar_default

Table 10.2: Traceability of the invalid test cases.

Chapter 11

Experimental Study

In order to test the provided solution prototype, as well as to find out an optimal configuration for such prototype several experiments were performed.

Only two real life exam schedules were provided by the client, and these did not contain any data about user constraints. It was needed to define some scenarios simulating expected instances based on the provided ones.

All the experiments were run on a Linux cluster (Intel Xeon 2.26 GHz. 128 GB RAM). The Java version installed in such machine is “java 17.0.2 2022-01-18 LTS Java(TM) SE Runtime Environment (build 17.0.2+8-LTS-86) Java HotSpot(TM) 64-Bit Server VM (build 17.0.2+8-LTS-86, mixed mode, sharing)”

11.1 Test Instances

As commented above, the School of Computer Science provided two real exam schedules used for the May and June 2021 calls. Just two instances were considered not enough to test the proposed solution, therefore it was necessary to create real-life shape instances for testing purposes. Probabilities are used to randomly generate instances, but also to ensure that they look similar to the real-life instances initially provided.

The two real-life instances were studied to compute the following data:

1. Average number of exams in total.
2. Average number of exams per course.
3. Number of elective subjects.
4. Average number of rounds.
5. Round sizes relative frequency.
6. Exam duration relative frequency.
7. Average number of days for the calendar of exams.
8. Deliveries relative frequency.

There are many functionalities included in the proposed solution that were not considered before, or if so, were not written down on the exam schedule file. For example, constraints involving an exam not being able to take place on a specific date.

The following functionalities had no predencent and their impact in the test instances was based on conversations with the client:

- Number of constraints.
- Extra times.
- Numerical complexity.

On top of that, it was also necessary to consider the fact that the user can specify both soft and hard constraints. Because the proportion of hard user constraints against soft user constraints is unknown, it was assumed that a user constraint has a probability of 0.3 of being hard. This value cannot greatly increase because it would likely lead to unsolvable instances.

Even though the tool generates good instances, and it was built to prevent straightforward incompatibilities such as: ”Exams A and B must be scheduled on the same day” and ”Exams A and B must be scheduled on different days”, there were also other circumstances that prevented the generated problem instance from having a solution. This required manually checking over a generated batch of instances in each scenario.

11.1.1 Test instances scenarios

Generated instances are grouped in three different scenarios, each of them containing ten instances.

All scenarios consider a total number of 74 exams and can have rounds. User constraints have a 0.3 of being hard constraints, and all the exam durations observed in the real instances have configured a probability based on their relative frequency. It was assigned a probability of 0.15 of an exam being a delivery based on the previously observed relative frequency in the real instances. The scenarios differ in the number of constraints and the enabled functionalities.

The considered scenarios are the following:

1. Expected Scenario: Around 60 constraints were considered. Numerical complexity and extra time features were disabled.
2. Hard Scenario: Around 90 constraints were considered. Numerical complexity and extra time features were disabled.
3. Complex Scenario: Around 100 constraints. Numerical complexity and extra time were enabled.

11.2 Algorithm Tuning

This section details the experiments done to tune the parameters of the algorithm. Two phases were carried out since testing all the combinations that would result as the mix of all the parameters to tune would not be viable.

Some experiments were done to obtain the pool of testing values to be considered in the following sections. In such experiments it was concluded that the elitism of the algorithm should be the 4% of the population, meaning that the 4% percent best individuals should go directly to the new generation. A lower value leaded to the algorithm not converging properly, while a greater one leaded to premature convergence.

11.3 Genetic Algorithm Tuning

This section is devoted to the experiments to tune the parameters of the genetic algorithm being such and their respective values the following:

- **Population size:** [100, 300, 500]
- **Mutation Probability:** [0.05, 0.1, 0.15, 0.2]
- **Crossover Probability:** [0.8, 0.9, 1.0]
- **Repairing Algorithm Maximum Depth:** [2, 3]

All possible combinations of the above values were executed over the expected scenario previously commented. This results in a total of 72 combinations to be run over the expected scenario. As this is formed by ten instances, the 72 combinations will be tested ten times per instance, resulting in a total of 7200 executions. It is executed ten times per instance because Genetic Algorithms are stochastic, and the common procedure to lead with this type of algorithms is to run them multiple times and find the average values. Therefore, during each of the ten iterations of each instance the fitness value will be recorded, so that at the end the average of all those values can be computed. The configured number of generations is 1500, this value is sufficient to make the algorithm converge in the majority of the cases, it was also obtained in some preliminary experiments.

As it will be messy to show all the results in a single plot, executions were grouped by population sizes. This also implies that there should be 24 configurations in a single plot, for the sake of clarity, even if within same picture, they are represented in two different graphs.

The results of this tuning experiments can be seen in Figure 11.1 which represents the results with a population size of 100, Figure 11.2 which represents the results with a population size of 300, and finally Figure 11.3 which represents the results with a population size of 500. All three plots have been plotted with the same axis scale. This was done because of two reasons. Firstly to represent visually the time difference between the different population size, and secondly to see clearly the convergence plots.

The algorithm was configured to run 1500 generations in each execution. This value was chosen by examining in some preliminary examples the convergence of the algorithm. In such examples it was observed that the algorithm tends to stabilize both its best and average fitness at that number of generations.

As can be seen in the plots, configurations with a population size of 500 are the ones with a smoother curve. Furthermore we can see that the higher the population size the better fitness we achieve. A fitness of less than 600 is obtained with population size of 500, while values around 700 are obtained with a population size of 100. Even though the results may imply that a bigger population size would turn out to be positive for the algorithm, this was discarded because of time issues. As can be seen in the plots, the higher the population size the longer it takes for the algorithm to end the execution. It was observed that running the algorithm twice or more times tends to be more efficient than increasing the population size.

The same results are also shown in Figures 11.4, 11.5 and 11.6 where they are compared against the number of generations. These plots were included to show in the same scale how smoother are the configurations with population size of 500 compared with the others, and that these configurations tend to converge better towards a same fitness, while the others provide a wider variety of results.

Having said that, by observing the results we can conclude that the best configuration is:

- **Population size:** 500
- **Mutation Probability:** 0.2
- **Crossover Probability:** 0.9
- **Repairing Algorithm Maximum Depth:** 3

11.4 Fitness Function Tuning

The provided implementation allows users to configure the coefficients of the fitness function. Deciding which should be the default configuration was done by running a set of experiments considering the following combinations:

- **User constraints:** [80.0, 110.0, 140.0][1]
- **Resting Interval Penalization:** [2.5, 3.0, 3.5]
- **Numerical Complexity Penalization:** [2.0, 2.5, 3.0]

Because we are changing these coefficients, the fitness is not a reliable metric to compare the output of this configurations. Thus, in this set of experiments quality metrics that are used to compute the fitness function will be the references, but not the fitness function itself.

The metrics were commented is Section 5.3.6, but for the purpose of this phase, numerical complexity will not be considered. The user made it clear that the number of user constraints and the amount of free time in the resting interval were far more relevant.

Having said that, all possible combinations of the above list were considered and each of them were run ten times, so that it was then possible to compute the average values. In total 27 configurations were tested ten times, which adds up to a total of 270 executions over the complex scenario. The following values were computed and were used to select the configuration:

- **Number of unscheduled exams.**
- **Number of unfulfilled user constraints.**
- **Number of minutes occupied in the resting interval.**

[1] Note that this value will be individually assigned to all types of user constraints.

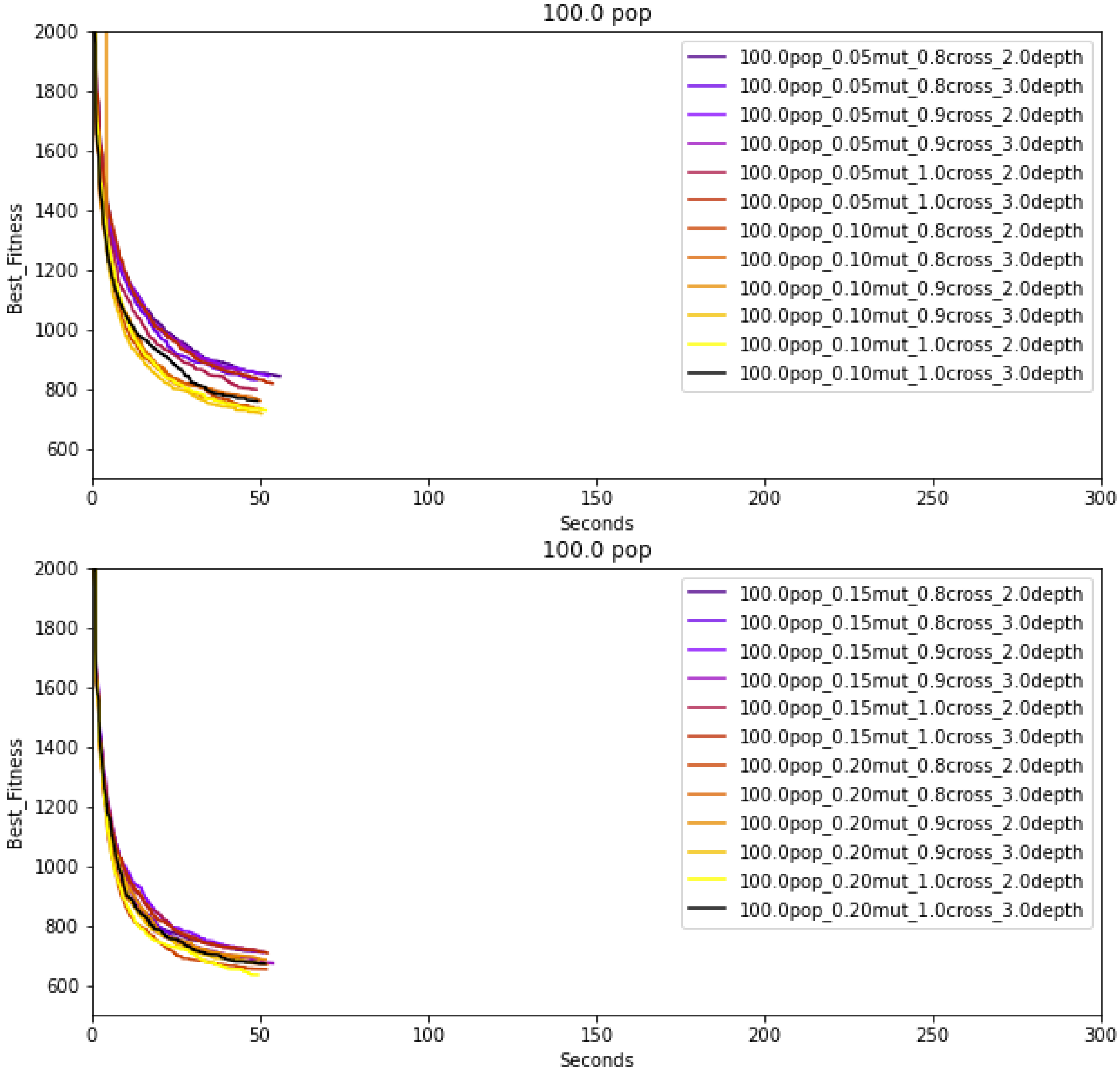


Figure 11.1: Genetic Parameters tuning - 100 population configurations - Against time.

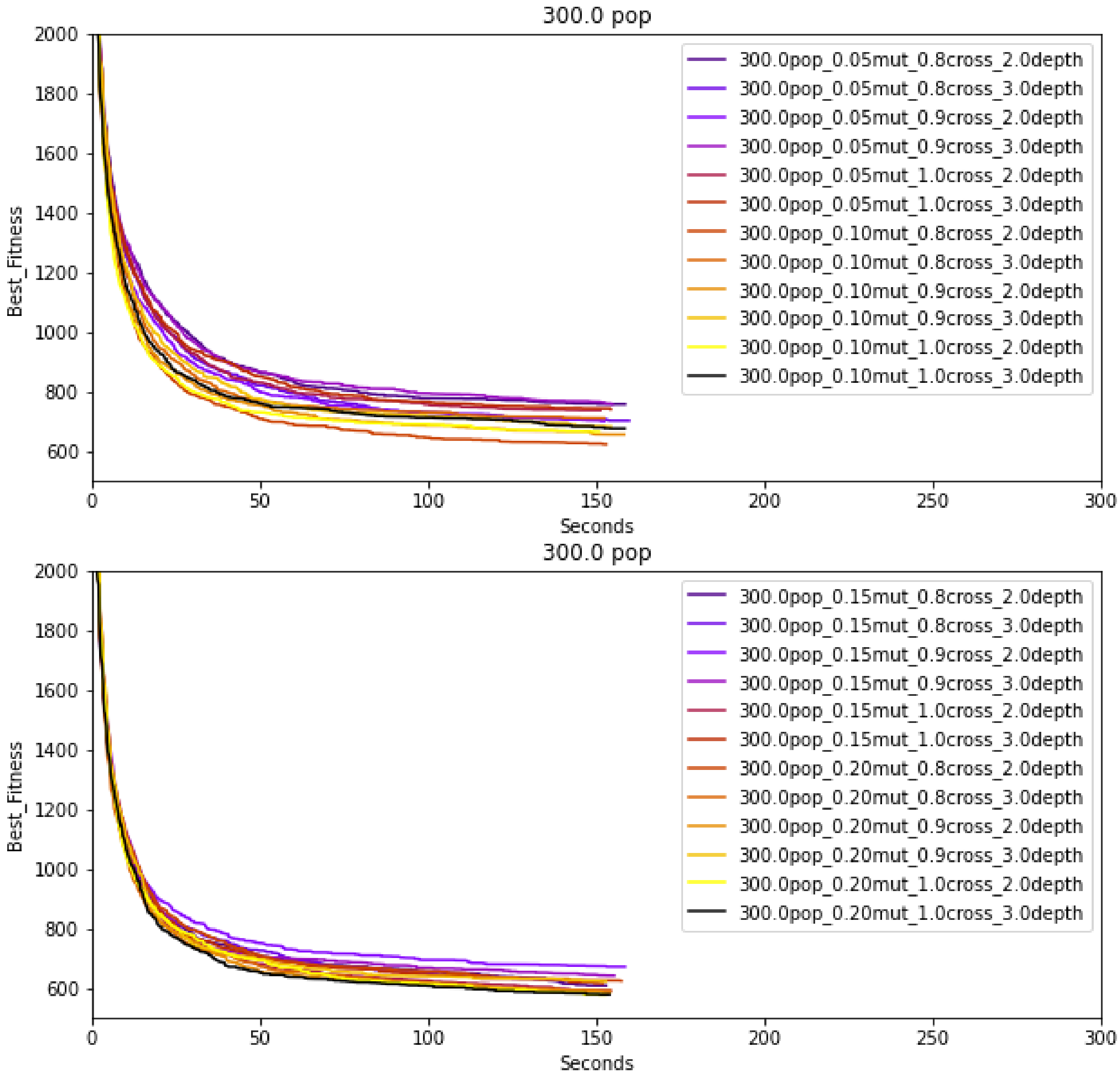


Figure 11.2: Genetic Parameters tuning - 300 population configurations - Against time.

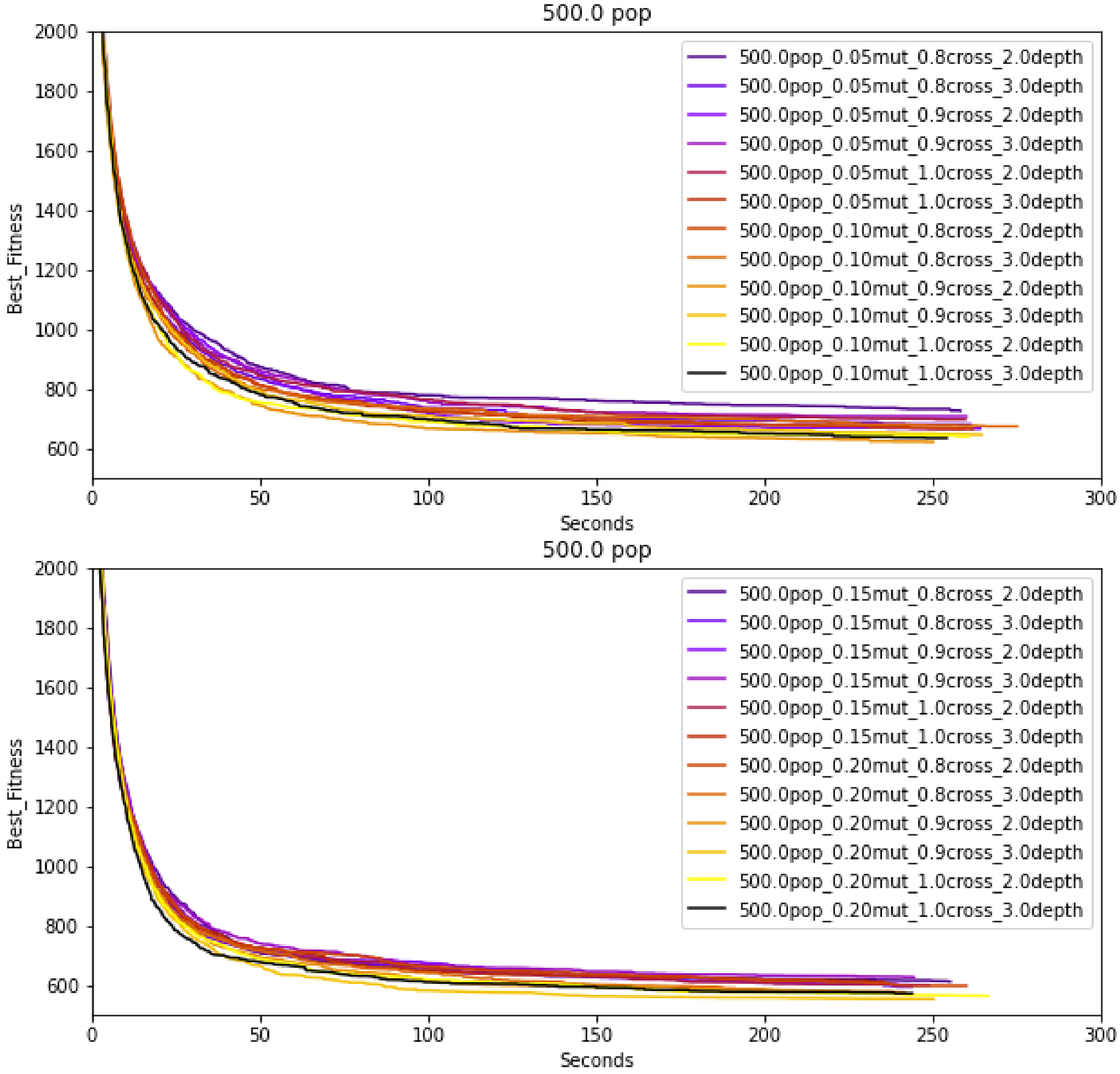


Figure 11.3: Genetic Parameters tuning - 500 population configurations - Against time.

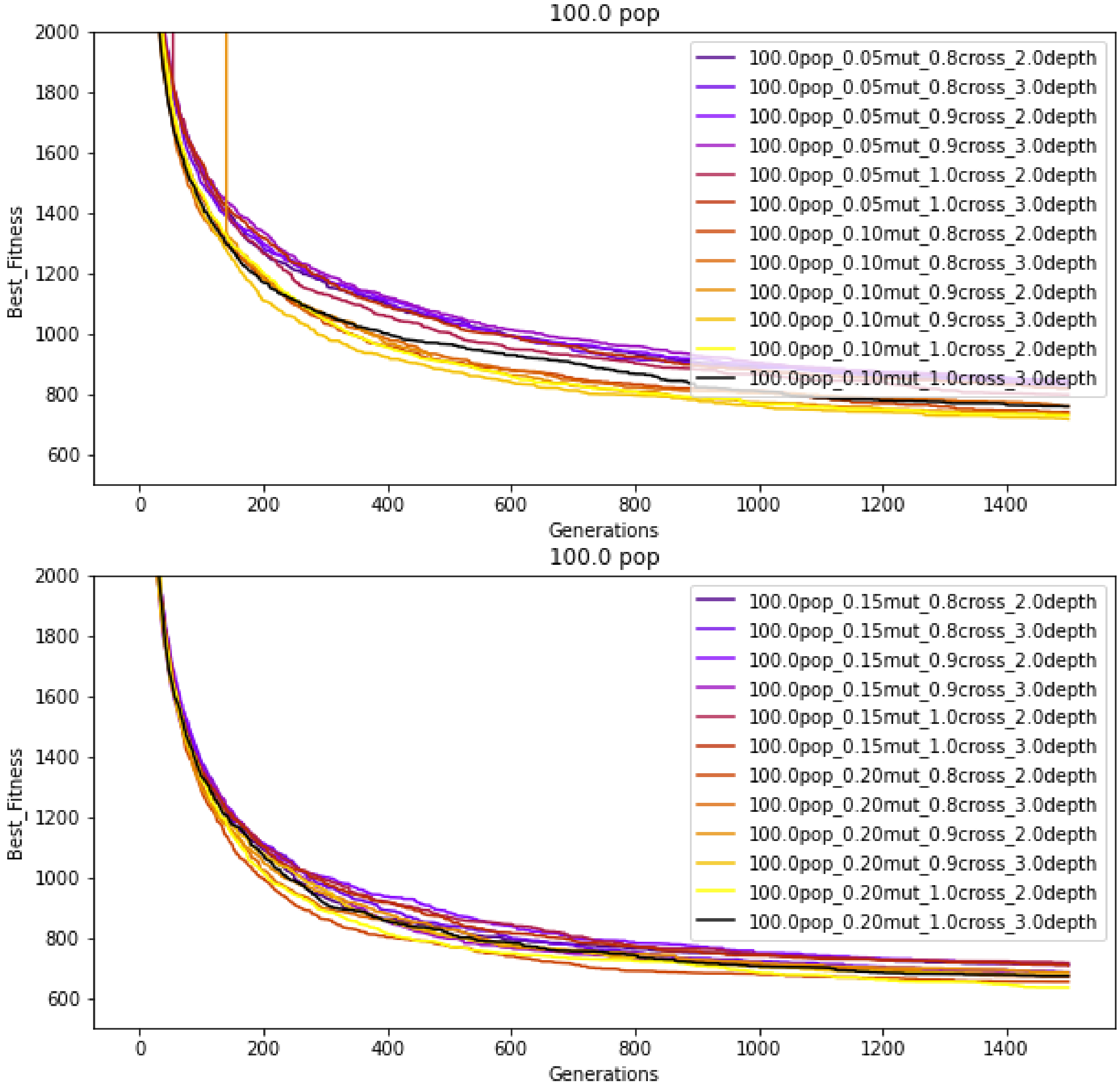


Figure 11.4: Genetic Parameters tuning - 100 population configurations - Against generations.

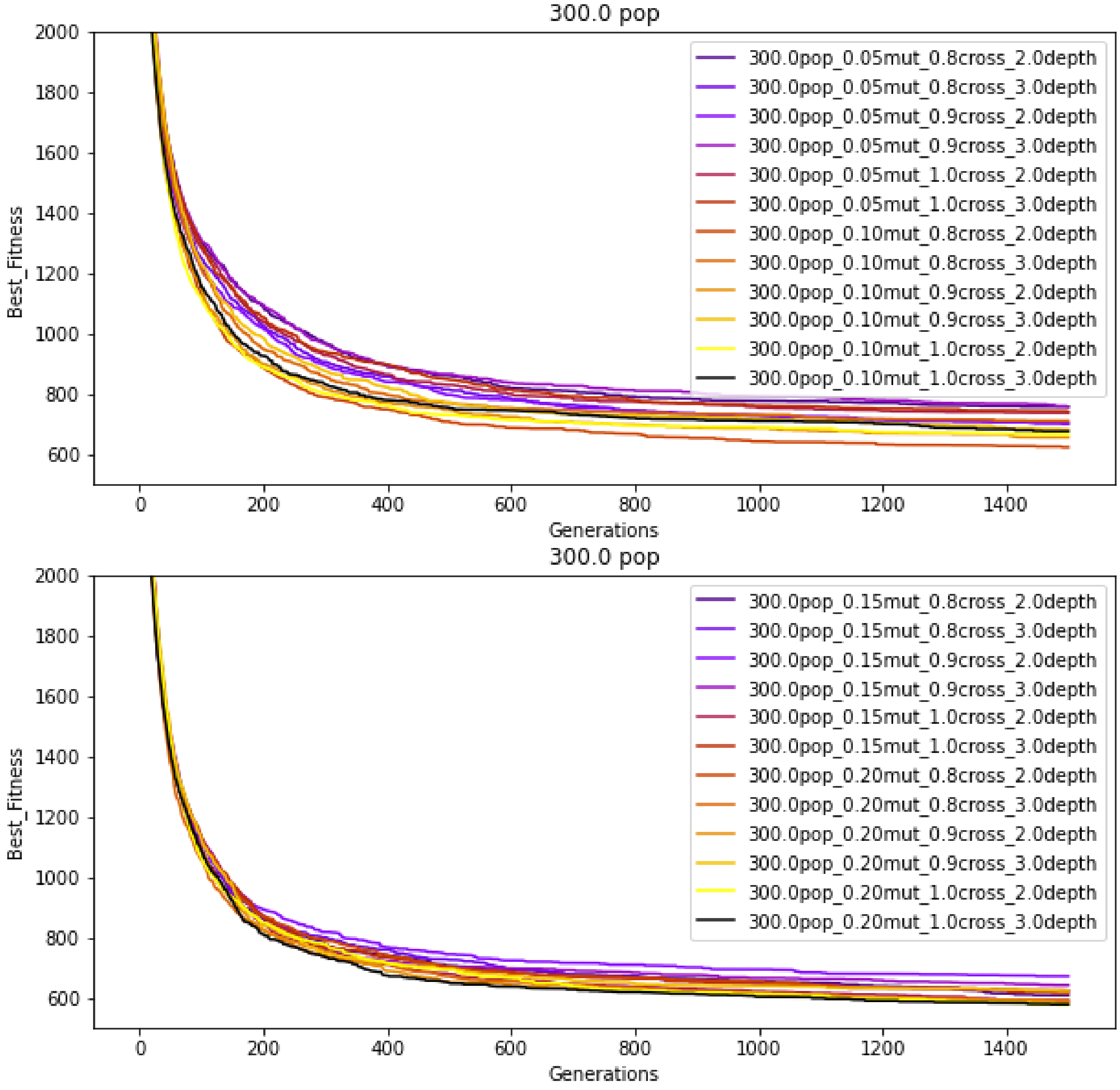


Figure 11.5: Genetic Parameters tuning - 300 population configurations - Against generations.

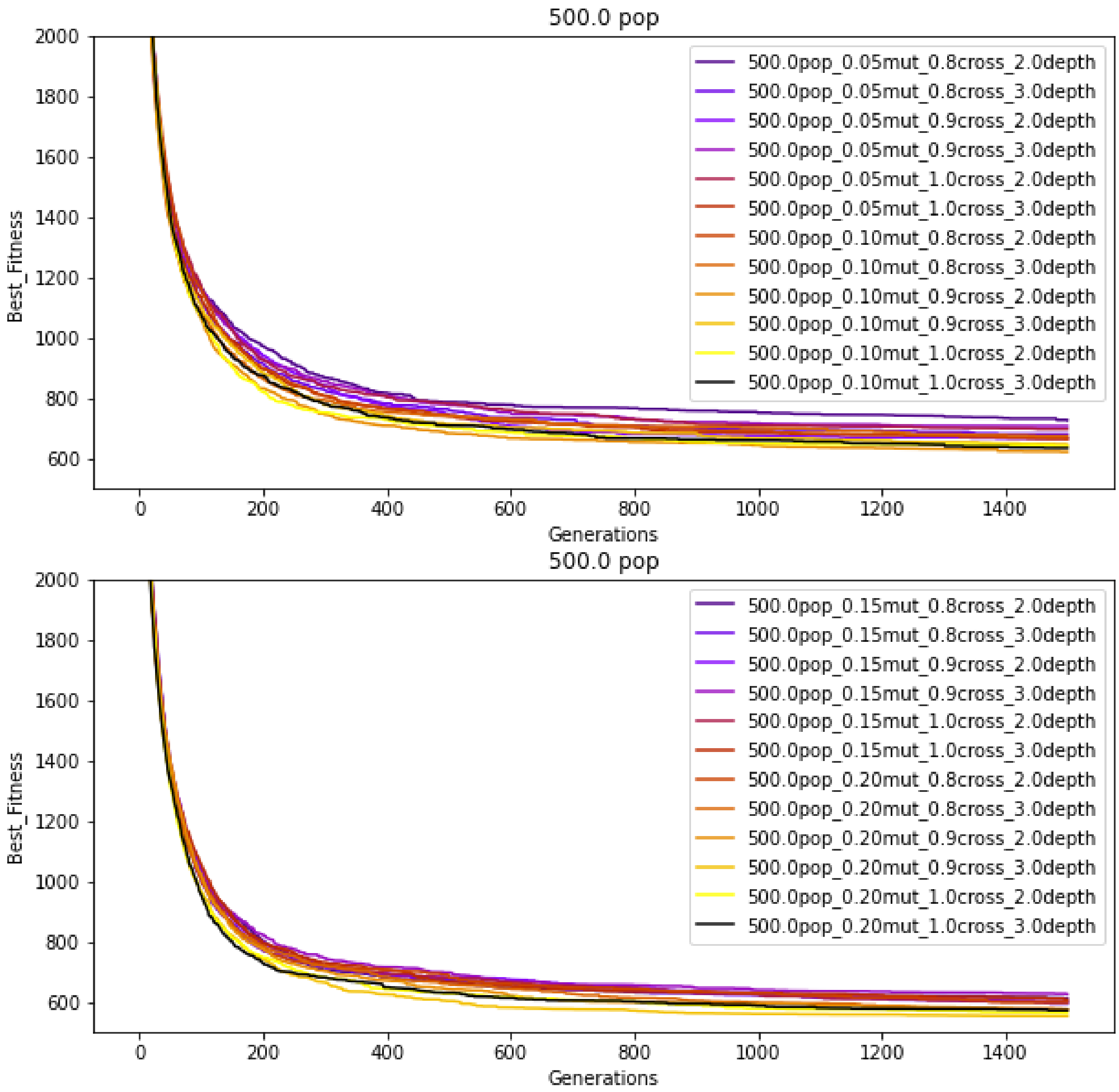


Figure 11.6: Genetic Parameters tuning - 500 population configurations - Against generations.

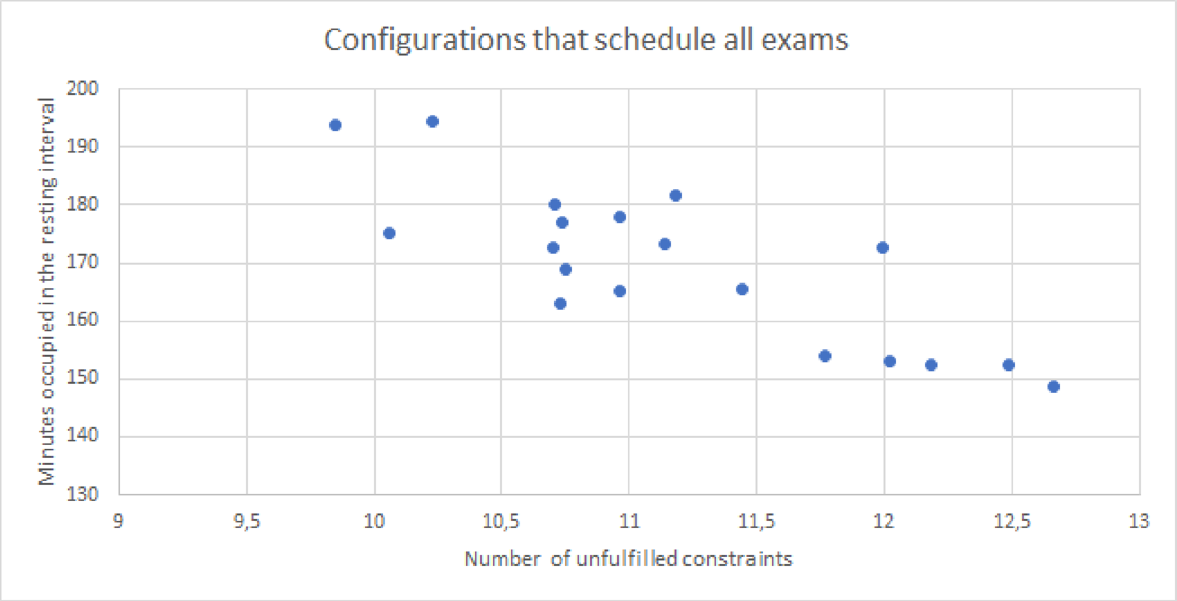


Figure 11.7: Fitness coefficients experiments.

In addition to the above, relative metrics of unfulfilled user constraints against the total number number of user constraints, and the total number of constraints were computed. In this context, they do not add additional information, since all configurations are run over the same set of instances and thus, over the same set of user constraints. They were computed as they could make the analysis clearer.

To run this set of experiments the chosen configuration of genetic parameters was the one obtained in the previous Section 11.3. The results can be found in Table 11.1, at the end of the section. Values are sorted by columns prioritizing the order previously commented.

As can be seen, the algorithm nearly always finds a solution to the problem, but there are some configurations with which in some of the executions, a solution was not found.[2] These configurations were instantly discarded.

On the rest of the results we can see that there is a soft inverse dependency among the *Number of unfulfilled user constraints* and *Number of minutes occupied in the resting interval*. The non-discarded configurations can be seen in Figure 11.7, where they are represented with the average number of unfulfilled constraints in the x axis, and the average number of minutes occupied in the resting interval in the y axis.

Since we want to provide a default and average value a Pareto Front was computed as can be seen in Figure 11.8, in which the configurations marked in green in Table 11.1 are represented. Finally a configuration in the middle of the frontier was chosen. It is indicated in yellow in Table 11.1, and it which was:

- **User constraints:** 140.0
- **Resting Interval Penalization:** 3.5
- **Numerical Complexity Penalization:** 2.5

[2] This implies that not all the exams were scheduled.

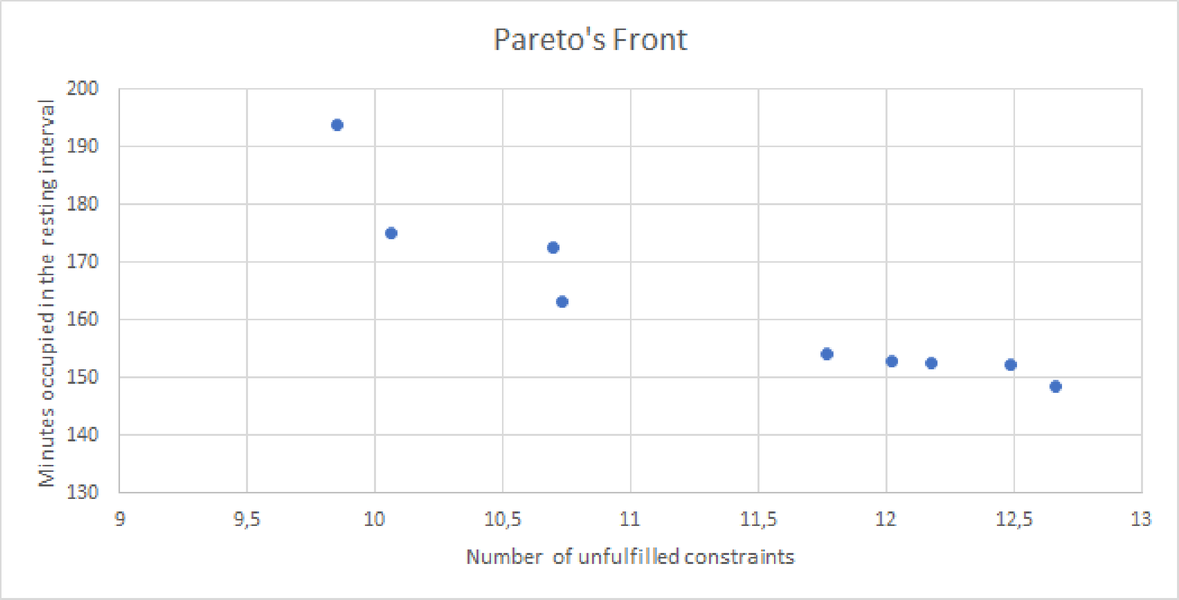


Figure 11.8: Fitness coefficients experiments - Pareto Front.

Configuration	Unscheduled Exams	Unfulfilled_Constraints	UC / SUsC	UC / UsC	Minutes_Resting_Interval
140.0uc_2.50pi_2.0nc	0	9,85	0,1441	0,1047	193,95
140.0uc_3.00pi_2.0nc	0	10,06	0,147	0,1074	175,15
140.0uc_3.00pi_2.5nc	0	10,23	0,1492	0,1088	194,5
140.0uc_3.50pi_2.0nc	0	10,7	0,1565	0,1135	172,65
140.0uc_3.50pi_3.0nc	0	10,71	0,156	0,1137	180,15
140.0uc_3.50pi_2.5nc	0	10,73	0,1572	0,1143	163,15
110.0uc_2.50pi_2.0nc	0	10,74	0,1567	0,1143	177
110.0uc_3.00pi_2.0nc	0	10,75	0,1572	0,1142	169
110.0uc_3.50pi_2.0nc	0	10,96	0,1597	0,1164	165,3
110.0uc_3.00pi_3.0nc	0	10,96	0,16	0,1167	178,05
110.0uc_3.00pi_2.5nc	0	11,14	0,1628	0,1185	173,45
110.0uc_2.50pi_3.0nc	0	11,18	0,1633	0,1191	181,85
110.0uc_3.50pi_3.0nc	0	11,44	0,1674	0,1219	165,6
80.0uc_3.00pi_2.0nc	0	11,77	0,1723	0,1257	153,95
80.0uc_2.50pi_3.0nc	0	11,99	0,1756	0,1281	172,75
80.0uc_3.00pi_2.5nc	0	12,02	0,1757	0,1284	152,9
80.0uc_3.50pi_2.5nc	0	12,18	0,178	0,13	152,4
80.0uc_3.50pi_2.0nc	0	12,49	0,1829	0,1332	152,3
80.0uc_3.50pi_3.0nc	0	12,66	0,1864	0,1358	148,55
140.0uc_2.50pi_2.5nc	0,01	10,16	0,1486	0,1081	192,8
140.0uc_2.50pi_3.0nc	0,01	10,24	0,1495	0,1086	198,6
140.0uc_3.00pi_3.0nc	0,01	10,42	0,1521	0,1109	187,6
110.0uc_2.50pi_2.5nc	0,01	10,67	0,1559	0,1139	184,45
80.0uc_2.50pi_2.0nc	0,01	11,22	0,1638	0,1197	156,15
110.0uc_3.50pi_2.5nc	0,01	11,55	0,1687	0,1229	163,6
80.0uc_2.50pi_2.5nc	0,01	11,7	0,1717	0,1247	162,25
80.0uc_3.00pi_3.0nc	0,01	12,23	0,179	0,1303	158,75

Table 11.1: Fitness Coefficients experiments results

11.5 Comparative Experimental Study

The aim of this section is to assess the effectiveness of the proposed solution with respect to other simpler alternatives:

- **Greedy Algorithm.** This implies that a set of solutions is randomly generated and the best one is chosen.
- **Greedy Algorithm with repairing procedure.** In addition to the above the greedy algorithm is now able to use the repairing procedure which should prevent it from leaving as many unscheduled exams.
- **Genetic Algorithm:** The whole implementation provided, which has an evolution process.

All the stated possibilities will be run in the three scenarios: expected, hard and complex. As done in previous sections, each instance within a scenario will be run ten times, which imply that 100 executions will be done over each scenario.

11.5.1 Greedy algorithm

Initially, in Section 4.1 it was mentioned that this problem could be solved randomly, but that this was complicated due to the constraints that users can state.

To prove such intuition, a random population of individuals will be generated and without any evolution process it will be directly evaluated by the Greedy Algorithm. This population will have a size of 500 individuals, which was designed as the default value for the genetic algorithm in Section 11.3.

In this set of experiments, there will not be any repairing process. Because of that it is expected that this approach struggles to schedule all the exams despite that typically they can all easily fit in the calendar leaving a reasonable amount of free time.

Because the problem size is not that big, a decent solution can be found provided a good initial population size. In this particular context this is expected to prevent from seeing a lot of unscheduled exams in the statistics.

The results can be seen in Table 11.2. Some exams were not scheduled, but as commented before, the number is not that high. As we can see in the statistics around a 35% of the soft user constraints were not fulfilled for all the scenarios. Note that the number of constraints in the three scenarios is different but still the algorithm tends to leave unfulfilled the same percentage of constraints. This leads us to conclude that a random approach could be expected to not fulfill at least 30% of the user constraints.

On the other hand, because of its simplicity, this algorithm takes negligible time to execute.

11.5.2 Greedy algorithm with repairing procedure

One of the problems of the previous approach was that once an exam is placed there is no way to backtrack that decision. This can easily lead to dead ends in which exams cannot be placed

	complex	hard	expected
Unscheduled Exams	1.79	1.69	0.49
Unfulfilled_Constraints	23.08	25.17	12.91
UC / SUsC	0.34	0.36	0.36
UC / UsC	0.24	0.26	0.25
Minutes_Resting_Interval	1033.5	921.3	708.6

Table 11.2: Average statistic values of the Greedy Algorithm with no repairing in the scenarios.

	complex	hard	expected
Unscheduled Exams	1.65	1.46	0.39
Unfulfilled_Constraints	23.54	25.02	12.88
UC / SUC	0.34	0.36	0.36
UC / UC	0.25	0.26	0.25
Minutes_Resting_Interval	1028.95	907.5	700.8

Table 11.3: Average statistic values of the Greedy Algorithm with repairing in the scenarios.

because the set of viable days in which they can be placed according to the hard constraints linked to them is fully occupied.

Introducing the repairing algorithm with the maximum repair depth of 3, which was the default value determined in Section 11.3, is expected to decrease the amount of unscheduled exams. There are no more expected changes with respect to the previous approach since the repairing algorithm only comes into place when an exam cannot be scheduled.

The results can be seen in Table 11.3. As expected, the number of unscheduled exams decreased and there was not a relevant difference in the other metrics. A better result could have been obtained increasing the maximum depth of the repairing algorithm, but the aim of this comparative study was to see how each part of the algorithm helps reaching better solutions, and because of that parameters values are left as default.

As the one in the previous section, this algorithm takes negligible time to execute.

11.5.3 Genetic algorithm

Finally, the full implementation of the proposed solution will be tested. It is expected to have no unscheduled exams and to decrease considerably the number of unfulfilled constraints. Moreover, by having the evolution process an individual with a permutation which minimizes the number of minutes occupied in the resting interval is easier to find.

Results of this set of experiments can be seen in Table 11.4. As expected, there was no unscheduled exam in any of the iterations over the instances. Furthermore the number of unfulfilled constraints was halved in the complex and hard scenarios, while reduced to one fourth in the expected one. Because of the evolution process, the best individual slightly has more and more constraints fulfilled along the generations of the Genetic Algorithm.

The minutes on the resting interval were reduced to a 10% of the values provided in the previous experiments. This soft constraint is hard to fulfil by generating random individuals, it is clear that an evolution process in which different combinations are tested slightly leads in a similar way as it does with user constraints to a better solution.

	complex	hard	expected
Unscheduled Exams	0.0	0.0	0.0
Unfulfilled_Constraints	10.61	11.28	3.71
UC / SUC	0.15	0.16	0.10
UC / UC	0.11	0.11	0.07
Minutes_Resting_Interval	177.3	141.6	54.3

Table 11.4: Average statistic values of the Full Algorithm in the scenarios.

However, such evolution process takes time to be computed. Around three or four minutes are needed to provide a solution. Therefore, this is the slowest algorithm but, according to the results, this is also the one that reaches the best solutions.

11.5.4 Greedy algorithm vs genetic algorithm

Previous sections have shown the plain results obtained by the algorithms considered, and these results point out the Genetic Algorithm as the best approach. This section further compares the Genetic Algorithm and the Greedy Algorithm, to show the performance gains it brings over the simplest approach.

Three metrics of the five considered previously are used.

- **Number of Unfulfilled Exams.**
- **Number of Unfulfilled user Constraints.**
- **Number of Minutes occupied in the Resting Interval.**

Both algorithms will be compared by means of scatter plots considering all instances in all scenarios. A plot will be shown for each of the stated metrics, with the exception of the number of unfulfilled exams. Because the Genetic Algorithm obtains a value of 0 for all instances in the metric, the scatter plot would result in all points being on one axis.

Both in Figure 11.9 and 11.10 we can observe clearly how the Genetic Algorithm achieved better results than the Greedy Algorithm, as all the values are far below the main diagonal. The represented values are the mean values considering ten execution for each instance.

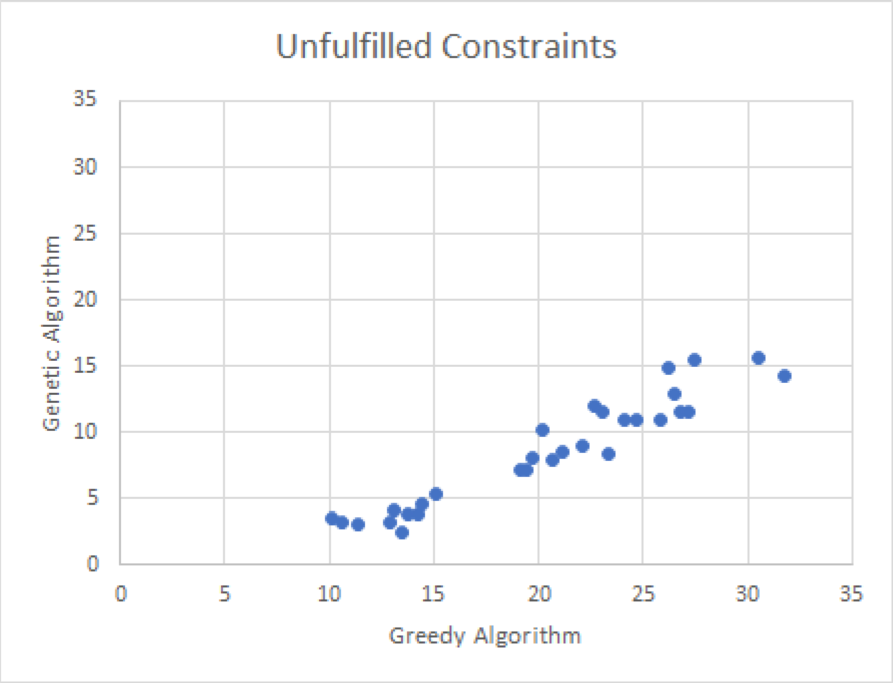


Figure 11.9: Number of Unfulfilled Constraints - Scatter Plot Comparison.

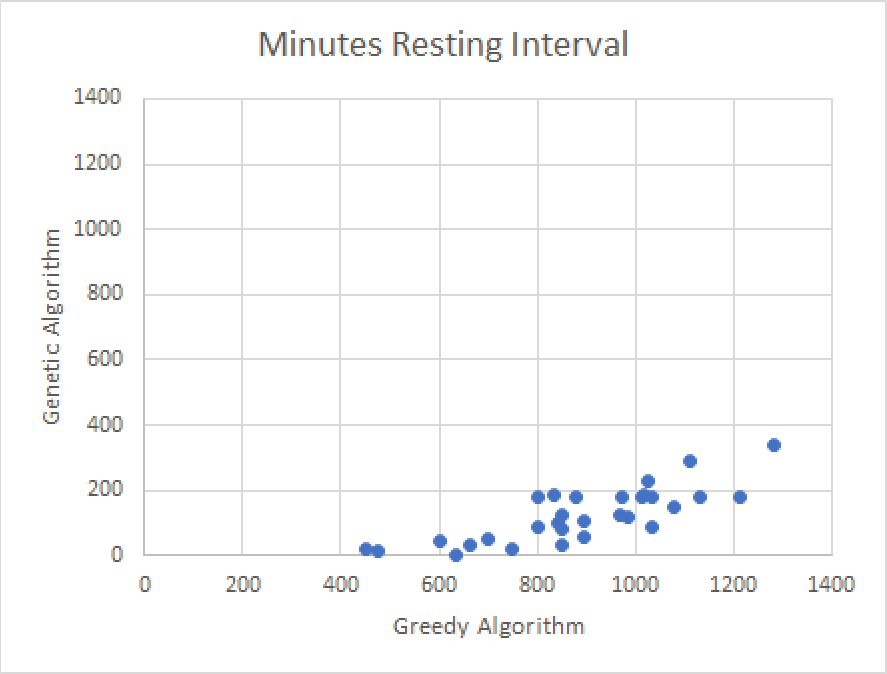


Figure 11.10: Number of Minutes occupied in the Resting Interval - Scatter Plot Comparison.

Chapter 12

User Manual

12.1 System Requirements

The prototype requires *Java JDK 13.0.1* or a newer version to be executed. Previous Java versions may show incompatibilities, because they may not have all the necessary features. An example of such features are the classes used for dates and times: *LocalTime* and *LocalDate*.

12.1.1 Java installation advice

Some years ago Java published its software in two different ways: Java Runtime Enviroments (JREs) and Java Development Kits (JDKs). Basically, JREs provided just the execution enviroment, whereas JDKs also provided tools to develop software, such as a compiler.

Nowadays, JREs are not used anymore, in fact the last available Java version for them was 8. To install newer versions of Java, it is necessary to install the corresponding JDK.

12.2 Execution Instructions

The developed prototype is encapsulated in an executable .jar file called *examScheduler.jar*. This executable receives two arguments: the first one is the path to the properties file which contains the filepath to all other necessary files. Additionally, the user can specify as a second argument a name for the statistics file, which will create a file with quality factors of each provided solution as explained in Section 12.3.1.

To launch the program, the system must have configured Java 13, at least SDK 13.0.1. Once that is provided, launching the program consists in executing the following command:

```
$ java -jar examScheduler.jar <pathToFilesFile> <nameOfOutputFile>
        <nameOfTheStaticticsFile>
```

Although any name or path can be used for the configuration files they must exist and be accessible by the program. The first refers to the *filesFile* which is detailed in Section 12.2.1. The second parameter is optional and it generates a file with some quality metrics of each generated solution.

12.2.1 Configuration files

There are many configurations that can be applied to the algorithm, so for the sake of simplicity similar configurations are grouped in the same file.

Filepaths file

This file has the references to all the other files, so that the users can place them wherever they want and with the name they prefer. It is referred as *filesFile* in this manual. The configurations inside it are the paths to all the other files:

- **inputFile:** Path to the input Excel data file.
- **outputBaseDirectory:** Path to the output directory in which the output must be placed. (It must end with "/". Ex: folder1/folder2/).
- **statisticsBaseDirectory:** Path to the output statistics directory in which the statistics file must be placed. (It must end with "/". Ex: folder1/folder2/).
- **excelConfiguration:** Path to the Excel configuration file detailed in Section 12.2.1.
- **dateTimes:** Path to the date and time configuration file detailed in Section 12.2.1.
- **weigthts:** Path to the weights configuration file detailed in Section 12.2.1.
- **geneticParameters:** Path to the genetic configuration file detailed in Section 12.2.1.

Date and time configuration

This file is referred to as *dateTime* in this manual. It contains configurations related to the time scheduling. The first one is the *default extra time.* An extra time at the end of each exam can be applied from the input file. For those exams that do not have configured an extra time it is possible to specify a default one.

- **defaultExtraTimeEnabled:** True implies that the default extra time functionality is enabled, False that it is not.
- **defaultExtraTimeMinutes:** This is the default extra time expressed in minutes.

The second feature to be managed in this file is the resting interval. That is, an interval in which exams cannot start and in which is desirable not to have exams.

- **beginningRestingIntervalHour:** This is the starting hour at which the resting interval starts. Must be in 24h format.
- **endRestingIntervalHour:** This is the ending hour at which the resting interval starts. Must be in 24h format.

The final configuration is related to the "deliveries". As previously stated in Section 4.2, there is a particular type of exams that can be scheduled at the same time as the other exams.

- **deliveryCollisionEnabled:** True implies that this particular type of exams work as the other ones, False that they can be overlapped with other exams.
- **deliveryIdentifier:** This is the identifier that it is expected to appear in the "modality" field of the exams in the input data file to mark the exams that are considered deliveries.

Genetic parameters

This file contains the configurations of the genetic algorithm. It is referred to as *geneticParameters.*

In this file there are two groups of configuration options, being the first one the ones over which the experimental study has been done, and being the second one the ones that the user is expected to change.

The first group includes the following options:

- **populationSize:** This states the size of the population that the algorithm must handle.
- **mutationProb:** This states the probability of mutating a new generated individual.
- **crossingProb:** This states the probability to keep a randomly selected pair of parents for crossover.
- **generations:** This states the number of generations that the algorithm must perform.
- **repairingDepth:** This states the maximum depth of the tree of changes of the repairing algorithm, which was explained in Section 5.2.1.

Although configurations provided in the first group will be assigned default values whose efficacy have been checked in the experimental study, the user is free to change them.

The second group are configurations which are likely to be changed by the user. These are:

- **algorithmRepetitions:** Number of times that the algorithm will be executed. The specified number in *maxSchedulesToTake* best fit individuals will be included in the population of following algorithm repetitions.
- **maxSchedulesToTake:** Number of different output schedules to be provided by the algorithm.
- **inputWarningsStop:** If set to true the application will pause when detecting parsing errors and warn the user. If set to false the execution will proceed if possible.
- **loggingFreq:** Amount of generations until writing into the log file the actual status of the population.

Weights

This file contains the the coefficients to be used in the fitness function, as detailed in Section 5.3.6. User constraints types are listed individually by their ID:

- **DB:** Day Banned. A day in which an specific exam cannot be placed.

- **SD:** Same Day. Two exams must take place on the same day.
- **DD:** Different Day. Two exams must take place on different days.
- **OE:** Ordered Exams. Two exams must be in an specific order, one after the other. This applies to days. The exams cannot be the same day.
- **TD:** Time Displacement. Two exams must be in an specific order, and with a specified minimum time distance between them. This applies to calendar days. The exams cannot be the same day.
- **DI:** Day Interval. An exam must be placed between two specified dates.

Additionally the weights of the soft constraints used to measure the quality of the solutions as stated in Section 5.3.6:

- **RIP:** Resting Interval Penalization. This is applied over the occupied minutes of the resting interval.
- **NCP:** Numerical Complexity Penalization. This is applied over the numerical complexity function detailed in Section 5.3.6.

Finally a high weight is assigned to unscheduled exams.

- **UE:** Unscheduled Exams. This should have a high value so that the algorithm can obtain high quality solutions.
- **SCDD:** Same Course Different Day. If a day have multiple exams of the same course it is penalized.

Excel configuration

This file encloses simple data about Excel parsing and it is referred as *ExcelConfiguration*. The configurations are the following:

- **examFirstHeader:** This is a string that must match the first header of the first column of the input Excel exam tab. It is used to know when to start parsing the exams.
- **constraintsFirstRow:** This specifies in which row can the algorithm start writing the constraints output in the user constraints tab.
- **constraintsFirstCol:** This specifies which is the base column in the constraints tab. This is equivalent to say how many columns are empty to the left of the constraints tables.

12.2.2 Excel data file

This file is the one that stores the information about the problem instance. The same format is kept for input and output so that an output file can be inputted again.

This Excel file requires three tabs: exams, constraints, and calendar days; which are commented in the following sections.

Exams tab

This must be the **first** tab and must contain the list of exams to be considered by the algorithm. The parsing will start in the headers row according to what was indicated in Section 12.2.1.

Exams will have a number of fields to be declared. For the sake of clarity they will be listed in the order they must be in the Excel columns. Since this input file is managed in Spanish, the headers will be listed in such language.

1. Curso: Subject course. It must be an integer between 0 and 4, standing 0 as the value for the elective subjects.
2. Sem: Subject semester. It must be an integer between 1 and 2.
3. Cod: Subject code. It is a sequence of characters.
4. Acron.: Subject acronym. Short sequence of characters to identify the subject.
5. Asignatura: Name of the subject.
6. Orden: It must be an integer value.
7. Contenido: Type of content for the exam.
8. Modalidad: Type of exam: online, delivery, etc.
9. Alumnos: number of students that may attend the exam.
10. Tiempo: Duration of the exam.
11. Fecha: Date on which the exam takes place.
12. Day: Week day on which the exam takes place.
13. Ini: Starting hour of the exam. It must be in Excel time format.
14. Fin: Ending hour of the exam. It must be in Excel time format.
15. Extra time: Extra time configured for the exam.
16. CN: Numerical Complexity value associated to the exam.
17. ID: Id assigned to the exam.
18. Tanda: Round to which the exam belongs.

As previously commented in Section 7.2, it is expected that the provided input list of exams contains both scheduled and unscheduled exams. Once the file is outputted, the exams that the algorithm was able to schedule will appear with the fields: “Fecha”, “Day”, “Ini” and “Fin” written.

Constraints tab

This must be the **second** tab and must contain list of constraint types, which will be aligned to the column specified in Section 12.2.1. Furthermore the lists must have at least one blank row between them.

There is a common structure in such lists. The first three rows must have the following pieces of data, one per row.

DD			
Two exams should/must not be placed on the same day.			
exam_id_1	exam_id_2	Hard?	Fulfilled?
1	42	True	
4	29	False	
5	38	False	
45	22	False	

Table 12.1: Constraint type list input example.

1. Constraint identifier: DD, SD, ...
2. Constraint description: A general text description.
3. Constraint headers: The headers needed for this constraint. (Names are not fixed, the parsing depends on the selected identifier.)

Having said that, each list headers depends on the constraint type, but they all have a similar format. Before explaining it, let us comment the data types that are handled:

- **exam_ids:** These are numbers which link to the ID of an exam in the exams tab. If no exam is found with that ID an error will rise.
- **dates:** Some constraints require one or more dates to work. It is important for that cells to be in Excel date format[1].
- **booleans:** Some booleans are used, for instance to state whether a constraint is hard or not.

All the constraints to be specified will have two common columns at the end, being such:

- **Hard?:** It is a boolean column which states if the constraint is hard.
- **Fulfilled?:** It is a boolean column. Its value is not read as input, it is used to specify in the output if that constraint was fulfilled in the provided schedule.

In the output, as an extra to the information that a constraint is not fulfilled, some extra fields are written for the constraint to easily know to which exam the ids where referring to. This information consists in the subject acronym and the content type of the exam.

An example of an input can be seen in Table 12.1. After processing this and more lists of different constraint types a possible result could be the one showed in Table 12.2.

Calendar tab

This must be the **third** tab in the Excel file. The available days for the algorithm to use as well as their associated time intervals must be written, starting on top left.

Even though there are no headers in the file, the format to do so is the following:

[1] This is done automatically by Excel when writing text similar to a date. Excel stores the dates with decimal numbers, thus in case the date format is not right the program will get incorrect data.

DD					
Two exams should/must not be placed on the same day.					
exam_id_1	exam_id_2	Hard?	Fulfilled?		
1	42	True	True		
4	29	False	False	Sub1-Theory	Sub2-Lab
5	38	False	True		
45	22	False	False	Sub5-Theory	Sub4-Theory

Table 12.2: Constraint type list output example.

Calendar day	Day initial hour	Day ending hour
14/06/2021	9:00	18:00
15/06/2021	9:00	19:00
16/06/2021	11:00	21:00
17/06/2021	9:00	21:00

Table 12.3: Calendar format in data Excel calendar tab.

12.3 Output Files

The execution of the algorithm will generate an output folder with the date and the time of the execution as name. Inside it the output files will be placed.

The directory contents are the following:

- **Exam Schedules Solutions:** There will be a number of folders that contain the solutions found by the algorithm.
- **fitnessGraph.csv:** This is a file containing the best fitness, average fitness and time records at each generation of the genetic algorithm. It can be used to plot a convergence graph. The order of the data is: generation, best fitness, average fitness, time (s).
- **errorLog.txt:** This file contains the last set of errors showed to the user in the console. It is generated at the same time as they are printed so that in case they are too many the user can read the file instead.
- **inputUncoloredLog:** This file shows the process of parsing the input files. It contains the number of exams, calendar days and constraints parsed and the errors.
- **gLog.txt:** This file shows in a verbose format the same data that can be seen in the fitnessGraph.csv file. The frequency of the written entries is configured at the *geneticParameters* file.

12.3.1 Exam schedules solutions

Inside a solution folder two files can be found. The first one is the Excel file that contains the exam schedule, that is, the output version of the file exposed in Section 12.2.2.

With that file, in case the user asks for its generation another .csv file will be created. This file contains a single row of data with quality factors of the solution within the same folder. It has no headers, but column order means:

1. **Number of unscheduled exams.**
2. **Number of unfulfilled user constraints.**
3. **Number of unfulfilled user constraints divided by the number of soft user constraints.**
4. **Number of unfulfilled user constraints divided by the total number of user constraints.**
5. **Number of minutes occupied in the resting interval in the whole schedule.**

Chapter 13

Budget

In this chapter an estimated budget will be provided based on the project planning shown in Chapter 6. To do so an imaginary company will be created, being the only workers of such company the student and its tutor.

13.1 Company Description

In the proposed scenario, the described company has hired both workers for the project duration. The income generated by this project must be sufficient to pay both workers for the nine months that the project lasts. Furthermore, a 25% benefit is considered.

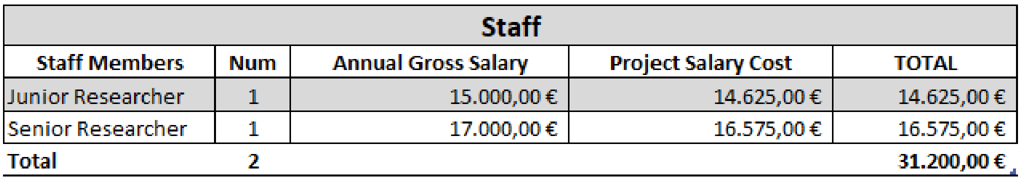
Because the work is done at the author's house there are no indirect costs such as cleaning, water, among others that are typically considered in big companies.

13.1.1 Workers

As commented in the introduction only two workers are considered. The average annual gross salary has been consulted on *glassdoor*[1]. That gross salary was reduced since we are considering a day having just 4 hours in the project.

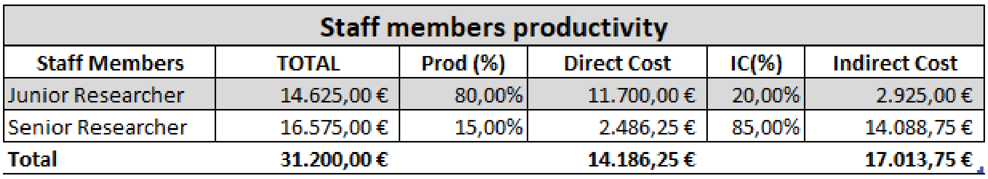
In this case a **30% increase** has been considered as the cost to be paid by the company for the worker, in addition to the actual salary. The total staff cost can be seen in Figure 13.1. Note that column *Project Salary Cost* considers the project lasting for nine months.

[1] The check has been checked on 16/May/2022 for the last time.



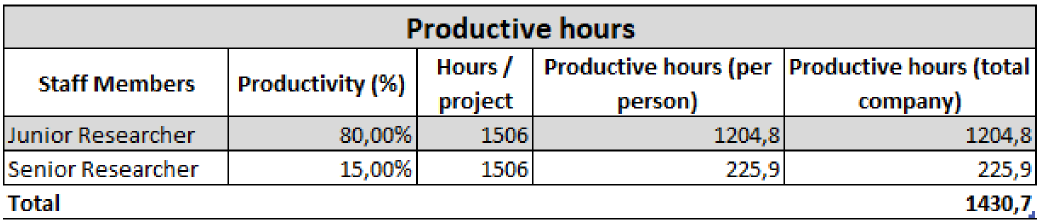
Staff				
Staff Members	**Num**	**Annual Gross Salary**	**Project Salary Cost**	**TOTAL**
Junior Researcher	1	15.000,00 €	14.625,00 €	14.625,00 €
Senior Researcher	1	17.000,00 €	16.575,00 €	16.575,00 €
Total	**2**			**31.200,00 €**

Figure 13.1: Budget - Company Description - Staff



Staff members productivity					
Staff Members	TOTAL	Prod (%)	Direct Cost	IC(%)	Indirect Cost
Junior Researcher	14.625,00 €	80,00%	11.700,00 €	20,00%	2.925,00 €
Senior Researcher	16.575,00 €	15,00%	2.486,25 €	85,00%	14.088,75 €
Total	**31.200,00 €**		**14.186,25 €**		**17.013,75 €**

Figure 13.2: Budget - Company Description - Staff Productivity



Productive hours				
Staff Members	Productivity (%)	Hours / project	Productive hours (per person)	Productive hours (total company)
Junior Researcher	80,00%	1506	1204,8	1204,8
Senior Researcher	15,00%	1506	225,9	225,9
Total				**1430,7**

Figure 13.3: Budget - Company Description - Productive Hours Company

13.1.2 Staff productivity

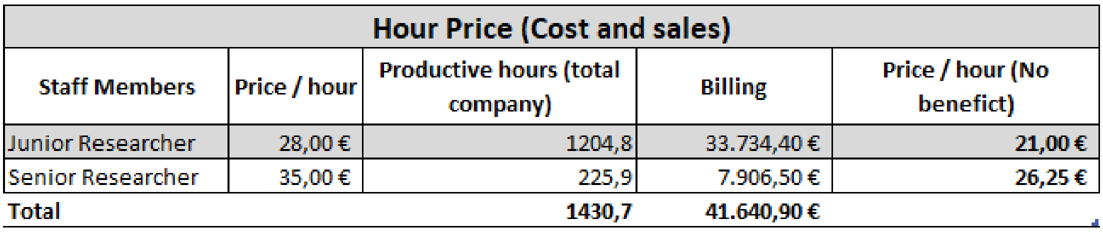
The productivity of the staff has an significant importance on the direct and indirect costs, specially when the company has just two workers and only one project. The *junior researcher* has much more workload than the *senior researcher* which is just a supervisor. This can be seen in Figure 13.2.

13.1.3 Productive hours

Taking into account the exposed values in Figure 13.5, and combining that information with the total time invested in the project we obtain Figure 13.3.

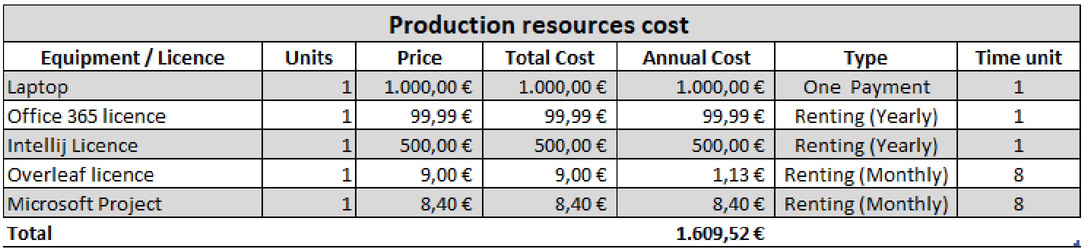
13.1.4 Prices per hour

Considering the amount of time that each staff person is productive and assigning a value at which we are selling such individual work, we obtain how much money is earned by means of their work. This can be seen in Figure 13.4.



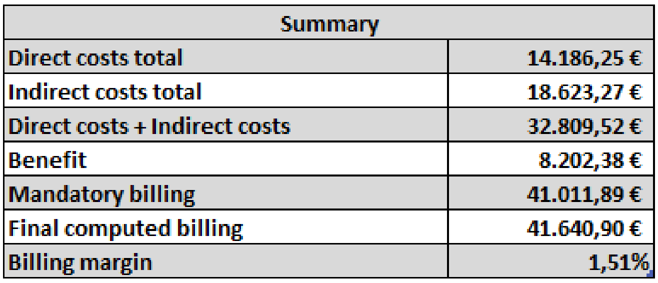
Hour Price (Cost and sales)				
Staff Members	Price / hour	Productive hours (total company)	Billing	Price / hour (No benefict)
Junior Researcher	28,00 €	1204,8	33.734,40 €	**21,00 €**
Senior Researcher	35,00 €	225,9	7.906,50 €	**26,25 €**
Total		**1430,7**	**41.640,90 €**	

Figure 13.4: Budget - Company Description - Prices per Hour



Production resources cost						
Equipment / Licence	Units	Price	Total Cost	Annual Cost	Type	Time unit
Laptop	1	1.000,00 €	1.000,00 €	1.000,00 €	One Payment	1
Office 365 licence	1	99,99 €	99,99 €	99,99 €	Renting (Yearly)	1
Intellij Licence	1	500,00 €	500,00 €	500,00 €	Renting (Yearly)	1
Overleaf licence	1	9,00 €	9,00 €	1,13 €	Renting (Monthly)	8
Microsoft Project	1	8,40 €	8,40 €	8,40 €	Renting (Monthly)	8
Total				**1.609,52 €**		

Figure 13.5: Budget - Company Description - Production Resources



Summary	
Direct costs total	**14.186,25 €**
Indirect costs total	**18.623,27 €**
Direct costs + Indirect costs	**32.809,52 €**
Benefit	**8.202,38 €**
Mandatory billing	**41.011,89 €**
Final computed billing	**41.640,90 €**
Billing margin	**1,51%**

Figure 13.6: Budget - Company Description - Summary

13.1.5 Production resources

In this section some equipment and licenses that are necessary for the project will be detailed. They can be seen in Figure 13.5.

13.1.6 Summary

Adding the exposed direct and indirect costs, as well as the benefits that the company wishes to earn, the total amount of income that the company must generate is obtained. In Figure 13.6, such value is compared with the total amount of money that the company earns which was previously obtained in Figure 13.4.

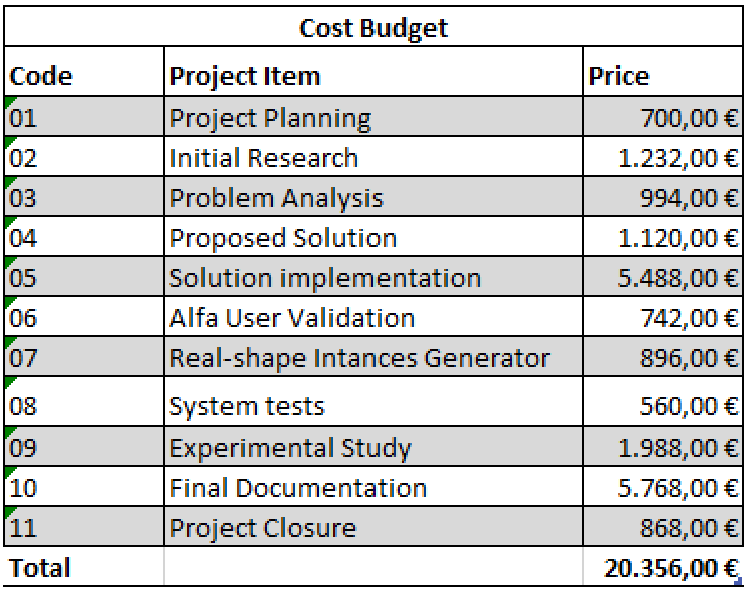
Finally the margin between these two amounts is computed. The fact that it is positive means that the company can pay the specified costs and earn the desired benefit. Since the margin is close to zero, the company cannot lower the prices in Figure 13.4 without starting to lose benefits.

13.2 Project Costs

In this section the development costs of the project will be detailed. As done in Chapter 6, the project is divided in eleven items:

1. **Project planning.** (Figure 13.10)
2. **Initial research.** (Figure 13.11)
3. **Problem analysis.** (Figure 13.12)
4. **Proposed solution.** (Figure 13.13)
5. **Solution implementation.** (Figure 13.14)
6. **Alfa user validation.** (Figure 13.15)
7. **Real-shape instances generator.** (Figure 13.16)
8. **System tests.** (Figure 13.17)
9. **Experimental study.** (Figure 13.18)
10. **Final documentation.** (Figure 13.19)
11. **Project closure.** (Figure 13.20)

The cost of each project item and the total cost of the project can be seen in Figure 13.7.



Cost Budget		
Code	**Project Item**	**Price**
01	Project Planning	700,00 €
02	Initial Research	1.232,00 €
03	Problem Analysis	994,00 €
04	Proposed Solution	1.120,00 €
05	Solution implementation	5.488,00 €
06	Alfa User Validation	742,00 €
07	Real-shape Intances Generator	896,00 €
08	System tests	560,00 €
09	Experimental Study	1.988,00 €
10	Final Documentation	5.768,00 €
11	Project Closure	868,00 €
Total		**20.356,00 €**

Figure 13.7: Budget - Project costs - Summary

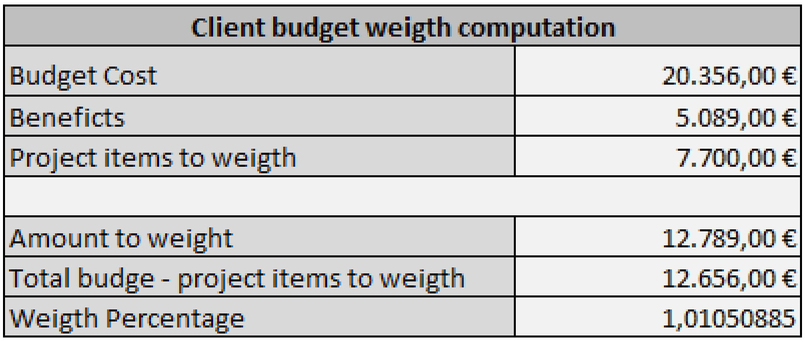
13.2.1 Client budget computation

As commented previously, it is intended to obtain 25% benefit from this project, therefore we must perform some weighting on the costs shown in Figure 13.6. Furthermore, there are some items that are inappropriate to be sent to the client. The cost of these project items will be also weighted over the rest of project items that are going to be listed in the client budget. For such reason we must consider the following quantities for the weighting.

- 25% of the total project cost.
- Project planning.
- Initial research.
- Final documentation.

The weighting process can be seen in Figure 13.8. The meaning of each field is the following:

- **Budget Cost.** The total amount of money that the project costs.
- **Benefits.** The 25% of *Budget Cost.*
- **Project items to weight.** The addition of the costs of the project items that are not going to be sent to the client.
- **Amount to weight.** The sum of *Benefits* and *Project items to weight.*
- **Total budget - Project items to weight.** The amount of money to which the weighting is going to be applied, that is, the subtraction of *Project items to weight* from *Budget Cost.*
- **Weight Percentage.** This is the amount by which the cost of the project items that are going to be sent must increase. For instance, if $WeightPercentage = 1$, then all the costs of the project items should be doubled.



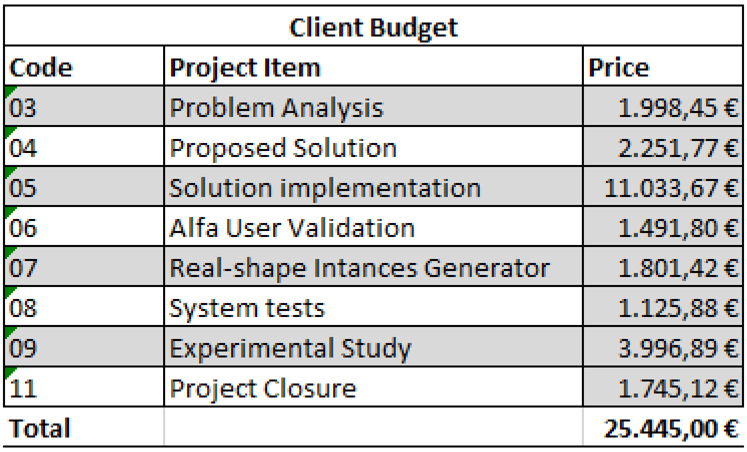
Client budget weigth computation	
Budget Cost	20.356,00 €
Benefticts	5.089,00 €
Project items to weigth	7.700,00 €
Amount to weight	12.789,00 €
Total budge - project items to weigth	12.656,00 €
Weigth Percentage	1,01050885

Figure 13.8: Budget - Client budget - Weighting percentage computation

As a final result we obtain the client budget that can be seen in Figure 13.9.

13.2.2 Project items detailed costs

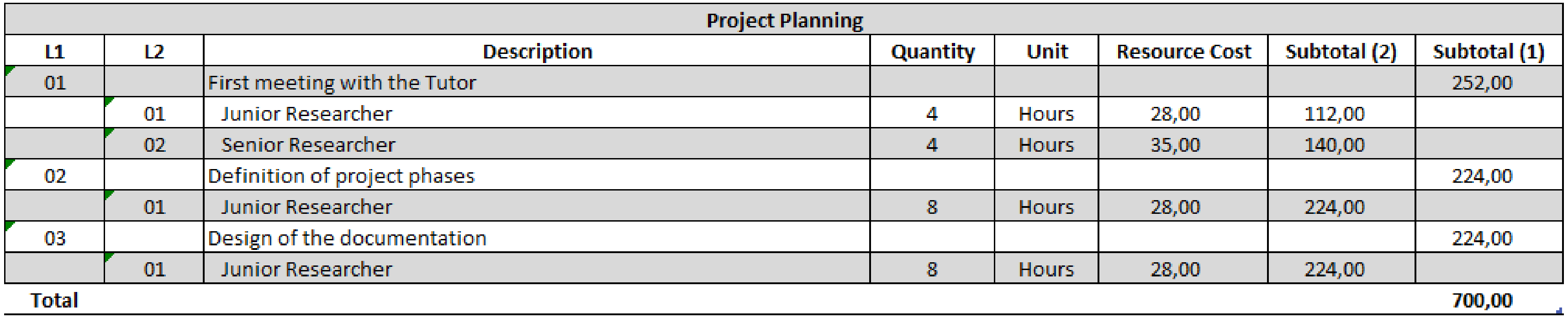
In this section each project item will be detailed.



Client Budget

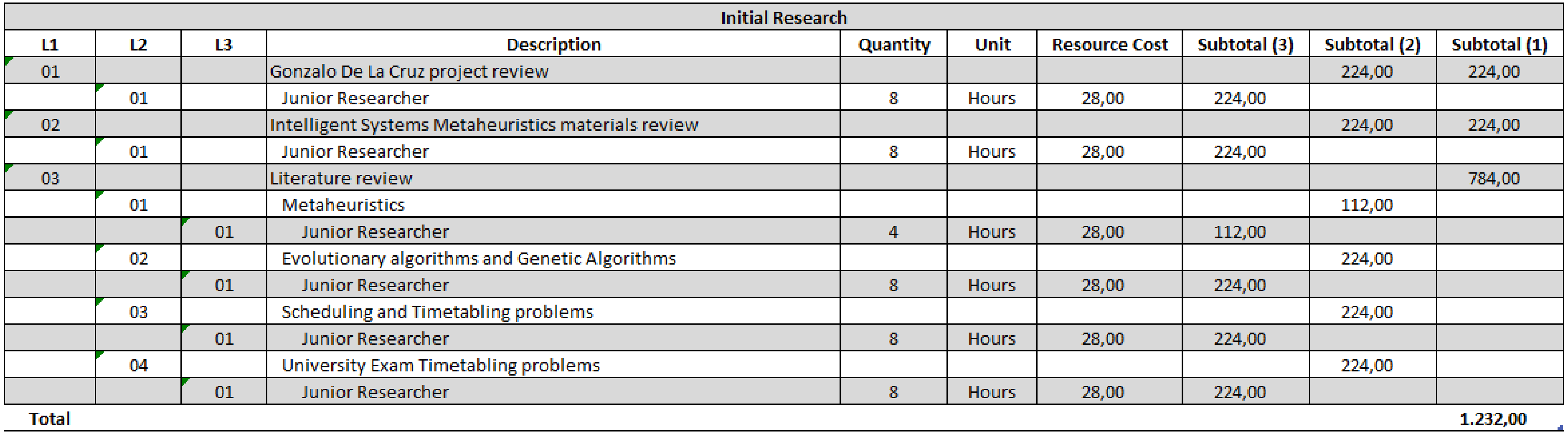
Code	Project Item	Price
03	Problem Analysis	1.998,45 €
04	Proposed Solution	2.251,77 €
05	Solution implementation	11.033,67 €
06	Alfa User Validation	1.491,80 €
07	Real-shape Intances Generator	1.801,42 €
08	System tests	1.125,88 €
09	Experimental Study	3.996,89 €
11	Project Closure	1.745,12 €
Total		**25.445,00 €**

Figure 13.9: Budget - Client budget



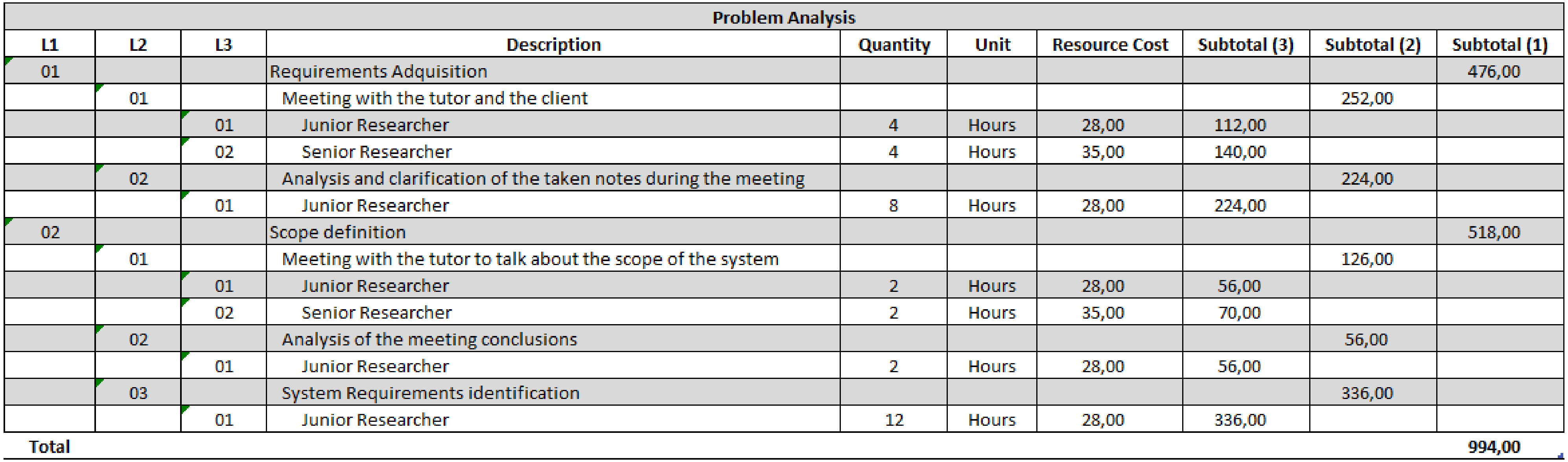
Project Planning							
L1	L2	Description	Quantity	Unit	Resource Cost	Subtotal (2)	Subtotal (1)
01		First meeting with the Tutor					252,00
	01	Junior Researcher	4	Hours	28,00	112,00	
	02	Senior Researcher	4	Hours	35,00	140,00	
02		Definition of project phases					224,00
	01	Junior Researcher	8	Hours	28,00	224,00	
03		Design of the documentation					224,00
	01	Junior Researcher	8	Hours	28,00	224,00	
Total							700,00

Figure 13.10: Budget - Project costs - Project planning



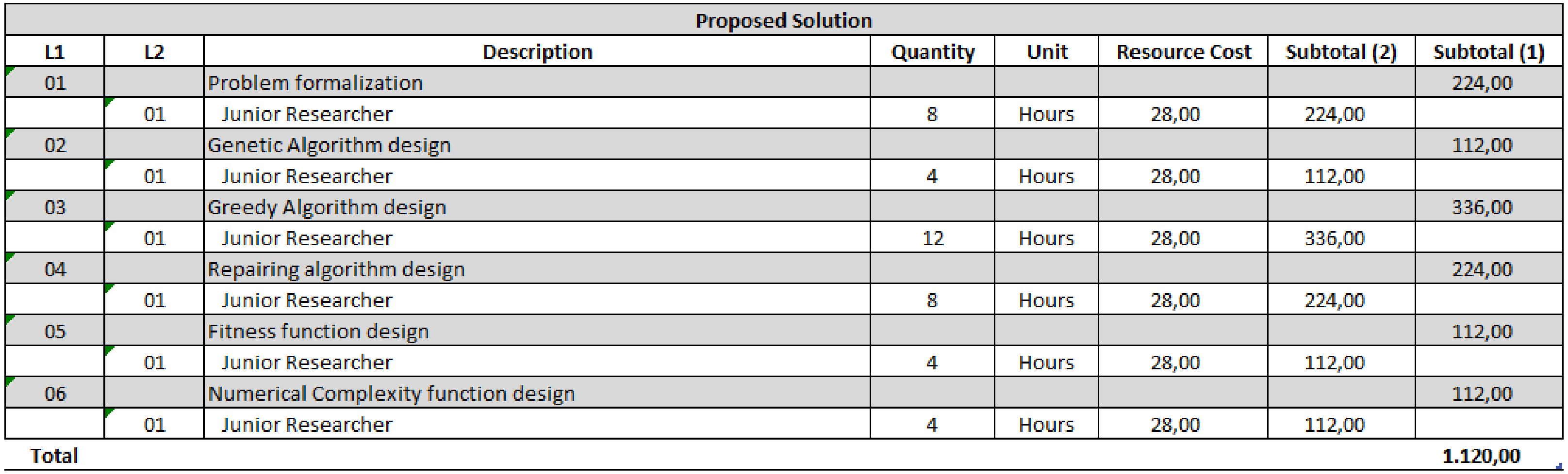
Initial Research									
L1	L2	L3	Description	Quantity	Unit	Resource Cost	Subtotal (3)	Subtotal (2)	Subtotal (1)
01			Gonzalo De La Cruz project review					224,00	224,00
	01		Junior Researcher	8	Hours	28,00	224,00		
02			Intelligent Systems Metaheuristics materials review					224,00	224,00
	01		Junior Researcher	8	Hours	28,00	224,00		
03			Literature review						784,00
	01		Metaheuristics					112,00	
		01	Junior Researcher	4	Hours	28,00	112,00		
	02		Evolutionary algorithms and Genetic Algorithms					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
	03		Scheduling and Timetabling problems					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
	04		University Exam Timetabling problems					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
Total									1.232,00

Figure 13.11: Budget - Project costs - Initial research



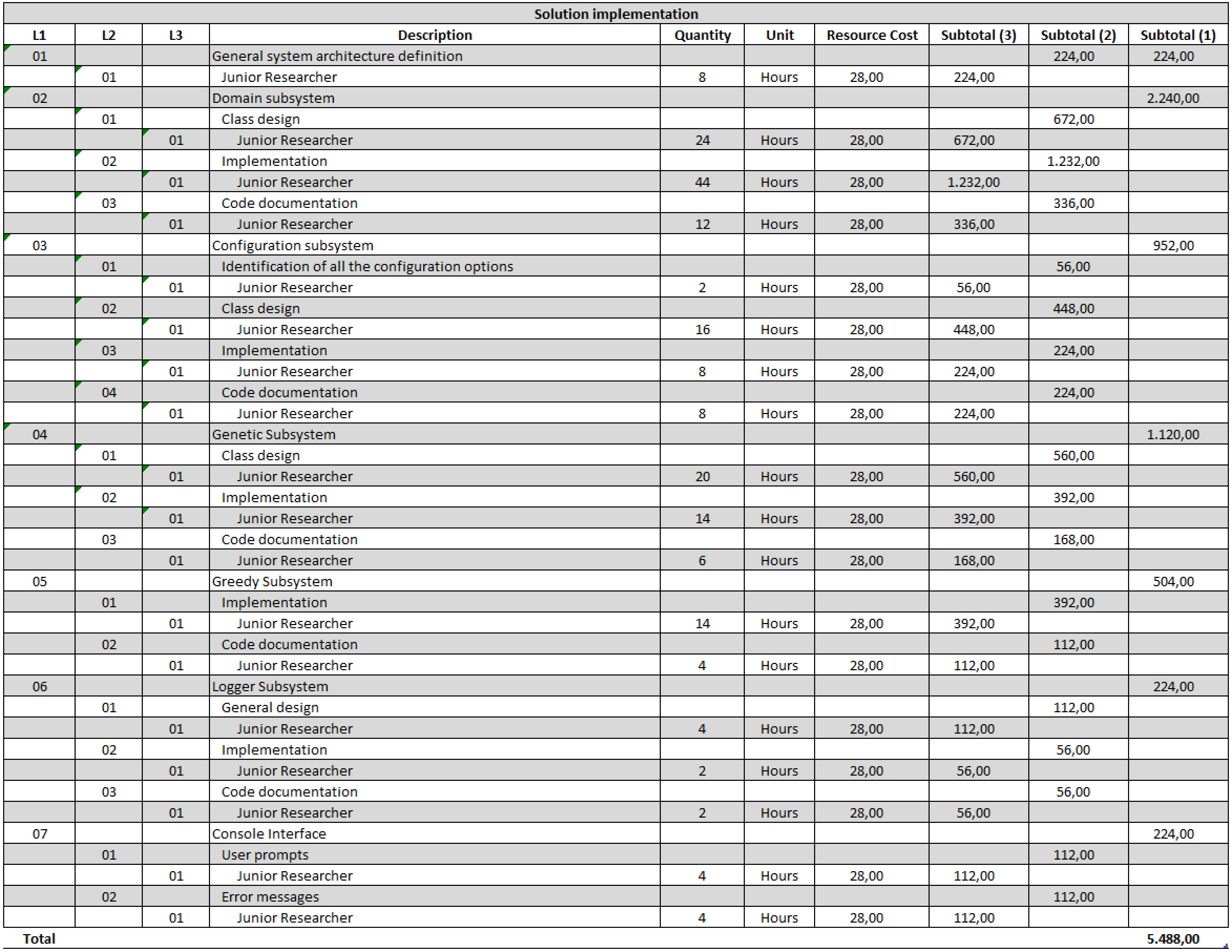
Problem Analysis									
L1	L2	L3	Description	Quantity	Unit	Resource Cost	Subtotal (3)	Subtotal (2)	Subtotal (1)
01			Requirements Adquisition						476,00
	01		Meeting with the tutor and the client					252,00	
		01	Junior Researcher	4	Hours	28,00	112,00		
		02	Senior Researcher	4	Hours	35,00	140,00		
	02		Analysis and clarification of the taken notes during the meeting					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
02			Scope definition						518,00
	01		Meeting with the tutor to talk about the scope of the system					126,00	
		01	Junior Researcher	2	Hours	28,00	56,00		
		02	Senior Researcher	2	Hours	35,00	70,00		
	02		Analysis of the meeting conclusions					56,00	
		01	Junior Researcher	2	Hours	28,00	56,00		
	03		System Requirements identification					336,00	
		01	Junior Researcher	12	Hours	28,00	336,00		
Total									994,00

Figure 13.12: Budget - Project costs - Problem analysis



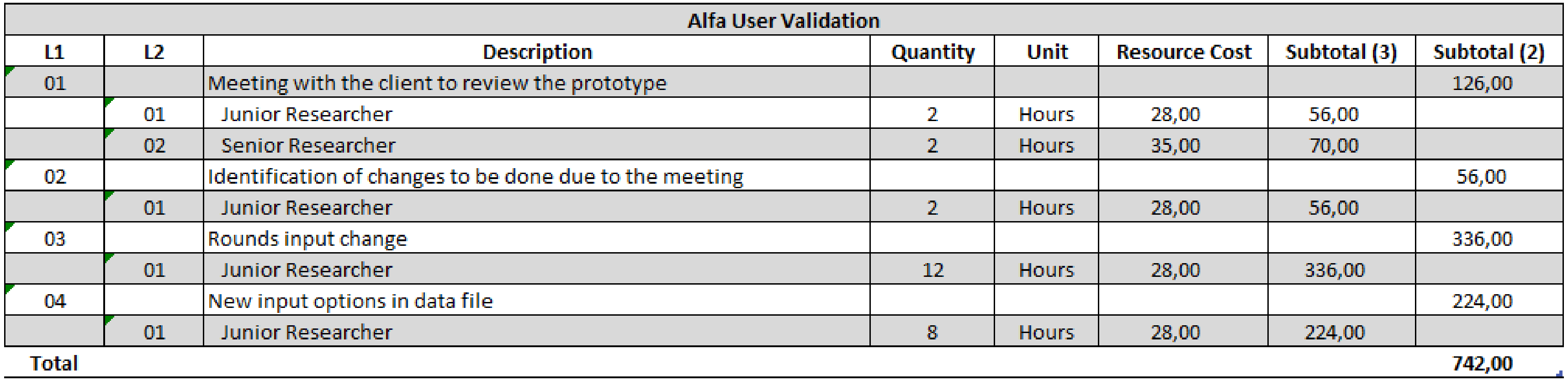
Proposed Solution							
L1	L2	Description	Quantity	Unit	Resource Cost	Subtotal (2)	Subtotal (1)
01		Problem formalization					224,00
	01	Junior Researcher	8	Hours	28,00	224,00	
02		Genetic Algorithm design					112,00
	01	Junior Researcher	4	Hours	28,00	112,00	
03		Greedy Algorithm design					336,00
	01	Junior Researcher	12	Hours	28,00	336,00	
04		Repairing algorithm design					224,00
	01	Junior Researcher	8	Hours	28,00	224,00	
05		Fitness function design					112,00
	01	Junior Researcher	4	Hours	28,00	112,00	
06		Numerical Complexity function design					112,00
	01	Junior Researcher	4	Hours	28,00	112,00	
Total							1.120,00

Figure 13.13: Budget - Project costs - Proposed solution



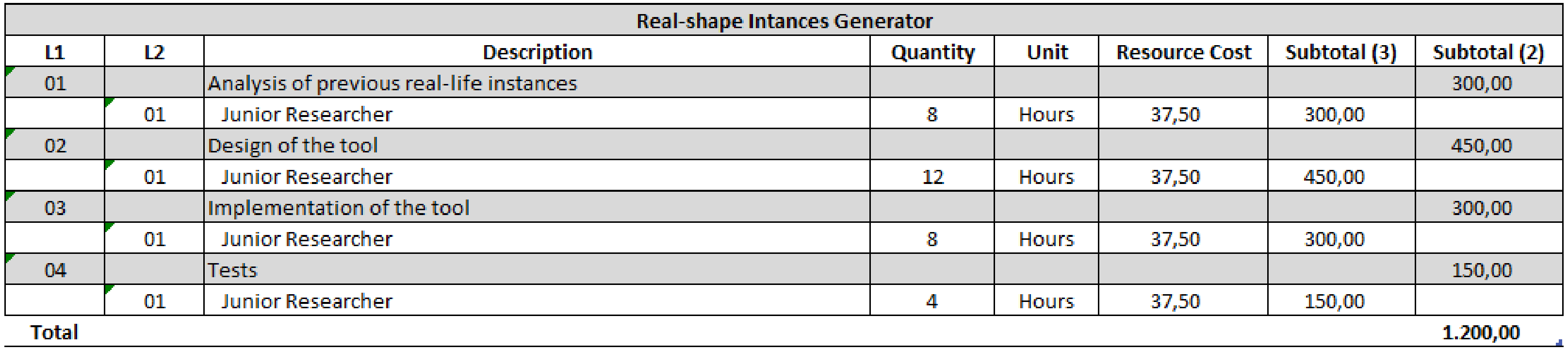
Solution implementation									
L1	**L2**	**L3**	**Description**	**Quantity**	**Unit**	**Resource Cost**	**Subtotal (3)**	**Subtotal (2)**	**Subtotal (1)**
01			General system architecture definition					224,00	224,00
	01		Junior Researcher	8	Hours	28,00	224,00		
02			Domain subsystem						2.240,00
	01		Class design					672,00	
		01	Junior Researcher	24	Hours	28,00	672,00		
	02		Implementation					1.232,00	
		01	Junior Researcher	44	Hours	28,00	1.232,00		
	03		Code documentation					336,00	
		01	Junior Researcher	12	Hours	28,00	336,00		
03			Configuration subsystem						952,00
	01		Identification of all the configuration options					56,00	
		01	Junior Researcher	2	Hours	28,00	56,00		
	02		Class design					448,00	
		01	Junior Researcher	16	Hours	28,00	448,00		
	03		Implementation					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
	04		Code documentation					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
04			Genetic Subsystem						1.120,00
	01		Class design					560,00	
		01	Junior Researcher	20	Hours	28,00	560,00		
	02		Implementation					392,00	
		01	Junior Researcher	14	Hours	28,00	392,00		
	03		Code documentation					168,00	
		01	Junior Researcher	6	Hours	28,00	168,00		
05			Greedy Subsystem						504,00
	01		Implementation					392,00	
		01	Junior Researcher	14	Hours	28,00	392,00		
	02		Code documentation					112,00	
		01	Junior Researcher	4	Hours	28,00	112,00		
06			Logger Subsystem						224,00
	01		General design					112,00	
		01	Junior Researcher	4	Hours	28,00	112,00		
	02		Implementation					56,00	
		01	Junior Researcher	2	Hours	28,00	56,00		
	03		Code documentation					56,00	
		01	Junior Researcher	2	Hours	28,00	56,00		
07			Console Interface						224,00
	01		User prompts					112,00	
		01	Junior Researcher	4	Hours	28,00	112,00		
	02		Error messages					112,00	
		01	Junior Researcher	4	Hours	28,00	112,00		
Total									**5.488,00**

Figure 13.14: Budget - Project costs - Solution implementation



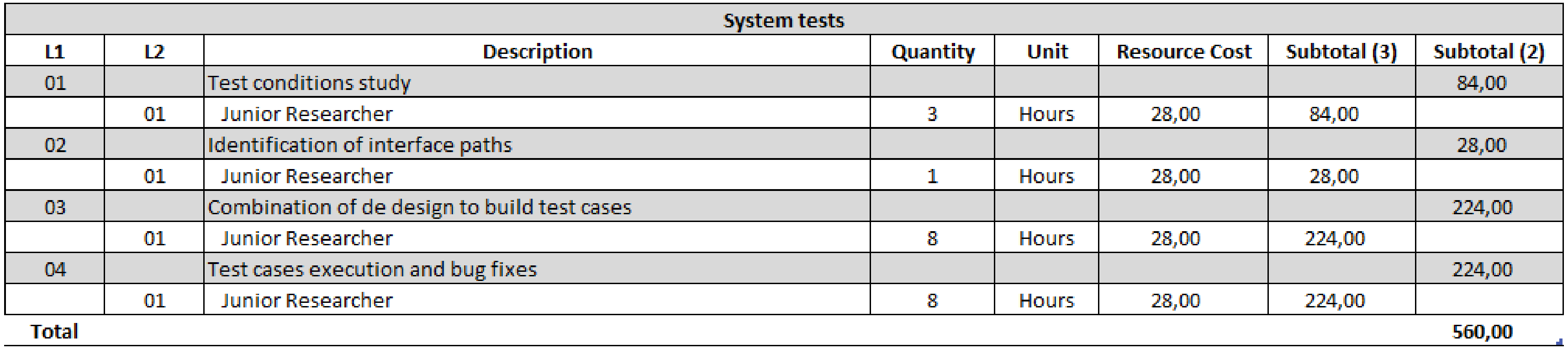
Alfa User Validation							
L1	L2	Description	Quantity	Unit	Resource Cost	Subtotal (3)	Subtotal (2)
01		Meeting with the client to review the prototype					126,00
	01	Junior Researcher	2	Hours	28,00	56,00	
	02	Senior Researcher	2	Hours	35,00	70,00	
02		Identification of changes to be done due to the meeting					56,00
	01	Junior Researcher	2	Hours	28,00	56,00	
03		Rounds input change					336,00
	01	Junior Researcher	12	Hours	28,00	336,00	
04		New input options in data file					224,00
	01	Junior Researcher	8	Hours	28,00	224,00	
Total							742,00

Figure 13.15: Budget - Project costs - Alfa user validation



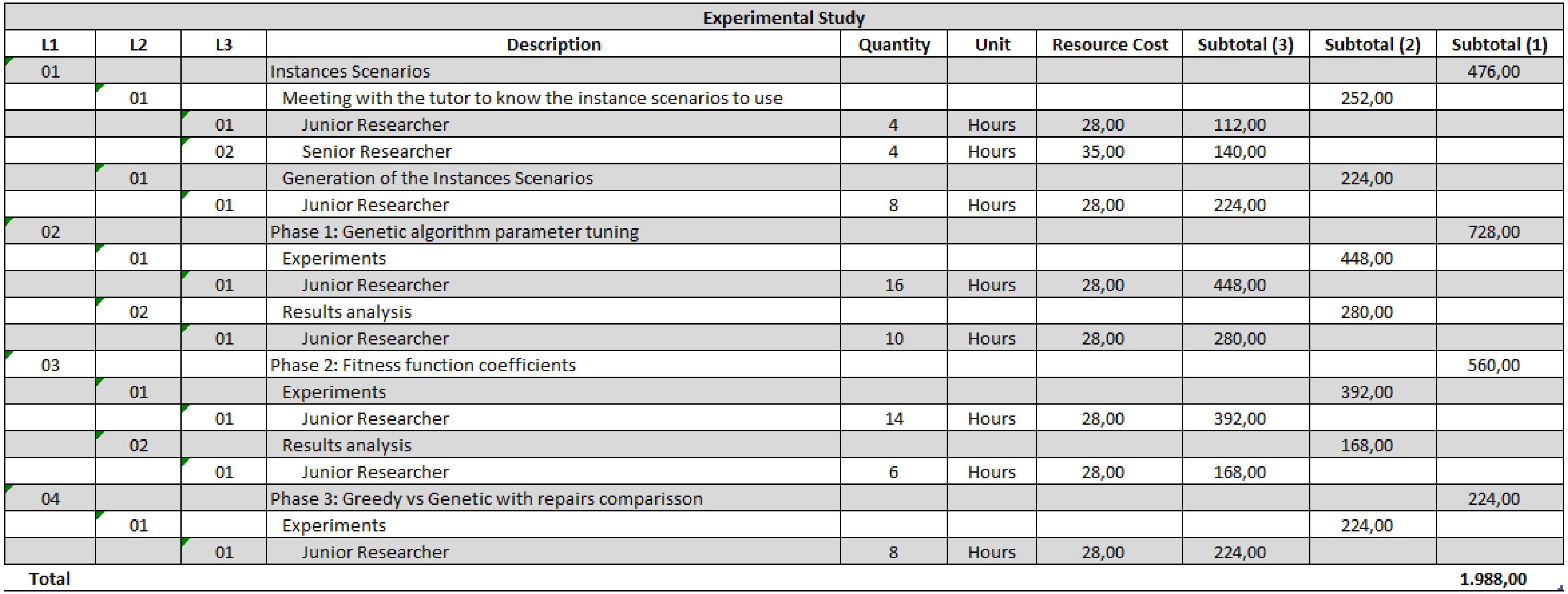
Real-shape Intances Generator							
L1	L2	Description	Quantity	Unit	Resource Cost	Subtotal (3)	Subtotal (2)
01		Analysis of previous real-life instances					300,00
	01	Junior Researcher	8	Hours	37,50	300,00	
02		Design of the tool					450,00
	01	Junior Researcher	12	Hours	37,50	450,00	
03		Implementation of the tool					300,00
	01	Junior Researcher	8	Hours	37,50	300,00	
04		Tests					150,00
	01	Junior Researcher	4	Hours	37,50	150,00	
Total							1.200,00

Figure 13.16: Budget - Project costs - Real-shape instances generator



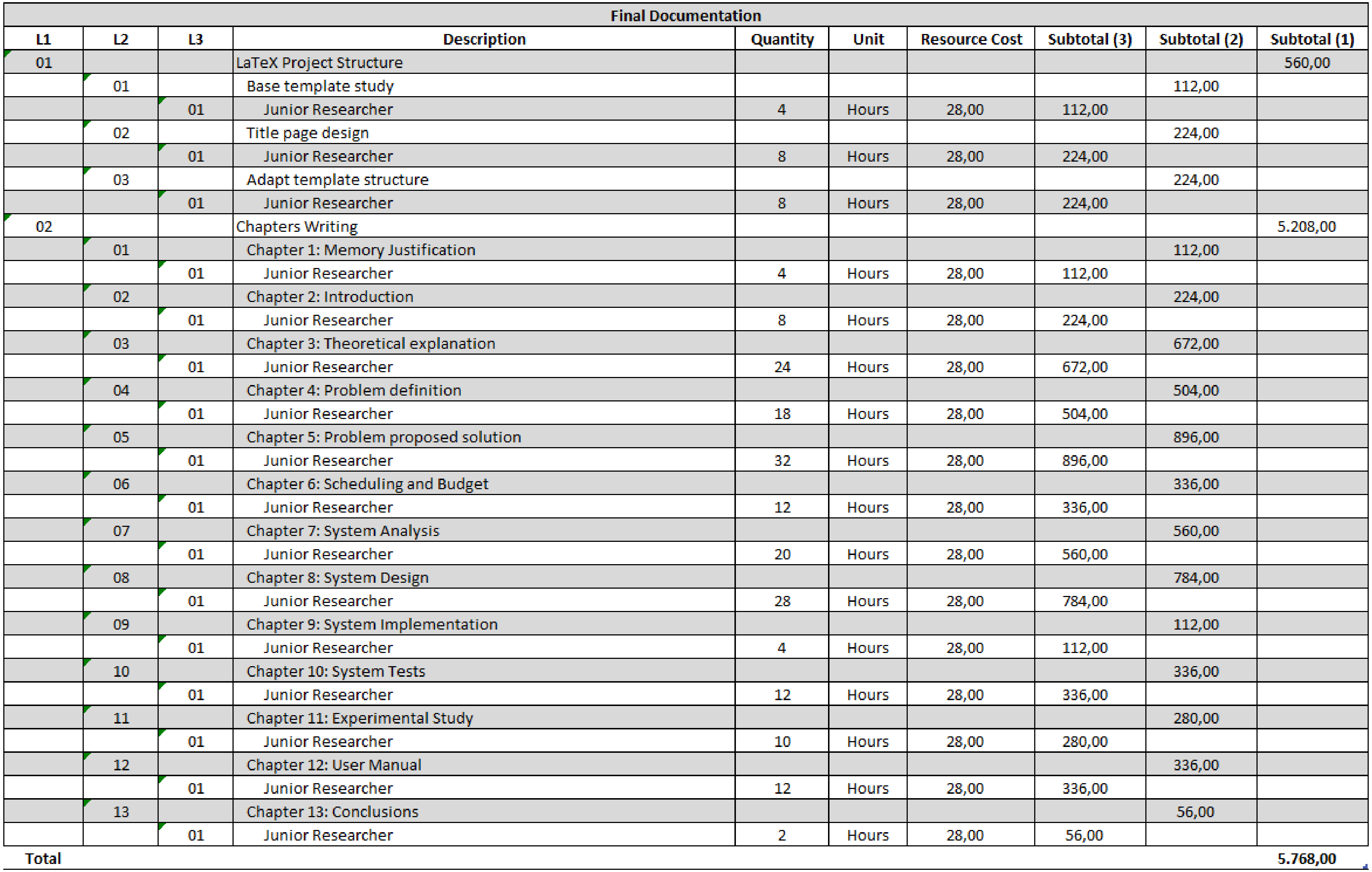
System tests							
L1	L2	Description	Quantity	Unit	Resource Cost	Subtotal (3)	Subtotal (2)
01		Test conditions study					84,00
	01	Junior Researcher	3	Hours	28,00	84,00	
02		Identification of interface paths					28,00
	01	Junior Researcher	1	Hours	28,00	28,00	
03		Combination of de design to build test cases					224,00
	01	Junior Researcher	8	Hours	28,00	224,00	
04		Test cases execution and bug fixes					224,00
	01	Junior Researcher	8	Hours	28,00	224,00	
Total							560,00

Figure 13.17: Budget - Project costs - System tests



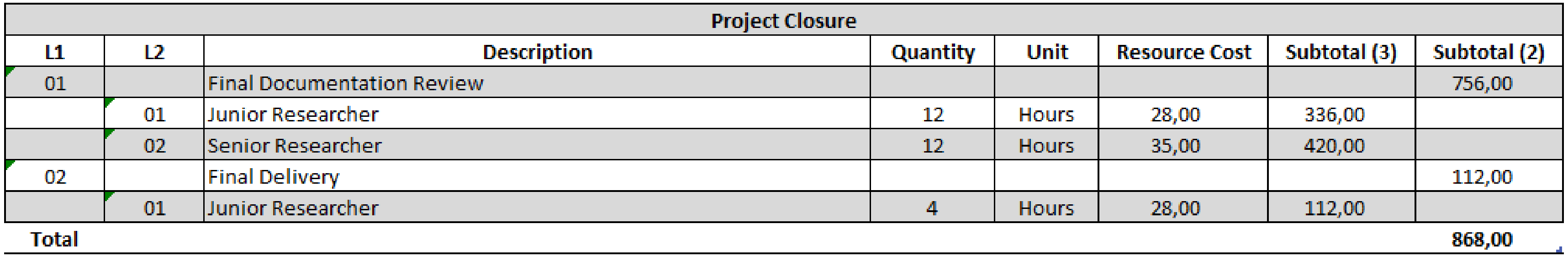
Experimental Study									
L1	L2	L3	Description	Quantity	Unit	Resource Cost	Subtotal (3)	Subtotal (2)	Subtotal (1)
01			Instances Scenarios						476,00
	01		Meeting with the tutor to know the instance scenarios to use					252,00	
		01	Junior Researcher	4	Hours	28,00	112,00		
		02	Senior Researcher	4	Hours	35,00	140,00		
	01		Generation of the Instances Scenarios					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
02			Phase 1: Genetic algorithm parameter tuning						728,00
	01		Experiments					448,00	
		01	Junior Researcher	16	Hours	28,00	448,00		
	02		Results analysis					280,00	
		01	Junior Researcher	10	Hours	28,00	280,00		
03			Phase 2: Fitness function coefficients						560,00
	01		Experiments					392,00	
		01	Junior Researcher	14	Hours	28,00	392,00		
	02		Results analysis					168,00	
		01	Junior Researcher	6	Hours	28,00	168,00		
04			Phase 3: Greedy vs Genetic with repairs comparisson						224,00
	01		Experiments					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
Total									1.988,00

Figure 13.18: Budget - Project costs - Experimental study



Final Documentation									
L1	L2	L3	Description	Quantity	Unit	Resource Cost	Subtotal (3)	Subtotal (2)	Subtotal (1)
01			LaTeX Project Structure						560,00
	01		Base template study					112,00	
		01	Junior Researcher	4	Hours	28,00	112,00		
	02		Title page design					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
	03		Adapt template structure					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
02			Chapters Writing						5.208,00
	01		Chapter 1: Memory Justification					112,00	
		01	Junior Researcher	4	Hours	28,00	112,00		
	02		Chapter 2: Introduction					224,00	
		01	Junior Researcher	8	Hours	28,00	224,00		
	03		Chapter 3: Theoretical explanation					672,00	
		01	Junior Researcher	24	Hours	28,00	672,00		
	04		Chapter 4: Problem definition					504,00	
		01	Junior Researcher	18	Hours	28,00	504,00		
	05		Chapter 5: Problem proposed solution					896,00	
		01	Junior Researcher	32	Hours	28,00	896,00		
	06		Chapter 6: Scheduling and Budget					336,00	
		01	Junior Researcher	12	Hours	28,00	336,00		
	07		Chapter 7: System Analysis					560,00	
		01	Junior Researcher	20	Hours	28,00	560,00		
	08		Chapter 8: System Design					784,00	
		01	Junior Researcher	28	Hours	28,00	784,00		
	09		Chapter 9: System Implementation					112,00	
		01	Junior Researcher	4	Hours	28,00	112,00		
	10		Chapter 10: System Tests					336,00	
		01	Junior Researcher	12	Hours	28,00	336,00		
	11		Chapter 11: Experimental Study					280,00	
		01	Junior Researcher	10	Hours	28,00	280,00		
	12		Chapter 12: User Manual					336,00	
		01	Junior Researcher	12	Hours	28,00	336,00		
	13		Chapter 13: Conclusions					56,00	
		01	Junior Researcher	2	Hours	28,00	56,00		
Total									5.768,00

Figure 13.19: Budget - Project costs - Final documentation



Project Closure							
L1	L2	Description	Quantity	Unit	Resource Cost	Subtotal (3)	Subtotal (2)
01		Final Documentation Review					756,00
	01	Junior Researcher	12	Hours	28,00	336,00	
	02	Senior Researcher	12	Hours	35,00	420,00	
02		Final Delivery					112,00
	01	Junior Researcher	4	Hours	28,00	112,00	
Total							868,00

Figure 13.20: Budget - Project costs - Project closure

Chapter 14

Conclusions and Future Work

14.1 Conclusions

Taking into consideration the results obtained we can state that the project has fulfilled the goals set at its beginning.

After performing an analysis of the particular problem that the University of Oviedo School of Computer Engineering had, we concluded that there were no exact match solved in the literature due to the domain specific peculiarities of the problem.

The problem was formalized and the shape of real problem instances was studied. A random generator of real-shape problem instances was done to test and tune the proposed solution algorithm. For doing the latter, an experimental study was carried out. In such study, multiple algorithms were compared to show the performance gain of the proposed solution.

Even though the application was initially thought as a prototype to showcase the proposed solution, Software Engineering methodologies were applied to build a high-quality application to be regularly used by the client. Good design principles were applied to make the application easy to extend and modify. Furthermore a test plan was designed to ensure the quality of the application.

A solution was proposed and tested, trying to optimize the current real life process in the school. Such solution was proven effective and far more efficient than the actual process during the experimental study done in Chapter 11.

The prototype application will be provided to the School of Computer Science, so that it can be used as it is or as a component for a larger system. Source code and documentation will also be provided.

14.2 Future Work

Some extra research and work can be done over the results of this project. A list of possible extensions will be provided briefly detailing each of them. Furthermore, an explanation of why some extensions were not considered in the actual project is also provided.

- **Implementing new constraints**: It is expected that the need of new user constraints arise. The application was built to ensure that this extension process is straightforward. No other constraints were considered in the project because the client did not need more at the present time.
- **Consider spatial resources (classrooms, laboratories, etc.)**: This project provides a schedule for the exams, but the students must fit into spatial resources. It would be interesting to use the obtained results to build a larger system to fully automate every resource assignment of the school, both temporal and spatial.
- **User interface**: A graphical user interface could be interesting, specially to change easily all the configuration options. No GUI was built because the client asked for a CLI.
- **Server application**: A server application built from the provided code would allow the functionality to be used in other faculties, since the context is similar.
- **Implementing the algorithm in a higher performance programming language**: Some performance issues could arise if considering higher problem sizes. Now that the design was proved useful, building the same system in C++ could be a interesting, but only if such performance is truly required.

References

[1] Roberto Javier Asín Achá and Robert Nieuwenhuis. Curriculum-based course timetabling with SAT and maxsat. *Ann. Oper. Res.*, 218(1):71–91, 2014.

[2] Fadi A Aloul, Syed ZH Zahidi, Anas Al-Farra, Basel Al-Roh, and Bashar Al-Rawi. Solving the employee timetabling problem using advanced "SAT" & ILP techniques. *J. Comput.*, 8(4):851–858, 2013.

[3] Lawrence Davis. Applying adaptive algorithms to epistatic domains. In *IJCAI*, pages 162–164, 1985.

[4] Gonzalo de la Cruz Fernández. Metaheuristics for the assignment of students to class groups. *Final Degree Project. Bachelor of Software Engineering. School of Computer Engineering. University of Oviedo*, 2018.

[5] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, 6(2):182–197, 2002.

[6] George H.G. Fonseca, Haroldo G. Santos, Eduardo G. Carrano, and Thomas J.R. Stidsen. Integer programming techniques for educational timetabling. *European Journal of Operational Research*, 262(1):28–39, 2017.

[7] Begum Genc and Barry O'Sullivan. A two-phase constraint programming model for examination timetabling at university college cork. In Simonis (13), pages 724–742.

[8] J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[9] Manoj Kumar, Mohammad Husain, Naveen Upreti, and Deepti Gupta. Genetic algorithm: Review and application. *Available at SSRN 3529843*, 2010.

[10] Nuno Leite, Fernando Melício, and Agostinho C. Rosa. A fast simulated annealing algorithm for the examination timetabling problem. *Expert Systems with Applications*, 122:137–151, 2019.

[11] Souad Larabi Marie-Sainte. A survey of particle swarm optimization techniques for solving university examination timetabling problem. *Artif. Intell. Rev.*, 44(4):537–546, 2015.

[12] Hugo Terashima Marin. Combinations of gas and csp strategies for solving examination timetabling problems. 1998.

[13] Helmut Simonis, editor. *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science.* Springer, 2020.

[14] Ting Song, Sanya Liu, Xiangyang Tang, Xicheng Peng, and Mao Chen. An iterated local search algorithm for the university course timetabling problem. *Applied Soft Computing*, 68:597–608, 2018.

[15] El-Ghazali Talbi. *Metaheuristics - From Design to Implementation.* Wiley, 2009.

[16] Joo Siang Tan, Say Leng Goh, Graham Kendall, and Nasser R. Sabar. A survey of the state-of-the-art of optimisation methodologies in school timetabling problems. *Expert Systems with Applications*, 165:113943, 2021.

Part II

Appendixes

Appendix A

Additional content to the main document

A.1 Javadoc.zip

This file contains the full Javadoc documentation of the project compiled to HTML.

Because of the amount of classes it is recommended to check this documentation before checking the source code.

A.2 SourceCode.zip

This files contains the whole code of the prototype application, as well as the single class used to generate the test instances.

A.3 ProblemInstances.zip

All instances used in Chapter 11 are grouped here.

A.4 Application.zip

This contains the executable file, the configuration files, a sample instance and a sample output. Moreover, a “readme” file with the command to run on the console is provided.

This .zip is intended to allow the user to easily test the application. Check Chapter 12 to get a full explanation of how to use the application.