FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

GPU-based timetable generation

Ricardo Filipe Pereira Neves

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Pedro Amaral Rodrigues

July 20, 2016

GPU-based timetable generation

Ricardo Filipe Pereira Neves

Mestrado Integrado em Engenharia Informática e Computação

July 20, 2016

Abstract

Throughout an academic year, educational institutions need to generate hundreds of different timetables, this complex task demands a considerable amount of time and human resources.

In the past, timetable generation was handmade, in current days as this task complexity increased, it is performed by specialized software which allows to reduce time and costs.

A GPU, is a very capable piece of hardware with two particularly important features: memory bandwidth (GB/sec) and raw power (GFLOPS). Most of the complex and difficult computational problems fall into these categories and timetable generation problem is no exception.

Since timetable generation software does not always take advantage of hardware capabilities to perform parallel computations, this dissertation aims to explore GPU's capabilities to solve this sort of problems.

In this dissertation it is proposed a fine-grained parallel genetic algorithm implementation using Nvidia CUDA framework. All testing stages will use real timetable generation input data provided by FEUP. Since FEUP's timetable generation deals with lessons, students and professors allocation, given the complexity involved in this dissertation only the problem of allocating lessons and students will be addressed.

The experimental tests revealed a positive results when addressing isolated portions of the problem. In these conditions the amount of overlapped lessons was reduced at almost 50% with only 500 generations, and the classroom usage was always improved with more lessons in their preferential or alternative classrooms with a more optimal amount of free seats. The experiment with the complete fitness function improved the timetables but not in a such satisfactory ways as isolated experiments. A comparison with CPU order one crossover execution time was performed and for a population size higher than 230 individuals the GPU performance overcame the CPU's.

Resumo

Ao longo de um ano académico, instituições de educação têm a necessidade de gerar centenas de diferentes horários, esta tarefa complexa requer uma grande quantidade de tempo e de recursos humanos.

No passado, a geração de horários era feita manualmente, nos dias de hoje com o aumento da sua complexidade, esta tarefa é realizada por software especializado permite reduzir tempo e custos.

Uma GPU, é um componente de hardware muito capaz que apresenta dois pontos fortes: largura de banda na memória (GB/s) e poder computacional (GFLOPS). A maioria dos problemas complexos encaixam nestas categorias e a geração de horários não é exceção.

O software de geração de horários nem sempre tira vantagem das capacidades do hardware para executar operações paralelas, esta dissertação tem como objetivo explorar as capacidades das GPUs de forma resolver problemas desta natureza.

Nesta dissertação, é proposta a implementação de um algoritmo genético *fine-grained* usando a *framework* CUDA criada pela Nvidia. Todas as fases de testes irão usar informações reais de geração de horários fornecidas pela FEUP. O sistema de geração de horário adotado pela FEUP lida com a alocação de aulas, estudantes e professores. Dada a complexidade envolvida nesta dissertação, apenas os problemas de alocar as aulas e os estudantes serão considerados.

As experiências realizadas revelaram resultados positivos quando consideradas porções isoladas do problema. Nestas condições, a quantidade de aulas sobrepostas foi reduzida em cerca de 50% com 500 gerações. A utilização das salas de aulas foi também melhorada com mais aulas alocadas nas suas salas preferenciais e alternativas com uma quantidade de lugares livres mais ótima. A experiência que considera a função de fitness completa melhorou a qualidade dos horários, contudo não tão satisfatóriamente como as experiências isoladas. Uma comparação dos tempos de execução entre implementações do operador genético *order one crossover* em CPU e GPU foi feita e para tamanhos da população maior que 230 indíviduos, o desempenho da GPU ultrapassa o do CPU.

Acknowledgements

There are some people that I want to thank because they were essential to turn this dissertation a reality, and without them nothing will be possible.

Firstly I want to thank to the people who contributed directly, professor and dissertation supervisor Rui Rodrigues for proposing this topic and being always present when guidance was needed.

CICA members, Susana Gaio and Elisabete Silva for providing all the required information used in this dissertation and for being always available to answer question related to data interpretation and professor Daniel Silva for exposing and explaining FEUP's timetable generation system organization.

Finally, I would like to thank my parents for giving me this opportunity, for understanding all the time and for supporting me since the beginning of this tremendous adventure.

Ricardo Neves

"First, solve the problem. Then, write the code."

John Johnson

Contents

List of Figures

Abbreviations

CPU	Central Processing Unit
GPU	Graphics Processing Unit
GPGPU	General-purpose computing on graphics processing units
BTTE	Bullet TimeTabler Education
SA	Simulated Annealing
GA	Genetic Algorithm
CUDA	Compute Unified Device Architecture
OpenCL	Open Computing Language
OpenACC	Open Accelerators
HLSL	High-Level Shader Language
GLSL	OpenGL Shading Language
CG	C for Graphics
RWS	Roulette Wheel Selection
TS	Tournament Selection
DNA	Deoxyribonucleic Acid
FEUP	Faculdade de Engenharia da Universidade do Porto
CICA	Centro de Informática Prof. Correia de Araújo
XML	Extensible Markup Language
SVG	Scalable Vector Graphics
NVCC	NVIDIA CUDA Compiler Driver

Chapter 1

Introduction

Since the creation of first civilizations, humans have the need to keep a certain level of organization, which provides a sense of control and predictability. Since ancient times that timetables are built for various reasons, they are synonym of organization, predictability which helps to create routines and task separation providing focus, productivity and confidence.

In today's world timetables can be used for a wide range of organizations, they play a major role for educational institutions, because there is a need for organization and task separation so that all stakeholders at a given time interval know what is the task and where it needs to be performed.

Although with the growth of our civilization, institutions became massive and more complex, the task of generating timetables that fit all stakeholders requirements is not trivial like it were in past, mainly because data proportions have increased and continue to grow over the years.

On educational institutions, buildings became larger with more classrooms to be allocated, the number of courses increased, and the number of stakeholders that institutions embrace become significantly larger. This growth boosts the timetable generation problem complexity, meaning that in theory more time and human resources should be needed to solve it. However at today's demand it is expected to get more in less with less, and an acceptable and suitable timetable needs to be found quickly with lowest costs possible.

1.1 Context

Today's software is getting more complex and resource demanding by after day, companies try to optimize software to be able to execute faster while demanding for less power.

Nevertheless, CPU technology is succeeding on keeping up this raising complexity pace and performance can be maintained or even improved and results can be produced without making the user wait too much time. For this reason, today, a big portion of all developed software is designed to use exclusively the CPU.

Timetable have to be generated by a computer software because it is too much complex for being generated by humans in a useful amount of time. These timetables require the available resources: classrooms, professors, students and time, to be explored at their maximum, in order to serve the raising amount of students.

When a demand for such timetable is present, a fully capable methodology needs to be designed and built. It is required for this methodology to output high quality solutions in a relatively short amount of time before those solutions are needed. For example on FEUP's timetabling system a new solution is needed every semester, although other educational institutions may have shorter time periods to build their solutions.

Since the complexity for building a complete and efficient timetable for large education institutions can be enormous, the need for using another and more powerful device to compute results arises.

This referred more powerful device is the GPU, it is a relatively inexpensive device for the amount of work that it can deliver when programmed efficiently, respecting it's architecture. Timetable generation software, can be considered critical. However GPU approach can help to reduce the risk that educational institutions face if a given timetable is not produced on time.

1.2 Motivation and goals

The timetabling problem is representative of the class of multi-constrained, NP-hard, combinatorial optimization problems [PU94].

Currently there are many solutions that implement many different algorithms which use many different heuristics. Despite that, these solutions usually do not implement GPU massive parallelism techniques. At present, GPU parallelism techniques are far from being fully explored, and in a near future the implementation and development of these techniques can lead to a major advance in this area.

With lacking of GPGPU approaches to solve the faculty timetabling or any general timetabling problem, the use of this type of hardware becomes a point that should be explored.

The fundamental question that should be answered on the conclusion of this thesis is if the use of a GPGPU framework to develop software that runs on a GPU device can be helpful and if it has potential to produce results with the required quality in a possible smaller amount of time compared with CPU approaches.

The main goal is to identify, design and implement a suitable method/algorithm that is able to efficiently tackle the FEUP's case timetabling problem. It should produce positive results that may prove the viability of using GPUs along with the implemented algorithm to solve problems of this nature. If this goals are fulfilled another step was taken to raise the importance of GPGPU field for solving another everyday problem.

1.3 Structure

Aside from this chapter, this dissertation document is structured as follows: Chapter 2 describes and analyzes the most relevant related works that fit in this dissertation's context. It also describes in detail the three main GPGPU frameworks that can be used on this dissertation giving the justification about the chosen technology. Chapter 3, presents the reason why a genetic algorithm was chosen and explains the main concept and operators that GA comprises . Chapter 4, introduces basic timetabling concepts, the provided input data and implementation details. Chapter 5, presents and analyses some experimental results, in order to test and validate the implemented solution. Finally, Chapter 6, draws general conclusions from this thesis implementation and its results, it also presents some possible ways to improve the work in this thesis scope.

Chapter 2

Related works

In this chapter, related works will be presented and analyzed in order to highlight what has already been studied in this domain and the unsolved problems that still exist. Some approaches regarding timetable generation problem, related problems, methodologies and algorithms that could be used will be presented.

The timetable generation problem is studied by the scientific community since 1960, and from this time a wide range of studies were published presenting different approaches to solve it. There are many other approaches that are able to solve related problems that can be applied to this specific problem.

2.1 Simulated Annealing

Over the last few years, several meta-heuristics have been successfully applied to timetabling problems. Simulated Annealing (SA) is a probabilistic method for finding a global optimum of a given function that can possess several local optima, in a large search space.

Several studies that propose SA combined with a new neighbourhood structure were developed. In this context a neighborhood structure, is when from a given solution / timetable it is possible to obtain a new neighbour by performing a sequence of swaps between two timeslots, instead of only one move in the standard neighborhood structure.

In [LZL09] SA was developed and applied together with a new neighbourhood structure in order to solve two typical real-world high school timetabling problems: Italian high-school and Greek high-school timetabling cases. This study [LZL09], presents a method that has two phases, the first phase presents a way to find an initial solution, and the second phase takes this possible solution and tries to optimize it. The authors developed a new neighbourhood structure. From a given solution/timetable it is possible to obtain a new neighbour by performing a sequence of swaps between two timeslots, instead of only one move in the standard neighborhood structure.. These swaps are performed one by one, but whether a swap can be accepted is decided by the SA approach, in which the improving move and the side move which gives a different solution but

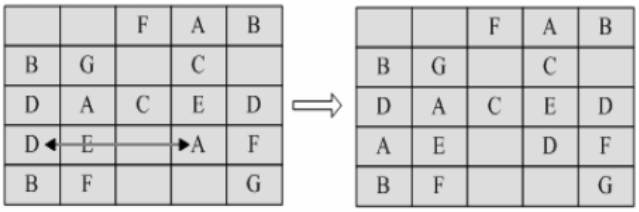

Figure 2.1: Example of a neighborhood structure

with same objective function value, are always accepted, while the deteriorating move is accepted with a probability:

$$P(\Omega_c \leftarrow \Omega_n | \, T, \Delta) = e^{-\frac{\Delta}{T}}$$

Where Ω_c in the new solution obtained by performing a move on the current one Ω_n, Δ is the value of the deterioration of the objective function, T is the current temperature, T parameter has the role of reducing the probability greedy movements over time. The authors believe that selecting all possible swaps between two time slots randomly sorted would keep diversity avoiding dropping into local optima too soon.

Once a feasible timetable is found, the method proceeds to phase 2, where a tabu-search is used alongside SA, in which iteratively a sequence of swaps are generated between a selected time slot and a random one. These swaps are made while maintaining the feasibility of the timetable. A tabu list is used to prevent cycling, and it is a list of solutions to which it is forbidden to move back.

2.2 Genetic Algorithms

Genetic algorithms (GA's), are based on Darwin's theory of evolution [Hol75] and as such iteratively refine an initial population of potential solutions to a problem until a solution is found. The members of the initial population are randomly created. On each iteration of the algorithm a new generation is created by firstly evaluating the current population using a fitness function. After a selection method is used to select parents for the next generation, generally these parents are individuals that have a good fitness value compared with the others. Once the selection method is complete, genetic operators are applied to create offspring for the next generation. The genetic operators commonly used are:

- **Crossover:** Involves two candidate solutions from the population and divides them swapping components to produce new individuals;
- **Mutation:** Involves taking a selected parent from the population and making changes to the content of the individual;

- **Reproduction:** Does not make any change to the candidate and the same individual is carried over to the new generation.

The termination criteria include a solution being found or a set number of generations being reached.

In [Rag13], each element of the population represents a timetable and is comprised of a set of chromosomes with each chromosome representing a period in the timetable. A chromosome consists in two genes, a class timetable and a teacher timetable, and each gene is represented by a two-dimensional matrix with the rows corresponding to time slots and the columns to classrooms.

From a list of requirements, each timetable in the initial population is created by implementing a sequential construction method (SCM). This method allocates tuples according to saturation degree which can be defined as the number of timetable periods that the tuple can be allocated to without causing a clash. Initial population timetables may not respect hard constraints and soft constraints are not considered by the author for simplicity sake.

The näive tournament selection is a method for selecting an individual from a population of individuals in a GA. A number of individuals [GD91] is chosen randomly from the population and the best individual from this group is selected as parent. The process is repeated as often as individuals must be chosen. After the initial population definition, a variation of tournament selection is used to select the parent for each mutation operation.

As in the standard tournament selection t elements of the population are randomly chosen. However instead of the fittest element of the tournament being returned as the parent, each element of the tournament is compared to the current winner, denoted by W, where W is randomly selected in first iteration from all tournament elements. The next element is also randomly chosen from population, denoted by P, and compared with the current winner. From these comparisons, three opportunities arise with $\frac{1}{3}$ of probability each:

1. The fitter of the two participants (W and P) is the winner;

2. The participant P is set a new winner regardless of whether W is better than P;

3. W is declared as the winner regardless of whether P is better than W.

This implementation is looped $t-1$ times. When it ends a winner element is returned, this element will be selected for a mutation operation. The mutation operator performs s swaps between randomly selected clashed rows. In the end if the mutated individual has best fitness than its parent, the parent proceeds to next generation.

This GA implementation was compared with two SA methodologies [RA01], one Tabu search [RA01], one greedy search [RA01] and two discrete Hopfield neural networks [SAD03], and the best overall results were obtained by applying this implementation to solve the school timetable problem.

2.3 FEUP Timetable Generator - BTTE

This study [**?**] [FBP13] presents an approach to solve timetable generation problem carried out by *Bullet Solutions*, which started in 2005 with the goal of building a real and comprehensive model of this problem. By the end of 2006, the model construction was concluded and it was used as the basis for the first version of the *Bullet TimeTabler Education* (BTTE), an automatic and optimized generator of timetables, which has been updated and improved over the years with contributions from almost all the Portuguese schools of higher education as well as some foreign institutions. At present, the BTTE software is successfully used in more than half of Portuguese schools of higher education (universities and polytechnic institutes), including the ten major ones.

2.3.1 Model definition

Since this study targets Portuguese schools of higher education the same as this dissertation, it is important to understand its system in the context of this thesis. An educational institution may offer various **courses**, each course can have multiple **curriculum plans** and each of which is composed of a set of **modules**. Associated with a curriculum plan is a set of **groups** that represents a set of students who follow the same plan and share the same timetable. Figure 2.2 represents a schematic representation of this structure.

Each institution has a range of **classrooms**, each classroom has its own capacity and another set of characteristics. The institutions also have a set of **teachers** who teach all existent modules. The **lessons** represent the way a particular module is taught, from the point of view of the student. A teaching load from one or more **types** and the **temporal space** in which it occurs (weeks) characterize each lesson which can have one or more turns. **Turns** represent how a particular module is taught from the point of view of the institution, that is how many times the same lesson is repeated to a different set of students.

To create timetables for one institution, a set of **events** should be placed in a structure that keeps track of the beginning and the end of each of them - timesheet. Each event is composed by a turn, one or more classrooms and one or more teachers. An event will be placed in a position of the timesheet, with a given duration and occurring in a set of weeks, respecting all existing constraints. The timesheet has a slot size, and an event must be a multiple of the defined slot size.

2.3.2 Proposed algorithm

In the following sections, the proposed algorithm is divided in two distinct parts, each part will be described and explained in detail with some examples, allowing a better understanding of how the presented set of steps and heuristics contribute for finding a quality final timetable.

2.3.2.1 Construction heuristics: building an initial timetable

A sequential heuristic is used to build an initial timetable from an empty timesheet, this heuristics workflow is represented on figure 2.3.

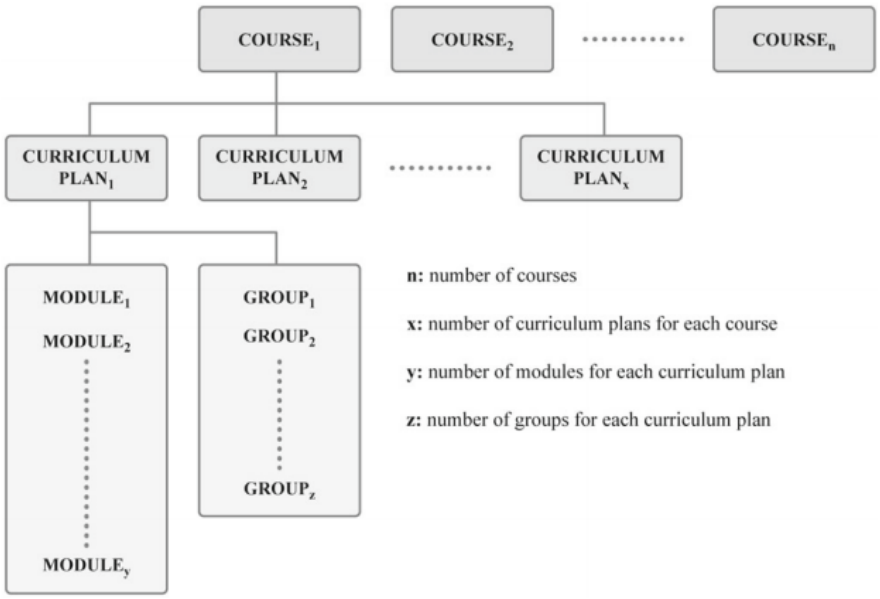

Figure 2.2: Basic concepts of a curricular structure

First there is a list of all events that have not yet been placed on timesheet. This list holds all feasible combinations, so not all elements of this initial list will be placed on the final timesheet. The authors introduce the concept of "ghost" events - events that cannot be allocated because some equivalent ones already did.

For better understanding of this concept the following example is given: if teacher P_a teaches two turns of Lesson Z (composed of turns Z_1, Z_2, Z_3, Z_4), and Teacher P_b teaches the other two turns of the same lesson, and no other restrictions exist, then either teacher can teach any of the four turns, it is only necessary to ensure that each one will teach two and only two different turns. Thus in the list of events to schedule, the following tuples are possible: P_aZ_1, P_aZ_2, P_aZ_3, P_aZ_4, P_bZ_1, P_bZ_2, P_bZ_3, P_bZ_4. From the set of events P_a, two events will be scheduled, and the other two will be transformed into "ghost" events, and the same happens with the set of events of P_b.

Once this process finishes (Step 1 in figure 2.3), and the list of events to be placed is defined, it is necessary to choose which event will be recorded on the timesheet. To perform this selection (Step 2 in figure 2.3) a criterion of urgency was defined, this criterion in based on 4 key points:

- Degree of dependence between events that are associated by means of constraints (the higher the dependence, the more urgent);
- Overlapping of availability of all resources involved in an event, which indicates how many different places the event can be assigned at each moment (the less places, the more urgent);
- Correction of the previous factor by the elimination of potentially feasible places that are no longer feasible because of constraints other than availability (the less places, the more urgent);

- Range of possible classrooms to host the event (the lower the range, the more urgent).

Following the previous selection of the most urgent event to schedule, it is now necessary to evaluate the best location for the event. The choice of the best location is defined based on two fundamental criteria:

- The location with highest quality, in an attempt to reach an initial solution that already meets soft constraints;

- The location that causes the least impact on the remaining events to schedule, avoiding choices that could lead to a limitation of the possibilities regarding remaining unscheduled events.

After an event is scheduled (Step 3 in figure 2.3), the list of events is updated, the criterion of urgency is recalculated and the selection of a new most urgent event is carried out. Although even with a careful choice of the sequence of events to process, by selecting first the urgent ones, and scheduling them by a certain criterion, it is natural that in very restricted problems, as in almost all real cases, sometimes an event that was chosen as the most urgent cannot be fit into the schedule.

After testing, without success, the traditional methods of backtracking to solve this problem, new processing methods were developed. To schedule an event cannot be fit into the schedule, it is necessary to change something in the current solution. Assuming that a feasible solution exists, at least one slot for all events will be found on the timesheet, this means that it is not necessary to cancel events that have already been scheduled, they just need to be moved to other slots in order to free a slot for the problematic event (Step 4 in figure 2.3). As soon as this process (escape methods) ends and a free slot is found, the scheduling takes place and the construction of the initial solution proceeds normally.

2.3.2.2 Improvement heuristics: finding a final timetable

Once an initial solution is found, the optimization phase is initiated. A simplified version of the improvement heuristics workflow is presented on figure 2.4. In authors' proposal, the search for new solution is based on neighbourhood structures, these structures are iteratively constructed in order to find a set of neighbours (solutions with small changes compared with current one). These solutions can be accepted or rejected according to specific parameters, thereby allowing an exploration of solution space.

The optimization algorithms implemented go through three major phases; normal, intensification and diversification, each of these phases is specified to achieve a particular purpose, and together they are the key to a final optimized outcome. Beyond that two different neighbourhood structures were developed, direct exchanges between events and exchanges between events in two steps.

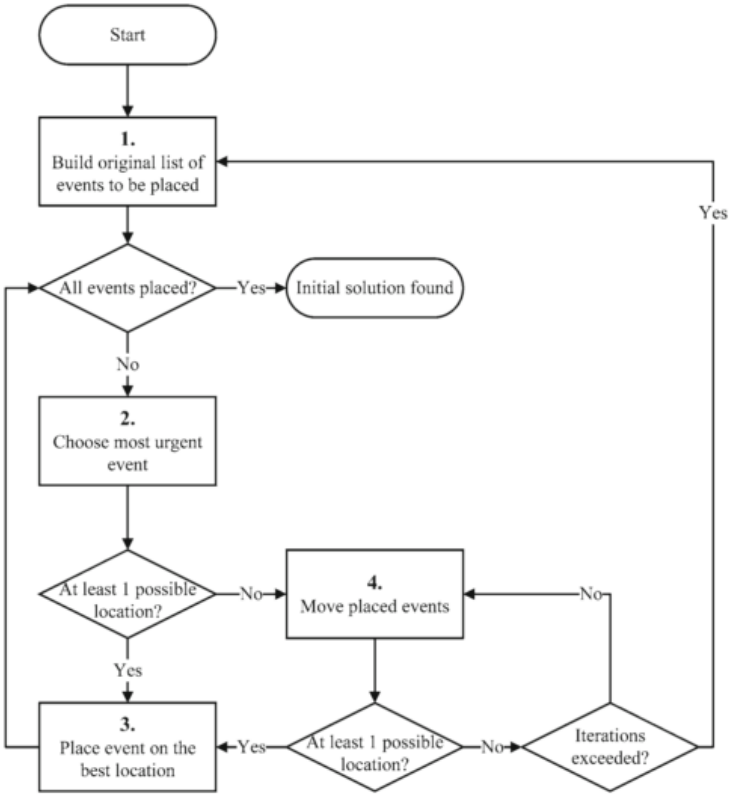

Figure 2.3: Construction heuristics workflow

In the first case, a pivot event is chosen, and, with the support of existing data maps, potential destination slots are selected, these slots can be free or occupied. If they are free, a valid neighbour immediately comes into existence, if they are occupied, the event or events occupying the destination slots will be moved to the original location of the pivot event.

In the second case, the main difference is how the events occupying the destination slot are treated. A more complete assessment of existing overlaps is performed and alternative locations for scheduling, besides the original site of the pivot event, are explored. In the two developed neighbourhood structures, in order to avoid local minima, the existing best neighbour is always accepted (Step 1 in figure 2.4) even if it has lower quality than current solution.

Depending on the search phase, the use of the two specified neighbourhood structures has specific characteristics. In the normal phase, the goal is to perform a faster search and a more free exploration of the solution space. In the intensification phase, the most promising search zones are explored in more detail, this is achieved by performing a deeper search around solutions returned by normal phase. In each of this phases the stopping criterion is applied after a number of consecutive iterations without improving the current solution. At the diversification phase, by means of the temporary adulteration of the objective function (Step 2 in figure 2.4), it is possible to simulate a new starting point for a new search, without losing time needed to create a new solution from the scratch and without losing too much quality. At the end of diversification phase,

the original objective function is reset (Step 3 in figure 2.4).

The optimization process may end on request by the user and, at that time, the best solution found so far is returned, or if the optimal solution to the problem is found ("zero" penalty of the objective function), which in real life scenarios is highly unlikely.

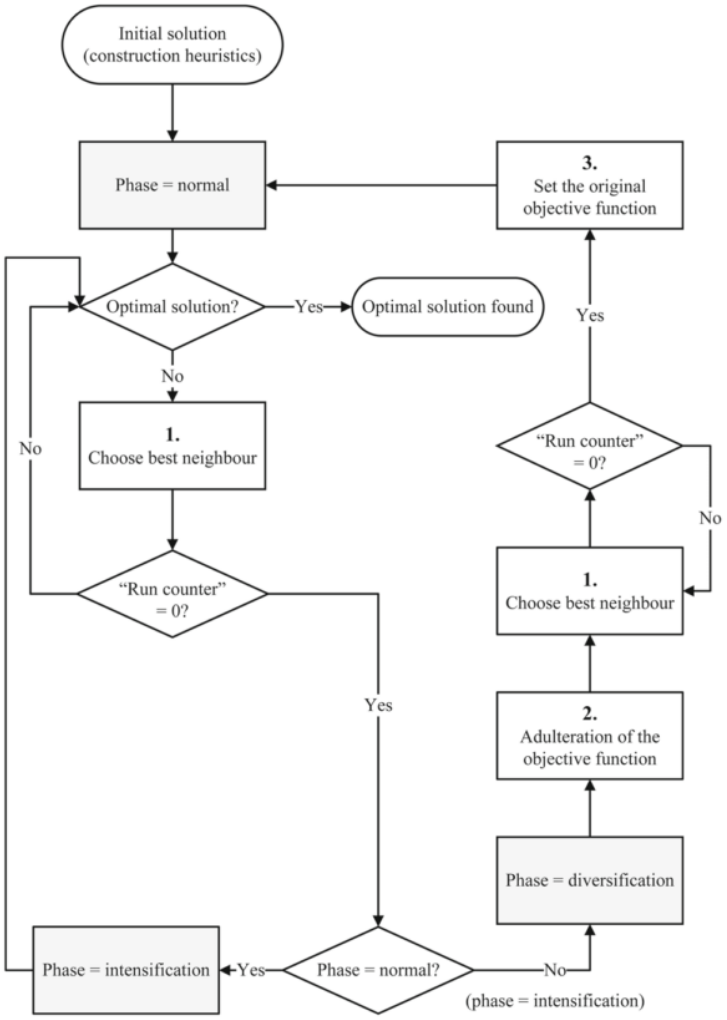

Figure 2.4: Improvement heuristics workflow

2.3.3 Conclusions and results

The authors claim that only with correct modeling of the problem and additional methods of optimization is possible to establish the information base necessary for creating timetables and obtain results that meet users' expectations, both in terms of quality of the solutions and the speed to achieve them.

After analyzing twenty real cases of Portuguese institutions, 85% of time savings were achieved when compared with previous methods. This allows to conclude that the algorithms used in BTTE

application have a considerable level of robustness and are easily adapted to the very diverse real-life scenarios in which they were evaluated.

2.4 GPU implementations

There are some studies that already use GPGPU-based approaches to solve timetabling problems, a recent study [KGG+14] tackles the examination timetabling problem, by using genetic algorithms along with a greedy steepest descent component. Examination timetabling problem is similar to school timetabling problem and methodologies presented in these set of papers can be useful and reused or adapted to tackle this dissertation problem.

In [KGG+14], the initial solutions are created using CPU side as a pre-processing step, and only nearly 250 initial individuals are created.

The operators used in the genetic algorithm for creating a new generation are crossover and mutation operator. Authors consider that the encoding of the chromosomes is of great importance since it affects the performance of the algorithm in terms of speed and quality of the solutions. The chromosomes were encoded using a direct encoding scheme, where each chromosome is an array where each position has a time slot ($T1$, $T2$, $T3$, $T4$, $T5$) which is mapped to a examination ($E1$, $E2$, $E3$, $E4$, $E5$) as it is represented on figure 2.5:

Figure 2.5: Chromosome representation that maps examinations to time slots

The first operation that can be parallelized is the computation of the objective function of each chromosome. This function is composed by two sum operations where each sum operation is independent one from another, here both operations are parallelized on chromosome level and on gene level.

After having the objective function computed for each individual, selection must be carried out, for this a tournament selection is performed along with one-point crossover operator. In this step parallelization is performed again at chromosome and gene level, where in chromosome level, according to their cost, each thread selects two chromosomes and in gene level, each thread produces only one gene (time slot of an examination) for each of the two new chromosomes.

After selecting the most promising individuals, mutation is performed in order to diversify the population. The mutation operator selects some random examination and changes their time slots also randomly. Again this is done at chromosome level, where each thread mutates the genes in a single chromosome, and at gene level where each thread mutates a single chromosome.

The algorithm stops after a specific number of generations or after a specific time limit without improvements.

In order to improve the results, in terms of quality, a local search algorithm was implemented, in this case a greedy steepest descent was implemented and parallelized where each thread calculates the cost of an examination for a specific time slot.

In [ALN13], a thorough survey was made about parallel meta heuristics, both trajectory-based and population-based. They examine many of the applications that parallel meta heuristics apply, the programming models, the available parallel architectures and technologies, as well as the existing software and frameworks.

In [LB08], there is a comparison of a parallel computational model of ant colony optimization of simulated annealing and of a genetic algorithm applied to the traveling salesman problem.

In [PJS10], the authors designed an island model for a genetic algorithm in CUDA software and tested it with some benchmark functions. All of the parts of the algorithms are executed in GPU and the results show a speedup of about 7000 times compared with a CPU single thread implementation.

Moreover in [MBL+09], a genetic algorithm was presented that evaluates its population in the GPU with a speedup of up to 25 times.

2.5 Hyper-Heuristics

Hyper-heuristic is a heuristic search method that automates the process of selecting, combining and generating simpler heuristics to efficiently solve search problems. Since timetabling problem differs from institution to institution, hyper-heuristics have a strong point because with this technique it is possible to generalize a solution that can handle a class of problems, instead of building a solution for just one problem that will not work effectively on problems of the same category.

There is a comparative study [Pil13] that examines the performance of different evolutionary algorithm hyper-heuristics. For each different approach, there is a process of evaluation and selection where for each element of the population is assigned a fitness measure. This fitness measure is computed by using each individual heuristics combination to produce a timetable, and then the quality of each timetable is evaluated.

A selection perturbative hyper-heuristic was implemented by [BBB+12] to find solutions to examination timetabling problem. The hyper-heuristic uses four heuristic selection operators including a mutation operator and five move acceptance criteria.

2.6 Technologies

In early days of GPU programming for general purpose computations, was mainly done using shader languages like HLSL for DirectX [Mic], GLSL for OpenGL [Opeb] or a more high level language like Cg developed by Nvidia [Nvia]. These languages were designed to execute only

graphical operations, although it was possible to map non-graphical computations to graphical operations in order to speed-up the overall process. The development time was very high and only graphical developers with a deep knowledge about GPUs were able to develop such software.

In order to simplify this process, new languages and new frameworks were designed. In this section, three GPGPU technologies will be presented and analyzed allowing to make an informed and conscious choice about the most suitable technology for solving timetable generation problem in this context.

2.6.1 GPGPU Brief History

Parallel computation has gained significant interest in recent years mainly because of the difficulty to increase processor clock speed as result of physical limitations of semiconductor technology. In order to sustain the constant increase of processing power, manufacturers promoted the multicore architectures. GPUs are powerful commodity computer components that include a large number of processing cores in order to provide better performance to computer graphics. Given the available processing power, GPU processors attracted the interest of many researchers and scientists for generic computation, as well, leading to the introduction of term GPGPU.

According to [GPG], GPGPU term was officially introduced in 2002 by Mark Harris when he recognized an early trend of using GPUs for non-graphics applications. Manufacturers recognized that this trend could actually represent a potential solution to increase processing power from computers.

In this section the three considered tools that were mentioned earlier are described and analyzed in more detail. The strong points and weak points will be indicated as well, considering not only the technologies raw characteristics but the problem scope as well.

2.6.2 CUDA

CUDA technology was created by Nvidia [Nvic], its first release dates mid-2007, nearly 9 years ago. This technology was designed with the intent of improving software performance while reducing the steep learning curve that is required to learn a whole new language.

CUDA uses high level languages, compared with shader languages, like C, C++ and Fortran. This code executes on CPU (host), however certain functions or kernels can be invoked from the host to the GPU (device). The kernel code is compiled using the *nvcc* compiler [Nvib] instead of traditional host compiler allowing a faster and integrated GPU software development.

With CUDA it is possible to explore more deeply the GPU capabilities in various science domains, making the development easier for software developers, scientists and researchers.

The main advantages and disadvantages of CUDA framework will be presented below:

+ Has developer-support in one package;

+ Has more built-in functions and features compared with other frameworks;

+ Low level explicit approach, the problem needs to be decomposed into parallel operations and mapped to the hardware;

− Only works on Nvidia GPUs;

− Does not support the full C standard.

2.6.3 OpenCL

OpenCL is an open standard for parallel programming on heterogeneous systems. It was firstly released on mid-2009, nearly 6 years ago and it was initially created by Apple Inc.. However today it is maintained by the non-profit technology consortium *Khronos Group*.

As a standard, OpenCL has available conformant implementations built by highly-regarded companies like *AMD*, *Apple*, *IBM*, *Intel*, *Nvidia* and *Qualcomm*. The standard considers that a computing system may include several different types of computing devices, they may be CPUs, GPUs and other processors or hardware accelerators - heterogeneity.

OpenCL first release used a specially created language called OpenCL C, which is based on C99 standard, but adapted to fit the device model. Currently OpenCL 2.2 supports OpenCL C++ that is based on C++14.

The main advantages and disadvantages of OpenCL framework will be presented below:

+ Has support for many types of processor architectures;

+ Is a completely open standard;

+ Low level explicit approach;

− Only AMD and Nvidia OpenCL drivers are considered mature;

− Provided by many vendors, not provided on a central package.

2.6.4 OpenACC

OpenACC is a portable directive based parallel programming standard. It is available for C, C++ and Fortran, it provides a simple approach to introduce code parallelization without requiring significant programming effort to convert sequential code into parallel code. OpenACC first version was launched on late 2011,

OpenACC uses a high level, descriptive approach, that allows developers to introduce parallelism in the code to the compiler using directives. Subsequently the compiler will be responsible for optimizing the code for the target parallel architecture from a single source code. These code directives specify loops and regions of code to be sent from host CPU to an attached accelerator (CPUs, GPUs and many-core multiprocessors).

The main advantages and disadvantages of OpenCL framework will be presented below:

+ Has support for different hardware brands and architectures;

+ Portable code to several hardware architectures and Operating Systems;
− High level descriptive approach;
− Level of optimization lower compared with low level approaches.

2.7 Concluding remarks

After researching about timetabling problem, there is clearly a gap to be filled. It represents the absence of solutions capable of solving this problem using GPGPU parallelism. Some solutions to related problems present some valuable information and knowledge that can be applied for solving this particular problem.

Since timetable generation software is not often distributed for a large number of users and commonly it will run on a dedicated machine that could be adapted and constructed with the intent to achieve more performance, it is not a requirement to build heterogeneous and portable software.

This dissertation aims to prove a concept and not build a commercial solution. Therefore a dedicated implementation to a single restricted group of GPU hardware is ideal allowing to redirect all effort and time spent to improve the efficiency and performance at its maximum.

OpenACC is a framework that was designed to provide portable software that just needs to be adapted to parallelism once. After, based on inserted directives the compiler will use them to provide the most suitable optimizations for the hardware where the software is compiled. Since that in this dissertation the main concern is about performance and reduce computing time, the use of a high level framework is not the most suitable since all improvements and optimizations will be built by compiler which may not be able to achieve produce such an efficient software when compared with other low level programming approaches.

For concluding both CUDA and OpenCL are suitable frameworks that could be used to develop a timetabling software that fits in this dissertation scope. However, CUDA will be the framework of choice mainly because it is a more mature tool. This will help to reduce the learning curve slope, allowing to develop parallel sections in a faster and more conscious way, which will provide more real develop time.

Chapter 3

A genetic algorithm for timetable generation

In this chapter the reason why the genetic algorithm was chosen for solving this problem will be explained. Afterwards the base genetic algorithm will be explained in more detail as well as several different types of genetic operators.

3.1 Base Algorithm choice

It is essential, and considering the analyzed related works, to select the most appropriate algorithm and methodology to solve the timetabling problem and simultaneously use CUDA massive parallelism architecture.

Approaching the faculty timetable generation problem using FEUP as context and using CUDA parallel framework was never done earlier. However, considering the analyzed works, more specifically [KGG+14], the author was able to parallelize a genetic algorithm implementation along with a local search algorithm (greedy steepest descent), to improve results in terms of quality after genetic algorithm stops improving the solutions.

Since that all genetic algorithms operations are highly parallelizable, and it was proven that this type of algorithms are capable of solving similar problems, this dissertation will consider the development of a genetic algorithm while using CUDA parallelism capabilities.

3.2 The basic genetic algorithm

Since it was decided that the a genetic algorithm implementation was the most suitable option to tackle this dissertation's challenges, this section introduces the genetic algorithm concept as well as its work basis.

A genetic algorithm is a search heuristic based on evolution and natural selection concept found in nature and first published by Charles Darwin in 1859 [Edi]. It is a variant of a larger class named evolutionary algorithms that uses mechanisms inspired by theories like survival of the fittest and biological evolution. In fact in nature this set of theories work together, and are capable to produce along millenniums living beings that are progressively more adapted to their environment.

Starting with a more familiar nature approach, each individual has its own *genetic code* which is composed by *genes*, this defines the individuals inner and outer characteristics. The individual genetic code is called *genotype* and the consequence of the *genotype* is the individual itself which is called *phenotype*. When two individuals reproduce, they share their genes, and create children that ends up having the genes of the two parents combined. Environment external factors or even DNA failure to copy accurately, can cause a mutation on an individual's genetic code. This phenomenon is just a small change in one individual's genetic code, usually it just affects a single gene.

Individuals live inside an environment in which all of them compete to spread their own genes to the next generations. However some individuals may not be able to reproduce, mostly because *evolutionary pressure* or *selective pressure* are exerted over the entire population. With this phenomenon comes the *natural selection* or *selection of the fittest*, where the fittest individuals and best adapted to the environment are more likely to survive and spread their genes to forthcoming generations. The unfit individuals are more likely to die and leave the next generation without its non-adapted genes. This process is repeated over millions of years, which leads to the evolution of the individuals, according to the environment where they are inserted.

This nature approach can be mimicked by a computer algorithm, so it can optimize a population of individuals (solutions) generation over generation, selecting the fittest ones generation after generation until the individual quality obtained is desirable. In section 3.3 the genetic algorithm workflow and the main operators involved will be analyzed more deeply.

3.2.1 Chromosome encoding

The chromosome should contain encoded information about the solution it represents. Before starting developing a genetic algorithm it is crucial to find a way to represent any potential solution to the problem. Any representation is valid although is preferable a simplistic representation.

For sake of simplicity in the next few examples, the chromosome will be encoded as a binary string. As an example a chromosome representation is represented in figure 3.1

Parent 1	1101100100110110
Parent 2	1101111000011110

Figure 3.1: Binary string individual representation

3.2.2 Crossover

Crossover selects genes typically from the two parents chromosomes and recombines them creating new offspring. This operator aims to introduce *exploitation* on the searching system, so that some solutions inside the population will be exploited to generate, hopefully, a better one. In a genetic algorithm crossover operation has often an associated probability to occur. A crossover probability of 1.0 or 100% means that all selected individuals are used to create new offspring, not preserving any parents to the next generation. If the crossover probability is less than 1.0 or 100% it is possible that parents are preserved unchanged (apart from any mutation operations) into the next generation.

There are many crossover operations variants. All crossover operators described bellow will consider that two parents create new offspring and they are not preserved for the next generation.

3.2.2.1 One point crossover

One Point crossover splits both parents by a single crossover point, all the data beyond that point is swapped between the parents, creating two new children.

Parent1	1101100 | 100110110
Parent2	1101111 | 000011110
Offspring 1	1101100 | 000011110
Offspring 2	1101111 | 100110110

Figure 3.2: One point crossover representation

3.2.2.2 Two point crossover

Two point crossover considers two points on both parents and all the genetic material between them is swapped.

Parent1	1101 | 10010011 | 0110
Parent2	1101 | 11100001 | 1110
Offspring 1	1101 | 11100001 | 0110
Offspring 2	1101 | 10010011 | 1110

Figure 3.3: Two point crossover representation

3.2.2.3 Cut and splice

Cut and splice, can be considered a one point crossover where the crossover point is different on the two parents, which may create two children with different genetic material length.

Parent1	110110 | 0100110110
Parent2	11011110000 | 11110
Offspring 1	110110 | 11110
Offspring 2	11011110000 | 0100110110

Figure 3.4: Cut and splice crossover representation

3.2.2.4 Order one crossover

This is a crossover method for permutations, this means that it performs crossover among two different parents without, the resulting children, having repeated elements. This can be useful in certain problems where the previous crossover methods fails to give an admissible solution.

This operator can be divided in four simple steps:

- Choose an arbitrary swath from the first parent;
- Copy this swath to the correspondent child, maintaining the parent's position;
- Copy the genes that are not in first child, from the second parent to the first child, starting right from the cut point of the previous copied part, in the parent 2 and in child 1, copy each gene respecting their order;
- Analogous for the second child, with parent roles reversed.

Figure 3.5: Order one crossover representation

3.2.3 Mutation

Mutation attempts to introduce some random alteration of individual genes. It maintains genetic diversity and prevents the algorithm of getting stuck on a local extreme, allowing the algorithm to *explore* the surrounding search space.

Mutation, like crossover, often has an associated probability, and it can change according to the problem that is being solved. The value of this probability is crucial because if it is zero, there are more chances to fall into a local extreme because the solution space is not being *explored*. On the other hand if the probability is too high, each individual will be almost random and no inheritance from parents will be present, achieving an approximate form of brute-force search.

Original offspring 1	1101100000011110
Original offspring 2	1101111100110110
Mutated offspring 1	1001100100111110
Mutated offspring 2	1101011100110111

Figure 3.6: Mutation representation

3.2.4 Selection

Selection operators give preference to the most fit or better solutions, allowing these solutions to spread their genes to the next generation. Some selection methods are able to control the *selection pressure*, in order to allow unfit individuals to spread their genes to the next generation in order to maintain the *population diversity*. For determining which solutions are the best, each solution has a *fitness score* associated to it. This score is calculated by the *fitness or objective function*. The fitness score of each individual is commonly represented by an integer or a floating point value that qualifies each solution.

3.2.4.1 Roulette wheel selection

This selection method is also known as *Fitness proportionate selection*. It is a genetic operator that gives each individual *i* of the current population a probability *p(i)* of being selected proportional to its fitness *f(i)* where *n* denotes the population size:

$$p(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)} \tag{3.1}$$

The *RWS* can be implemented according to the following pseudo-code:

Algorithm 1 RWS pseudo-code

```
1:  procedure RWS
2:      for each individual do
3:          sum_all_individuals_fitness ← sum_fitness()
4:          random_number ← random[0, sum_all_individuals_fitness]
5:          i_sum ← 0
6:          j ← 0
7:          do
8:              i_sum ← i_sum + f(j)
9:              j ← j + 1
10:         while i_sum < random_number and j < population_size
11:         Select the individual j
```

3.2.4.2 Rank selection

The RWS selection method analyzed above has a well-known problem: if the best individual fitness is much higher than remaining individuals, they will have very few chances for being selected which can compromise variability and lead to premature convergence.

Rank selection is a variant of RWS that is based on the rank of the individuals rather than on their fitness value. Selection probability is proportional to relative fitness rather than absolute fitness like the RWS. Ranks are based on the absolute fitness, the worst individual will have the

rank 1, the second worst rank 2, and the best individual will have the rank *N*, where *N* represents the population size.

In figures 3.7 and 3.8, it is noticeable that the individuals with less probability of being selected for crossover on RWS have more possibilities of being selected on rank selection. However the probability of fitter individuals being selected decreases, balancing the selection process to avoid premature convergence towards a local extreme.

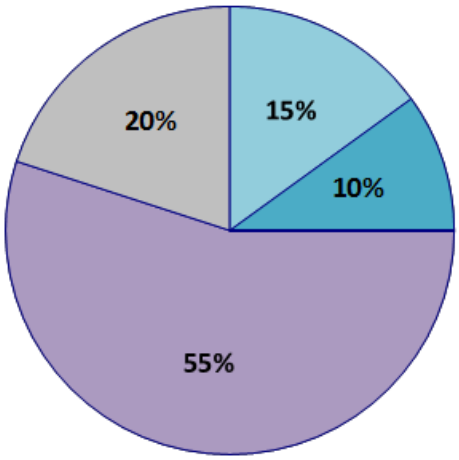

Figure 3.7: RWS individuals probabilities of being selected

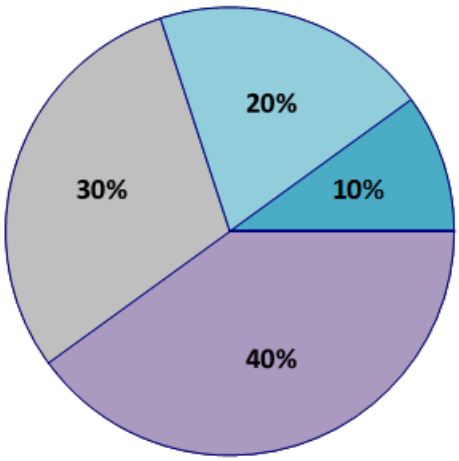

Figure 3.8: Rank selection individuals probabilities of being selected

3.2.4.3 Tournament selection

The idea of this selection method is to create a series of *tournaments*, with smaller size than population's, among the entire population. Random individuals are chosen with equal probability to enter a tournament. When the tournament is full, the fittest individual is the tournament's *winner* and it is selected to perform the crossover operation.

There are many variants of tournament selection but the basis is always the same and can be adapted to fit the problem context. In the algorithm 4 it is assumed that the population size is preserved across generations and *tournament_size* variable is an user defined variable.

Algorithm 2 TS pseudo-code

```
1:  procedure TS(tournament_size)
2:      num_individuals ←0
3:      for each individual do
4:          num_elements ←0
5:          tournament[ ]
6:          selected[ ]
7:          while num_elements <tournament_size do
8:              random_individual ← random[0, population_size]
9:              tournament[num_elements] ← population[random_individual]
10:             num_elements ← num_elements + 1
11:         selected[num_individuals] ← best_individual(tournament[ ])
12:         num_individuals ← num_individuals + 1
```

3.2.5 Elitism

Elitism involves copying a small group of the fittest / elite individuals, directly and unchanged, into the next generation. In some situations elitism can have a significant impact on performance as it eliminates the possible need of rediscovering previously discarded individuals, which costs computation time. Usually elite individuals are copied unchanged to next generation but they are still eligible for selection as parents for possible crossover and further mutation.

3.3 Genetic algorithm pipeline

Generically, a genetic algorithm has always the same sequence of operations. Before beginning the loop, a *new random population* is created and *evaluated* by the fitness function.

After performing these two steps, the algorithm starts a sequence of genetic operations in loop: first the parents which were evaluated previously are selected for *crossover* according to the crossover probability, exchanging each other's genes to create the children for the next generation. These children are *mutated* according to the mutation probability. Lastly each individual of the new offspring is *evaluated* in order to support the selection operator placed in the beginning of the loop.

The genetic algorithm interrupts the search when certain conditions are met. These conditions may vary from implementation to implementation. Generally the most used stop conditions can be triggered by an *execution time* or *number of generations* created limit, a reasonable *fitness value* is reached and *best fitness individual is stall* across a determined number of generations. Some implementations can even use a combination of some of these stop conditions or implement

different ones that suit the problem that is being solved. In figure 3.9, and in algorithm 3, is represented the pipeline that was described above.

Algorithm 3 Genetic algorithm pseudo-code

```
1:  procedure GENETIC ALGORITHM
2:      generation ← 0
3:      population ← create_initial_population()
4:      population ← evaluation(population)
5:      while not done do
6:          generation ←generation + 1
7:          population ←select_parents(population)
8:          population ← crossover(population)
9:          population ← mutation(population)
10:         population ← crossover(population)
11:     individual ←get_fittest _individual(population)
```

In this chapter, the possible technologies/frameworks and algorithms/methodologies were analyzed in more detail in order to choose the most suitable combination, where the algorithm could take the advantages of the technology to produce a quality solution in an agile way.

Two crucial but informed decisions were taken in order to maximize this thesis potential to solve the purposed problem in a more suitable and efficient way.

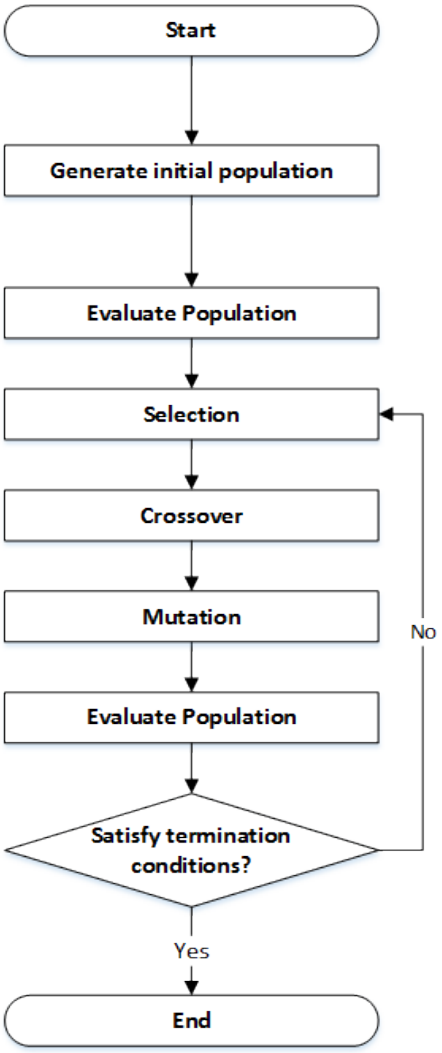

Figure 3.9: Genetic algorithm flowchart

Chapter 4

Methodology and Implementation

4.1 Context

In this dissertation, all data with real timetable related information was provided by *Centro de Informática Prof. Correia de Araújo* (CICA) placed in *Faculdade de Engenharia da Universidade do Porto* (FEUP).

Since the provided data will be used to support this dissertation, it is essential to introduce its structure and characteristics. It is also relevant to state that this data was actually used by FEUP system to generate classrooms, classes and professors timetables for the entire faculty on 2015/2016 school year, first semester.

The provided data is in XML format and it is a set of files that aggregate and combine information in a relational way. In section 4.1.2, these files will be explained in more detail.

4.1.1 Concepts

Before introducing the provided data and since it uses concepts that may not be completely clear for readers that are not familiar with school timetabling problem, these concepts will be first introduced to make the following sections more understandable.

- **Class**: A class represents a group of one or more students;
- **Discipline / Course unit:** Represents a study of a particular subject that is composed by one or more lessons;
- **Lesson:** Represents a particular lesson, that belong to a course unit. It is qualified by its typology and other parameters;
- **Turn**: A turn is composed by one or more classes which attend to the same lesson. The lesson is taught in a common classroom with common professors.
- **Repetition**: Represents the number of times per week that a given lesson is repeated per turn;

- **Curricular plan / Syllabus**: Represents the group of disciplines/course units that compose a given course;
- **Course / Cycle of studies**: Represents a particular academic education degree that is available for students to attend.

4.1.2 XML data

A brief description about each individual provided XML input data will be presented in table 4.1.

File	Description
AreasCientificas.xml	Lists all the scientific areas that coexist in FEUP ecosystem.
Caracteristicas.xml	Lists all classroom features present in entire FEUP campus.
Categorias.xml	Maintains information about different groups/categories of professors that may be present in FEUP's ecosystem.
Cursos.xml	Holds information about each course/cycle of studies available for FEUP's students.
Disciplinas.xml	Holds information about all the existent course units in FEUP.
Edificios.xml	Lists all the buildings that compose FEUP faculty.
FileAulas.xml	Lists all the lessons that must be taught referent to all faculty courses.
PlanosCurriculares.xml	Holds information about each course's syllabus in FEUP's system. There is one entry for each year of each syllabus.
Professores.xml	Contains information about each professor that reside in FEUP's ecosystem.
RestricaoObjectivo.xml	Contains information about existent restrictions and objectives that may exist and be considered for timetabling construction process.
Salas.xml	Lists all the classrooms that are available for teaching lessons on FEUP's infrastructure.
Tipologias.xml	Lists all the typologies that a lesson may belong to.
Turmas.xml	Lists all the classes that exist inside FEUP's ecosystem.

Table 4.1: XML input files description

In FEUP's currently used timetable generation software documentation there are several many more files that are certainly used in the real world. However these files are not relevant for this dissertation purpose and therefore are not focused here.

We will focus on relevant parts namely, **FileAulas.xml**, **Turmas.xml** and **Tipologias.xml**, described in B.

4.2 Implementation

The software was implemented in C++ and CUDA C/C++ using Microsoft Visual Studio 2013 and it can be separated in three logic parts:

- Read, parse and store XML provided information;
- The genetic algorithm itself;
- The Exportation of timetables to SVG.

Those logic parts, were developed from scratch, and they will be described next in sections 4.2.1, 4.2.2 and 4.2.3.

4.2.1 Read, parse and store XML provided information

In this stage all the needed files are parsed and stored in a previously constructed data structure designed to accommodate this type of data. To simplify the XML files parsing, a light-weight C++ XML processing library named *pugixml* [pugb] was used.

In figure 4.1, is shown the designed data structure's representation. Green containers refer to classes that were designed specially for being used on CUDA architecture. Their size was reduced at the maximum, each class instance has a unique identifier, and the associations between different classes are made by id and not using the object itself which will represent a waste of *Device* memory . This approach helps to reduce the amount of GPU global memory needed to store all the information used for subsequent parallel computations. Blue containers refer to classes that were created without the intent of being used directly on GPU.

This is the first task that is done when the software starts running, all the XML files are read and parsed sequentially. The data defined to be used directly on GPU, is stored on *Device* global memory, the remaining data that is not used directly on GPU calculations is stored on *Host* memory.

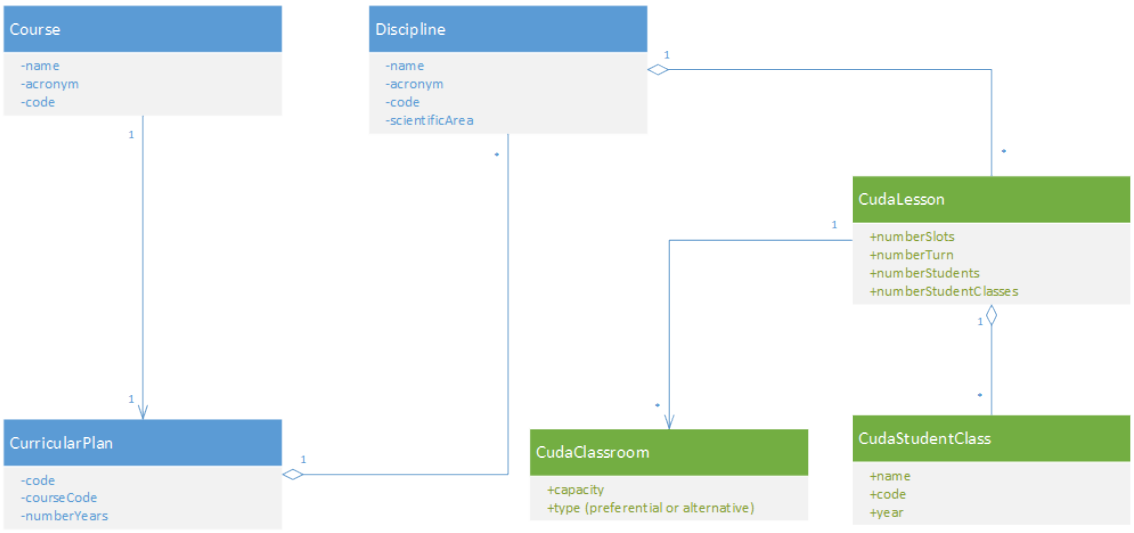

Figure 4.1: Data Structure UML class diagram

4.2.2 Genetic algorithm

In this subsection the genetic algorithm implementation for the timetable generation problem will be explained in more detail. In particular, it will be shown how individuals are represented in memory, the details about how the initial population is created, how the genetic operators were implemented and the required mechanisms to ensure that inconsistencies created on individuals are eliminated.

4.2.2.1 Individual representation

Two types of individuals, with identical structures, were developed: the **classroom individual** represented in figure 4.2 and the **class individual** represented in figure 4.3. Each individual represents an equivalent timetable in an unidimensional array but considering a different view.

While the first individual represents the timetable from the point of view of the classroom, the second one represents the timetable from the point of view of each class/turn.

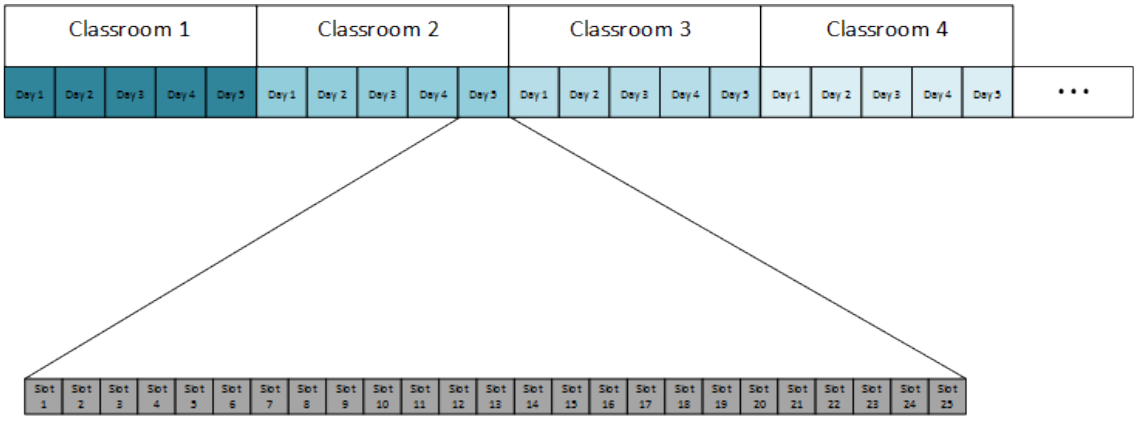

Figure 4.2: Graphical representation of the classroom individual

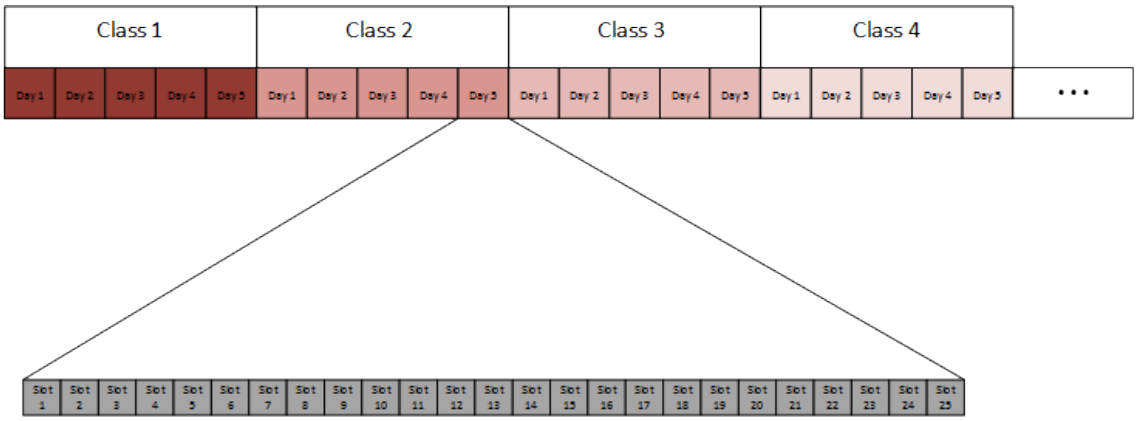

Figure 4.3: Graphical representation of the class individual

Considering both representations, each classroom and each class have five days. These days represent days from Monday to Friday. As it is expressed in figures 4.2 and 4.3 each day includes twenty five slots and each slot represents a time period of 30 minutes, starting at 8am and finishing at 8:30pm. The timetable format is the same as FEUP uses, excluding that lessons may be taught on Saturday. However this is an uncommon case hence it is not considered on this dissertation.

In classroom individual each slot only is allowed to have one corresponding lesson. In other words, it does not considers neither allows time-overlapped lessons. When considering class individual, it is known that it just represents the same information present in classroom individual, but considering a different perspective. Since the genetic algorithm applies all its operators over classroom individual, in order to evaluate the class perspective a conversion from classroom individual to class individual must be made. In this conversion there may be for example two lessons that are attended by the same class in different classrooms, but in equivalent time slots. When this happens, after conversion, the resulting class individual will present in those two slots both lessons listed, which mean that they are overlapped.

4.2.2.2 Initial population generation

The implemented genetic algorithm starts by creating a population of classroom individuals. This process is not totally random unlike the generic versions of genetic algorithms. Next will be presented a few facts that support the use a of non-complete random initial population.

The lessons have a probability, that progressively increments for each allocated lesson, of being overlapped when allocated completely randomly. Since the individual is an unidimensional array, a lesson allocated randomly could be split between two different days in the same classroom or even between two different days in distinct classrooms (Friday of previous classroom and Monday of next classroom). If those two condition are broken, the final timetable is considered invalid, and a *hard constraint* is broken.

In the provided XML data, each lesson has an optional parameter that lists the classrooms where it should be allocated. It is differentiated between alternative and preferential classrooms. Using this information it is possible to create an initial population that considers it, which avoids increasing the computation time in the next algorithm phases on trying to find a solution that may already be present on those recommended classrooms. The placement of the lessons on classrooms present on this list is considered a *soft constraint*, since that if the lesson is not placed in one listed preferential or alternative classroom the correspondent timetable continues to be a valid one.

This process of creating an initial population is executed once in the beginning of the genetic algorithm, thus it was not parallelized and is running fully sequential on CPU. The process starts with a list of all lessons that need to be allocated. In first place, the list of preferential classrooms associated with each lesson is iterated and for each classroom a random slot inside it is chosen. If after the random slot including itself, there is enough space to allocate the lesson (in the same day) and there isn't another lesson occupying the amount of space required, the lesson is successfully allocated. If that lesson cannot be allocated in any preferential classroom, or there are not preferential classrooms associated to it, the same process is taken for all alternative classrooms. Finally

if it was impossible to fit the lesson in one of its alternative classrooms, or if this list is empty, the function tries to allocate the lesson in a random classroom until it succeeds. This process is repeated until all lessons are allocated. This procedure is represent in algorithm 4.

Algorithm 4 Initial population creation pseudo-code

```
1:  procedure CREATEINITIALPOPULATION(lessons, classrooms)
2:      for each individual do
3:          for each lesson do
4:              preferential_classrooms[] ← get_preferential_classrooms(lesson)
5:              alternative_classrooms[] ← get_alternative_classrooms(lesson)
6:              possible_allocation ← false
7:              for each preferential_classroom do
8:                  if allocate_lesson(preferential_classroom, random_slot) then
9:                      possible_allocation ← true
10:                     break
11:             if possible_allocation == false then
12:                 for each alternative_classroom do
13:                     if allocate_lesson(alternative_classroom, random_slot) then
14:                         possible_allocation ← true
15:                         break
16:             while possible_allocation == false do
17:                 if allocate_lesson(preferential_classroom, random_slot) then
18:                     possible_allocation ← true
```

In the end all initial population's individuals respect all the hard constraints and try to respect the soft constraints. This method helps avoiding to increase the computation time on forcing to allocate lessons in their preferential or alternative classrooms, which is the most suitable set of classrooms to host them.

4.2.2.3 Selection operator

The role of a selection operator is often underrated in literature, but without this operator, genetic algorithms are not able to converge towards a global or local optimum. Its implementation allows individuals who have a higher fitness value to have a higher probability of being selected for the next generation. This aspect is crucial because a selection method focuses search in promising areas of the search space. The selection operator aims at exploiting the best characteristics of good candidate solutions in order to improve these solutions throughout generations, which, in principle, should guide the GA to converge to an acceptable and satisfactory solution of the optimization problem [Gol89].

RWS was not implemented because its drawback is the risk of premature convergence to a local extreme/optimum. Premature convergence can happen when there is a presence of a most fitter

and dominant individual which will have a much higher probability of being selected. Rank selection can overcome the RWS premature convergence, but it is possible to lead to a much slower convergence because the best individual could not differ that much from others. Finally the tournament selection appears to be the proper choice, mainly because it is tunable allowing to control the selection pressure. Another strong point is that it can be parallelized, its implementation followed the process described in chapter 3.

The idea behind the implemented GA is to maintain the population size (**N**) throughout generations. Assuming that each tournament chooses the fittest individual, it is mandatory to execute N tournaments. In practice each of these N tournaments can be executed in parallel, since they do not share any dependencies.

An array with random population individual indexes is generated using CUDA architecture, it is allocated one thread for each array position with a total of *tournament size* (**T**) $\times$ *population size* (**N**) threads. After creating this array the main tournament selection kernel is called with **N** threads, each of these threads creates a tournament that uses its random population individual indexes array portion with size **T**. The individual fitnesses are compared and the fittest individual inside the tournament is selected.

4.2.2.4 Crossover operator

The crossover operator allows to each previous selected parents to spread their genes to coming generations. Neither of most usual crossover operators could be implemented because some lessons could appear repeated while others could be suppressed after the crossover operation.

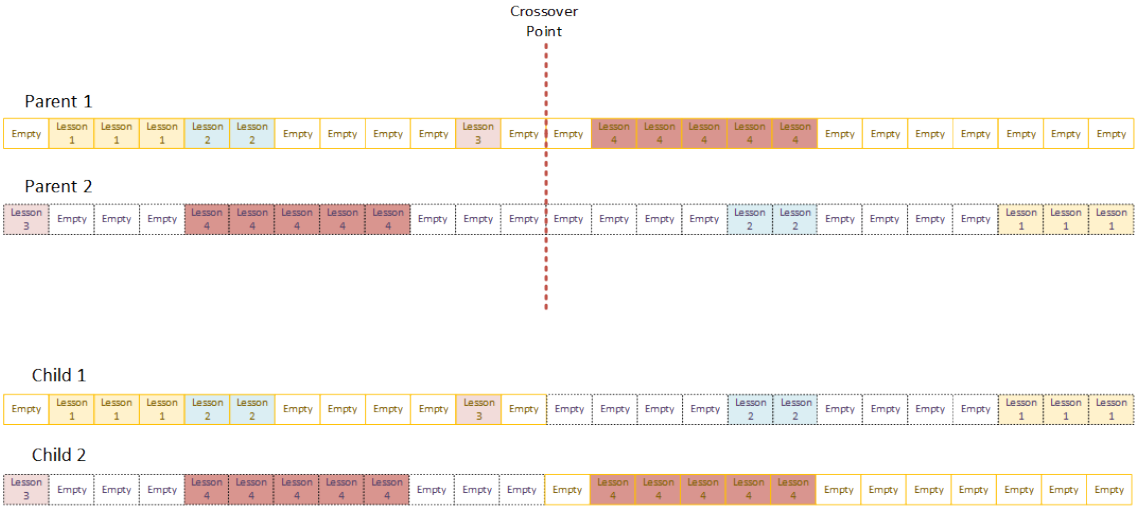

Figure 4.4: One point crossover problem in this context

In figure 4.4 one point crossover was applied on parent 1 and 2, generation child 1 and 2. In child 1, lesson 4 was suppressed and lesson 2 appears twice. In child 2 lesson 4 is repeated twice and lesson 1 and 2 were suppressed. Furthermore, if the crossover point crosses over any

lesson that it will appear incomplete in both individuals. One point crossover was presented as an example, but any technique that consists on unordered crossover like the previous one will fail to produce a valid individual.

In order to overcome this problem, an ordered crossover technique was used. It forces all the individuals to have the same number of non-repeated lessons and empty slots as well. There are several methods that were analyzed that are capable of solving this problem, however the implemented method was *order 1 crossover*. This method copies a portion (swath) of one parent to one child, this swath is then mapped on the second parent and all the genes that were not mapped are copied to that child. This process is repeated for the second child, with parent roles switched.

In figure 4.5 the repeated and suppressed problem is completely solved, however two problems remain. The first problem can be identified in figure 4.5: in child 2 it is possible to observe that lesson 1 was split. In this case it was split within the same day. This happened because it was split on the child 2 extreme, if the individual was longer, it will become split among two different days. This problem is unavoidable in practice and it allows lessons to be split among two days which breaks a *hard constraint* and invalidates the individual. To solve this problem a mechanism described in section 4.2.2.9 was designed and built.

The second problem is eminent, it occurs when one or both crossover swath limits cross over a lesson. This problem is not observable on figure 4.5, but in practice it separates a lesson not between days but inside the same day. To solve this problem a different mechanism (described in 4.2.2.5), similar to the one referred above was created, avoiding another *hard constraint* to be broken.

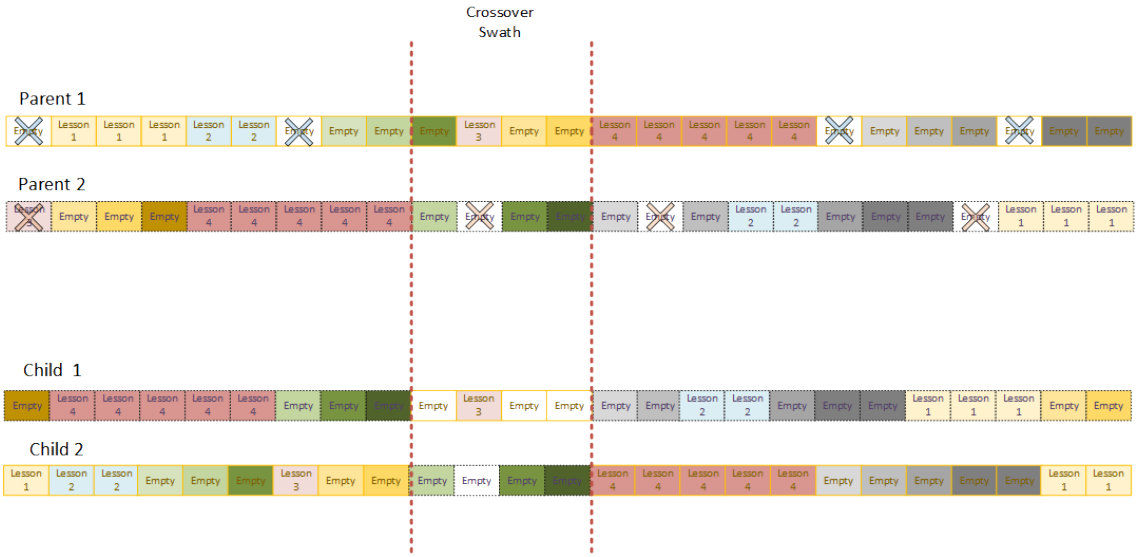

Figure 4.5: Order one crossover

Since the crossover operation is applied in every generation, it is imperative to parallelize it as much as possible, this operation was divided in four different CUDA kernels:

- **Mark the selected swath area on both parents:**

  Initially a random swath location, with fixed and configurable size, is generated for each parent individual using one thread for each individual. This information is sent to the main kernel responsible for marking the slots inside the correspondent swath area, it uses $total\ individual\ slots(\mathbf{T}) \times population\ size(\mathbf{N})$ threads, leaving each thread responsible with one slot.

- **Copy the swath area to children:**

  After the previous kernel ends its execution, this kernel is launched with the same amount of threads as the previous one - one thread per slot. If that slot is marked it is copied for the same correspondent child index, else nothing is done.

  After this kernel finishes its execution, each child owns the correspondent parent swath.

- **Mark equivalent slots on each parent:**

  This kernel is launched using one thread per slot - $total\ individual\ slots(\mathbf{T}) \times population\ size(\mathbf{N})$ threads. It maps and marks correspondent slots of parent 1 in parent 2 and vice-versa. In order to find correspondent slots between parents, each slot is identified by its slot ID when empty. However if a lesson is allocated on it, it is identified by a double key identification: the correspondent lesson ID and the lesson index - the position of the slot within the lesson. Using this identification system is possible to easily identify equivalent slots and mark them between homologous parents included in the population.

- **Copy the unmarked slots from each parent to child:**

  The last kernel is launched with population size (**N**) threads - one thread per individual. In this kernel it is not trivial to predict the position to where a slot should be copied without knowing how many previous slots were not copied before due to not being marked by last kernel. Considering the existence of this intra-individual dependency, the parallelism was only implemented at individual level and not slot level like the previous ones.

4.2.2.5 Swath repair mechanism

Before proceeding for crossover operation, and in order to solve the problem where one or both swath limits cross a lesson, a parallel repair mechanism with one thread per individual was implemented.

For each thread/individual, it is verified if a lesson is crossed by any of the two crossover swath borders. If it is crossed by these limits, the repair mechanism tries to shift up, to early slots. If it is impossible due to insufficient space, it tries to shift down, to later slots. Considering the impossibility to perform both lesson shift operations, the next step is to try to move it to a preferential classroom. Like on initial population generation, a random start position slot is chosen inside that preferential classroom. If it is impossible to place the lesson due to lack of space inside the day where the slot is inserted (it is impossible to fit the lesson until the 25^{th} slot or there is

another lesson which will overlap with the one to be allocated), the next preferential classroom is considered.

If it becomes impossible to allocate the lesson inside any preferential classroom or if this list is empty, the same step is performed for the alternative classrooms. In case of impossibility of moving this lesson to these two lists of classrooms, a random classroom will be chosen and a random slot inside it as well. This step is repeated until it becomes possible to allocate the lesson.

In terms of number of threads, if shift operations were always possible, at least two threads could be launched (one for each lesson), or even the sum of both lessons length, to parallelize at slot level. However since these shift operations could fail to allocate the lesson, it is impossible to have more than one thread per individual because there is the risk of two or more threads to write in the same position at same time. In this case each thread does not have knowledge where the other one is trying to allocate its lesson, and a racing condition may occur.

4.2.2.6 Mutation operator

This operator performs after the crossover kernel finishes its execution and returns the new offspring. Before tackling the mutation problem, its associated probability needs to be considered. In order to accomplish that task, a separate kernel generates an array of random floats between 0 and 1 with $population\ size(\mathbf{N}) \times number\ of\ lessons\ of\ each\ individual(\mathbf{L})$ size, using one thread/position.

The mutation kernel has now all input data that it needs to execute, if the array index float value corresponding to the lesson that is being processed is smaller or equal to the pre-configured mutation probability, a random classroom, a random week day inside it and a random slot inside the randomly selected week day are generated. Afterwards it is tested if it is possible to move the lesson, starting on the previously random selected slot considering the lesson's temporal length. If there is empty and enough space within the random selected position, the lesson can be allocated and the mutation is executed successfully. If it is impossible, the mutation of that lesson is aborted. It is possible to try to mutate the lesson until an available space is found, although by experimenting it was found that trying it again can waste precious execution time delaying the execution time. Aborting the mutation if it does not succeed at first try, decreases the mutation operation and overall execution time. However the predefined mutation probability may be misleading, in the end it is a small price to pay to improve execution time, since the mutation probability can be increased to compensate the aborted mutations.

The parallelism is made at individual level, it is used one thread for each individual, because of intra-dependence between random positions that are chosen to perform the mutation. Like in precious described kernels it may occur a race condition where two or more threads can write the same position.

4.2.2.7 Individual conversion operator

For each generation, two different types of individuals need to be evaluated. As it was said before, these two individual types represent the same information from two different point of views.

This kernel is responsible for converting from **classroom individual** to **class individual**. It is possible to evaluate both individuals without performing this conversion, however it makes the evaluation straightforward and simpler.

The kernel takes advantage of parallelism at class level, there is one thread for each class on each class individual. A higher number of threads is not possible because there may be race conditions present inside each class in each individual. They can occur since one individual class view lessons can be overlapped when mapping directly from classroom view. Unlike on classroom individual, overlapped lessons are considered soft constraints, it should be avoidable, but whenever it happens the individuals/timetables do not become invalid.

4.2.2.8 Fitness function

The fitness function includes both individual views evaluation, in **classroom view** individuals the following aspects were taken into consideration:

- **Which classrooms have lessons allocated:**

  As was described in section 4.1.2, each lesson node on *FileAulas.xml* file has an optional parameter that indicates the **preferential** and **alternative** classrooms. The fitness function has two different parameter scores, one for each lesson inserted in any of its preferential classrooms and another if the lesson is placed in one of its listed alternative classrooms. If the lesson is not inserted in any of them, no score is added to individual's fitness.

- **The classroom seats free rate:**

  The free rate of each classroom with a lesson allocated is calculated using the following expression:

$$Free\ rate = \frac{Classroom\ Capacity - Lesson\ Number\ Students}{Classroom\ Capacity} \tag{4.1}$$

  After computing the *free rate* for each classroom where each lesson is allocated in the considered individual, there are six discriminated levels where different fitness scores can be applied:

  - $0 \leq\ free\ rate \leq 0.05$;
  - $0.05 < free\ rate \leq 0.15$;
  - $0.15 < free\ rate \leq 0.30$;
  - $0.30 < free\ rate \leq 0.5$;
  - $0.5 < free\ rate \leq 0.7$;

- $free\ rate > 0.7$.

It is important to discriminate a significant number of different rates, so that, if needed, the algorithm is able to find a solution for a more refined and detailed list of requirements regarding the desired classroom *free rate*.

- **The classrooms occupancy rate:**

  Classroom's *occupancy rate* is computed using the *occupancy mean*, that can be computed using expression 4.2, and the correspondent lesson's number of total occupied slots.

$$Occupancy\ mean\ (OM) = \frac{Individual\ total\ occupied\ slots\ (ITOS)}{Individual\ total\ classrooms\ (ITC)} \tag{4.2}$$

$$Occupancy\ rate\ (OR) = \frac{Classroom\ occupied\ slots\ (COS)}{Occupancy\ mean(OM)} \tag{4.3}$$

  Like the *free rate*, the *number of classroom occupied slots* are discriminated in five different levels, where distinct scores can be applied to each level:

  - $COS \leq 0.4 \times OM$;
  - $0.4 \times OM < COS \leq 0.8 \times OM$;
  - $0.8 \times OM < COS \leq 1.2 \times OM$;
  - $1.2 \times OM < COS \leq 1.6 \times OM$;
  - $COS > 1.6 \times OM$.

Concerning the considered ranges, it is important to possess a wide enough range of levels in order to offer the possibility to refine how the *occupancy rate* should be on the desired solution. The expression 4.4, represents the fitness function of a **classroom view** individual.

$$Classroom\ individual = \sum_{i=1}^{pcl} P_i + \sum_{i=1}^{acl} A_i + \sum_{i=1}^{tl} F_i + \sum_{i=1}^{tln} FN_i \times 0.2 + \sum_{i=1}^{tc} O_i \tag{4.4}$$

P = score of a lesson that is placed in one of its preferential classrooms
A = score of a lesson that is placed in one of its alternative classrooms
F = correspondent score of a classroom free rate level where a lesson is allocated
FN = number of negative free seats of a classroom with a lesson allocated
O = correspondent score of a classroom occupancy rate level
pcl = number of lessons assigned to their preferential classrooms
acl = number of lessons assigned to their alternative classrooms
tl = total number of lessons
tln = total number of lessons with negative free seats
tc = total number of classrooms

In **class view** individuals only one aspect was considered, the number of overlapped lessons, for each set of overlapped lesson, a user defined value will be added to the fitness function.

$$Class\ individual = \sum_{i=1}^{tol} E_i \tag{4.5}$$

E = score one or more overlapped lessons
tol = number of overlapped lessons

The individual **global fitness function** is the sum of classroom individual and class individual fitness functions.

The fitness function implementation can be divided in two distinct parts: the computation of classroom view population fitness and the computation of class view population fitness.

For computing the classroom view population's fitness it is launched one thread per day for each individual of the population: $population\ size\ (\mathbf{N}) \times total\ individual\ days\ (\mathbf{D})$ threads, each block has D threads and it is assigned to one particular individual.

Inside the kernel two arrays that reside on shared memory were created. Since that in CUDA shared memory, the memory is shared on block level, in this case each of these arrays is shared at individual level as well.

One array uses each position to store the correspondent individual day score. The day score considers if a lesson is on a preferential or alternative classroom and the classroom seats free rate. The second array uses each position to store the correspondent individual classroom occupancy rate. When contemplating execution time, it is crucial to use shared memory when possible because it is much faster, for read and write operations compared with global memory. This approach helps reducing drastically the amount of time spent on memory accesses.

In order to join each day (preferential, alternative and free rate scores) and each classroom (occupancy rate) finesses value, a block level synchronization barrier is used. This barrier synchronizes all threads that belong to a given block, allowing to join each individual portion fitness only when all threads ended their computations. After joining the fitness values of each classroom view individual, it is stored on its correspondent position in a global memory allocated array.

To compute the class view population's fitness, a different approach was followed. The maximum number of threads per block on *Kepler* architecture is 1024. In this particular case, for each individual it will be needed $335 \times 5 = 1675$ threads per block, where 335 is the total amount of classes per individual and 5 the number of days for each class. Considering this restrain, it was impossible to consider one block for each individual like it was done on classroom view fitness computation. The down side of not using a block to process an entire individual is the impossibility of using shared memory to store the fitness score of each particular individual.

In order to overcome this problem, in the most time efficient way, a set of five kernels is launched. With each kernel being responsible for computing the fitness score of a particular portion of each individual. Each kernel of the five launched kernels, is launched with N blocks with $\frac{1675}{5} = 335$ threads and it deals with one fifth of each individual. From here each kernel computes

its portion fitness, using again a block level synchronization barrier to join together the individual portion score of each day in one of the five arrays allocated on global memory (one for each kernel). Each position of those five arrays is reserved for an individual portion total fitness processed in one of those five kernels. After running the five kernels, the resulting five arrays are combined, where the sum of each homologous position represents the class view individual total fitness.

4.2.2.9 Repair mechanism

When the algorithm reaches a stop condition, the solution often is not valid, it may break the hard constraint that states that a lesson may not be split between two different days.

The crossover process does not guarantee that this hard constraint is not broken, and it is not viable to repair the solution on each GA iteration. The constant repair of entire population creates undesired mutations in the individuals and it will misuse valuable GPU computing time to solve a problem that probably will be created on the next generation's crossover operation.

The repair mechanism receives the fittest solution that was obtained at the end of the algorithm. It works with one thread per day, $total\ number\ of\ classrooms(\mathbf{C}) \times number\ of\ days\ per\ week(\mathbf{W})$. Each thread checks if there is a lesson that is present in its assigned week day and in the next thread's assigned week day. In case of being truth, it means that the lesson is split across two distinct week days breaking a hard constraint.

In order to try to respect this hard constraint, the lesson needs to be moved to a place where it has available space fitting inside the same day where it begins. First it is verified if the lesson can be shifted for earlier slots on the origin week day, if in fact this is possible the lesson is shifted the minimum amount of slots needed to not be split. However shifting it for earlier slots may not be possible. In this case, it is tried to shift it completely for the next day. If it is possible, the lesson is shifted the minimum amount of needed slots, starting in the first slot of the next day.

The lessons that cannot be shifted neither to the origin week day, nor to the next day, are not reparable by a parallel approach. This problem must be solved by the classic serial approach. For gathering the lessons that cannot be repaired on the parallel approach, while not losing performance, a shared memory array was created. The shared memory array will store the lessons that cannot be repaired. After all threads of all created blocks finish execution, each block's shared memory array needs to be joined together. The joined information is processed by one thread of each block, the junction is stored into an array allocated on GPU's global memory.

After having the complete list of irreparable lessons copied from *Device memory* to *Host memory* to be serially computed due to race conditions that may occur when trying to move two or more lessons to one or more equal slots. The implemented serial function tries to allocate a lesson on one of its preferential classrooms. In case of not being possible it tries to allocate on one of its alternative classrooms. Finally, if none of the previous options are possible a random classroom is chosen until it is possible to allocate it without being separated among two different days.

4.2.3 Exportation of generated timetables to SVG

A timetable is represented by a 25×5 matrix, however each individual is represented by a one dimensional array and it is not the most user friendly way for presenting and analyzing results. In order to simplify the solutions analysis process, it was crucial to build a more user friendly system that could show timetables in a more natural way.

With this purpose, a SVG exporter was built. It converts a one dimensional individual, with a set of class and classroom timetables to a vector based image (SVG). A sample timetable is presented on figure 4.6.

This exporter generally is called at the end of the GA algorithm, and it exports the two considered timetables views. Since it it only executed once, the exportation is serially executed on CPU. It is exported a distinct SVG file for each classroom and class present on their classroom and class individual respectively.

First a timetable template was created using a SVG editor, marking each slot with an unique identifier. Since an SVG file is represented by an XML specification, it can be easily read and parsed by the developed software. Accordingly with the defined position of a given lesson, using the slot ID and the lesson number of slots, it becomes trivial to represent it on the SVG file.

On the left most column the slots time intervals are represented. On top most row, the correspondent week days are represented. In the center of each lesson block, it is represent the lesson name. On bottom left corner it is represented the lesson typology. In order to easily identify the lesson typology, it is filled by it correspondent color. A white slot means that it is empty and there are no allocated lessons.

Figure 4.6: SVG timetable sample

4.3 Closing remarks

In this chapter, some concepts regarding the school timetabling problem in general were introduced. The provided XML input data and all different implementation parts were described in detail. This implementation architecture is targeted for using CUDA massive parallelism capabilities, in order to increase the chances of achieving better performance. Found problems were identified, and the most suitable solution was found and implemented in order to produce a more solid implementation.

On chapter 5, this thesis implementation will be tested, results analysis will be executed allowing to evaluate implementation's capacity to create a quality solution in a relatively short period of time.

Chapter 5

Testing and results analysis

This section experiments can be divided in four distinct parts: the first part tests the classroom view fitness function portion in isolation, the second part tests the class view fitness function portion isolated. The third part tests the global fitness function and lastly in the fourth part some comparisons and analysis are conducted among CPU and GPU crossover operation execution times.

5.1 Test Environment

All the tests were executed on a desktop computer with the following characteristics:

- **Operating System:** Microsoft Windows 10 Pro 64 bits;
- **CPU:** Intel Core i5-4670K running at 3.4 GHz;
- **RAM:** 8GB of DDR3;
- **GPU:** Nvidia Asus GTX 770 DirectCU II with a base clock of 1058 MHz, boost clock of 1110 MHz, 2GB GDDR5 memory clocked at 7010 MHz and 1536 CUDA cores.

The GPU belongs to Kepler architecture and has compute capability 3.5 and official Nvidia drivers with version 353.90.

The code for the host side, was compiled by Microsoft C/C++ Optimizing Compiler Version 18.00.21005.1 for x64. Device code was compiled by nvcc Release 7.5, Version V7.5.17.

All the tests were executed with the computer in idle with minimum services running in order to keep the performance higher and consistency across test as invariable as possible. Because of the time needed to run a particular test, no multiple tests were executed, the presented results consider just one execution per result.

5.2 Testing Information and Considerations

The information used to run the following tests was provided by FEUP and CICA. Given the problem complexity and in order to boost performance and reduce the amount of Host and Device memory needed, timetables were grouped by their week equivalency. When referring to week equivalency means that there are some consecutive weeks that have the same classroom and class timetables. In that way and with help of CICA, there was found three distinct week ranges where weeks can be agglomerated by their equivalency. The tests were focused on the core period of classes, which corresponds to the peak in terms of concurrent lessons. In the considered period there are a total of 1766 lessons to be allocated on 143 possible classrooms, along with 335 classes.

Some inconsistencies were found on information that was provided. These needed to be fixed in order to get more credible and real results.

There were some classrooms, mainly laboratories, with a declared capacity of zero seats. This information is erroneous and in order to minimize its impact on final results, all classrooms with zero available seats were considered with twenty available seats, since this is an acceptable value for a FEUP's laboratory capacity. Another inconsistency was present on turns definition: all the turns are composed by the same group of classes, an example of this inconsistency is exemplified in the XML fragment 5.2.

```
<Aula>
  <Nome>PL1</Nome>
  <Tipologias>
    <Nome>PL</Nome>
    <NumSlots>4</NumSlots>
  </Tipologias>
  <NumTurnos>4</NumTurnos>
  <Repeticao>1</Repeticao>
  <Turno>
    <NomeTurma>MIB2A</NomeTurma>
    <NomeTurma>MIB2B</NomeTurma>
    <NomeTurma>MIB2C</NomeTurma>
    <NomeTurma>MIB2D</NomeTurma>
    <NumAlunos>16</NumAlunos>
  </Turno>
  <Turno>
    <NomeTurma>MIB2A</NomeTurma>
    <NomeTurma>MIB2B</NomeTurma>
    <NomeTurma>MIB2C</NomeTurma>
    <NomeTurma>MIB2D</NomeTurma>
    <NumAlunos>16</NumAlunos>
  </Turno>
  <Turno>
    <NomeTurma>MIB2A</NomeTurma>
    <NomeTurma>MIB2B</NomeTurma>
    <NomeTurma>MIB2C</NomeTurma>
```

```
27      <NomeTurma>MIB2D</NomeTurma>
28      <NumAlunos>16</NumAlunos>
29    </Turno>
30    <Turno>
31      <NomeTurma>MIB2A</NomeTurma>
32      <NomeTurma>MIB2B</NomeTurma>
33      <NomeTurma>MIB2C</NomeTurma>
34      <NomeTurma>MIB2D</NomeTurma>
35      <NumAlunos>16</NumAlunos>
36    </Turno>
37  </Aula>
```

As it is possible to notice all four turns are composed by the same classes: **MIB2A**, **MIB2B**, **MIB2C**, **MIB2D**. This information does not make much sense. If the intention is to create four turns that are composed by the same classes, just one turn needed to be specified with repetition of four times per week.

Anyway, since this presents data inconsistencies, at XML parser level, was selected the correspondent class for the turn when the number of turns is equal to the number of classes. If this is not the case and if there was still defined equal turns with more classes compared with number of turns, it is impossible to recognize to which turn a remaining class belongs to. For overcoming this situation, the remaining classes are excluded.

Lastly, some lessons have the total number of students defined as one, since this value is totally unrealistic, the implemented XML parser will automatically store that lesson as having twenty students.

5.3 Tests and Results

In this section performed experiments results and input parameters will be presented and analyzed.

All tests consider a crossover swath size of 71 classrooms, 355 days or 8875 slots. This size was considered because nearly half of each parent genetic material is exchanged for each children.

5.3.1 Testing considering only classroom view individuals

In the following tests, populations of **360 individuals** were considered since it is the maximum amount of individuals supported by testing hardware. Crossover probability, mutation probability and tournament size will vary in order to find the most suitable genetic algorithm parameters according to fitness function ones.

5.3.1.1 Experiment 1

In this first experiment, occupancy rate parameters were not considered, mainly because generally in a real timetable generation not always is necessary to control the classroom lessons occupancy. Since a considerable number of lessons have their preferential and alternative classrooms listed,

the classroom distribution is indirectly considered thus there is no need to add another level of complexity.

Levels	Score
< 0	$0.2 \times free\ rate$
$> 0 \wedge \leq 0.05$	0.5
$> 0.05 \wedge \leq 0.15$	3
$> 0.15 \wedge \leq 0.3$	3
$> 0.3 \wedge \leq 0.5$	2
$> 0.5 \wedge \leq 0.7$	0.5
> 0.7	0

Table 5.1: Fitness function free rate scores

Parameter	Score
Preferential Classroom	1.0
Alternative Classroom	0.5

Table 5.2: Fitness function classroom scores

Apart from the added scores to fitness function regarding the level of free seats presented on table 5.1, for each lesson, the amount of free seats is added to fitness function. The considered fitness function can be defined as:

$$\sum_{i=1}^{pcl} P_i + \sum_{i=1}^{acl} A_i + \sum_{i=1}^{tl} F_i + \sum_{i=1}^{tln} FN_i \times 0.2 + \sum_{i=1}^{tl} FS_i \tag{5.1}$$

P = score of a lesson that is placed in one of its preferential classrooms
A = score of a lesson that is placed in one of its alternative classrooms
F = correspondent score of a classroom free rate level where a lesson is allocated
FN = number of negative free seats of a classroom with a lesson allocated
FS = number of free seats on a classroom where a lesson is allocated
pcl = number of lessons assigned to their preferential classrooms
acl = number of lessons assigned to their alternative classrooms
tl = total number of lessons
tln = total number of lessons allocated in classrooms with negative free seats
tc = total number of classrooms

It is important to note that the expression 5.1 is similar to expression 4.4 where the occupancy rate member was eliminated as it it not considered in this test. However a new member was added, each lesson's amount of free seats in the classroom where it is allocated. It is relevant to estimate the theoretical maximum of the defined fitness function for allowing to evaluate the genetic algorithm solution quality comparing it with its maximum.

The total amount of students considering all lessons is 62171, and the total number of available seats in all FEUP's available classrooms is 5663.

Since there are 143 available classrooms, it is possible to compute an approximation of available seats/classroom: $\frac{5663}{143}$. With the total amount of lessons, it is possible to compute an approximation of students/lesson: $\frac{62171}{1766}$. With those two values, it is possible to compute the mean of free seats per lesson, $\frac{62171}{1766} - \frac{5663}{143} = 4.4$. The free rate of each lesson allocated in a classroom can be computed as $\frac{4.4}{39.6} = 0.1111(1)$, this value enters on the third free rate level with score of 3. The maximum theoretical approximated value that the fitness function can achieve with this problem considered parameters is presented in expression 5.2.

$$\sum_{i=1}^{pcl} 1 + \sum_{i=1}^{tl} 3 + \sum_{i=1}^{tl} 4.4 = 14053 \tag{5.2}$$

pcl = number of lessons assigned to their preferential classrooms
tl = total number of lessons

There is a total of 1766 lessons to be allocated, where 985 of them declare their preferential classrooms. There are no lessons that declare one or more alternative classrooms without declaring preferential ones. So the ideal is to place the 985 lessons on their preferential classrooms.

In figure 5.1, tournament size parameter will be tested while leaving the other parameters immutable, crossover probability is maintained at 70% and mutation probability is maintained at 10%.

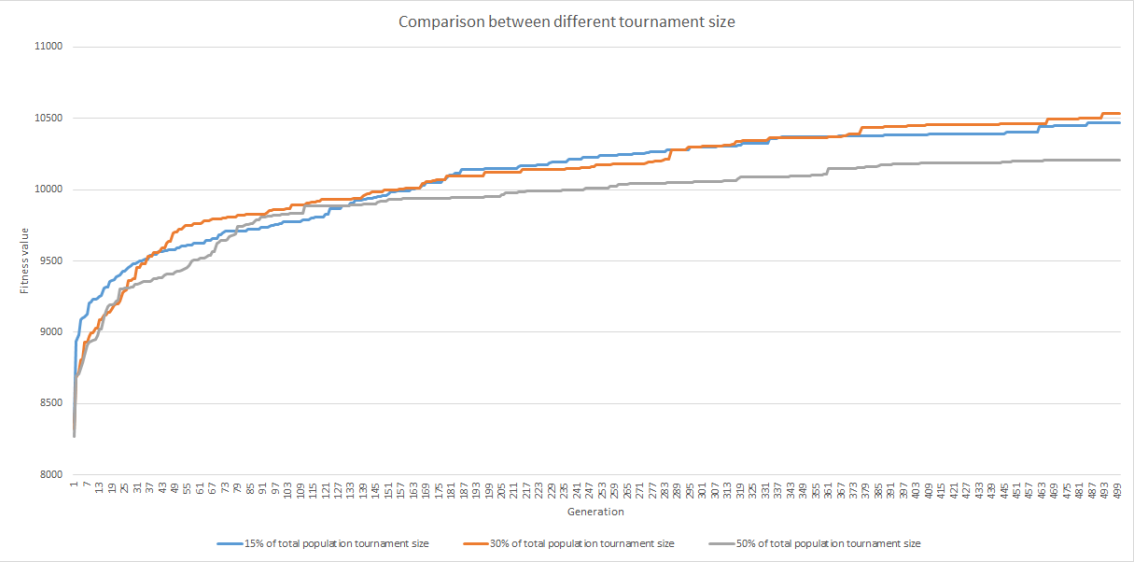

Figure 5.1: Fitness function evolution over 500 generations with different tournament sizes

In figure 5.1, three different tournament sizes were tested, **15%**, **30%** and **50%** of the population size, which gives tournament sizes of **54**, **108** and **180** respectively.

For tournament size with 15% and 30% of the total population size, results were very similar, with a maximum of 10470.4 and 10533.4 fitness values respectively. On the other hand, for 50% approach the fitness value maximum was only able to reach 10210.0 fitness value.

With tournament selection operator it is simple to adjust the *selection pressure*. With a bigger tournament size the *selection pressure* increases because there is a larger chance of having one of the fittest individuals inside the tournament. A smaller tournament size offers more chances of less fit individuals being selected.

In this experiment, the idea was to vary the *selection pressure* in order to find the most suitable value. Starting with the worst result, 50% applies too much *selection pressure*, which causes the population to lose variability leading to a smaller fitness score in the end. Because 15% and 30% fitness function scores were very similar, it is possible to assume that this problem is not very sensible to *selection pressure*. In the end 30% has a slight advantage and produced a fitter individual, in the same amount of time and generations.

Since that the most suitable **tournament size** was found, in the next experiment this parameter will maintain a constant value of 30%, as well as a constant crossover probability of 70%, while mutation probability will vary. The results are presented in figure 5.2.

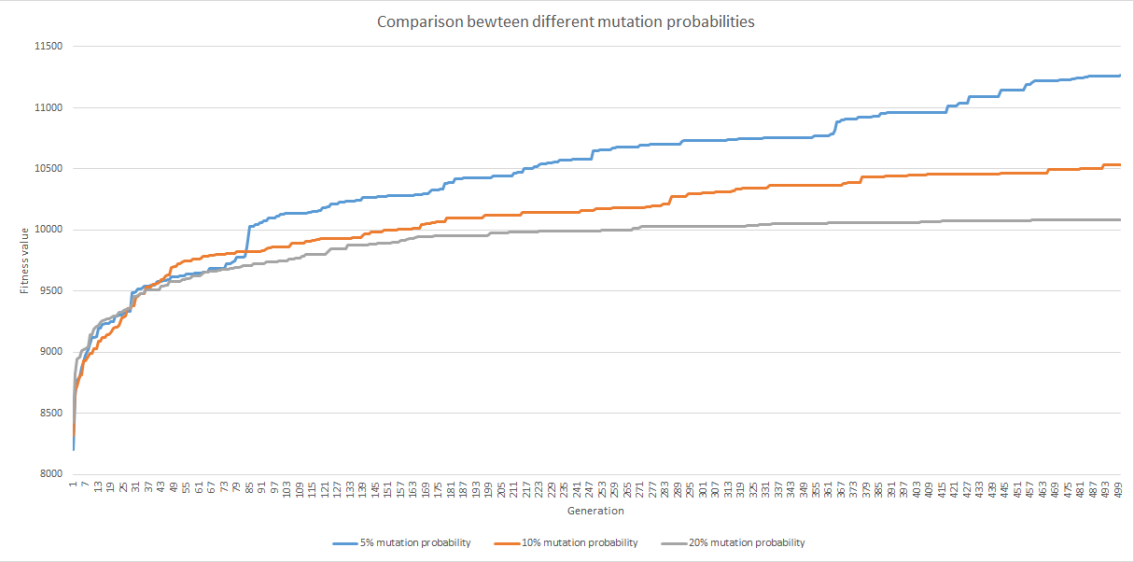

Figure 5.2: Fitness function evolution over 500 generations with different mutation probabilities

Mutation operation allows the solution to diverge by occasionally mutating one or more members of the population. This operation pulls the individual out of a possible local extreme allowing it to potentially discover a better local or even global extreme. The mutation probability plays a big role on conducting the genetic algorithm path. If it is too high, the individuals will become almost completely random with a weak divergent pattern. On the other hand, if it is too small, the individuals will likely converge to a local extreme without exploring another possible spaces that could offer better solutions.

For the three values, three distinct results were obtained, starting with the worst, 20% induces too much randomness and even with a fitter initial population individual compared with the other two mutation probabilities, a worst individual was obtained with fitness value of 10082.6. 10% mutation probability achieved a better result, with a final fitness value after 500 generations of 10533.4. Lastly the mutation probability of 5% was the most suitable, it induces the right amount of randomness that allows to reach a better balance among *exploration* and *exploitation*.

Since that the most suitable **tournament size** and **mutation probability** were found. In next test, both parameters will be maintained constant with values of **30%** and **5%** respectively. In this final test, the crossover probability will vary. The experiment results are presented in figure 5.3.

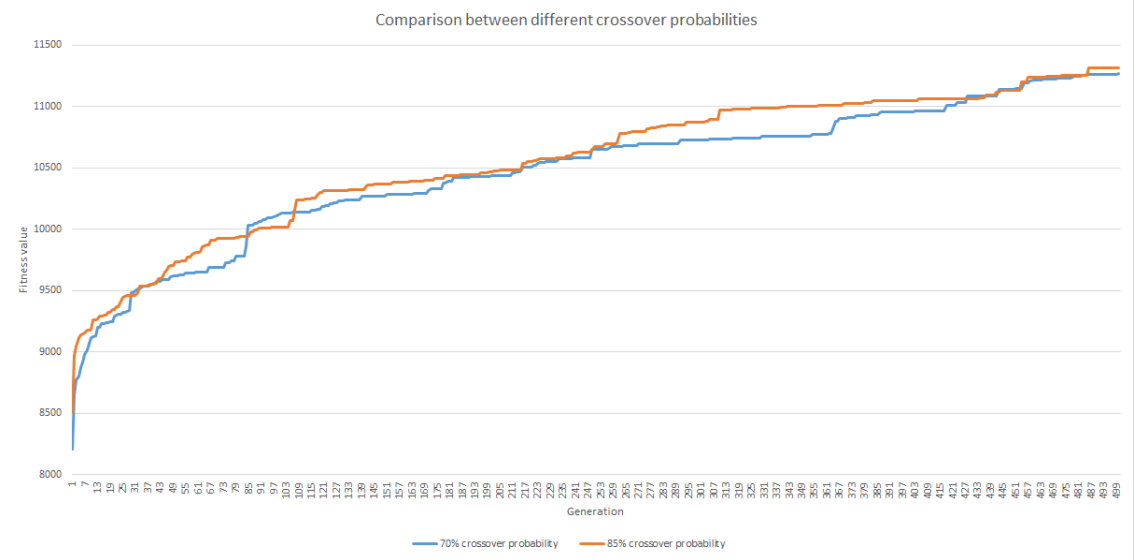

Figure 5.3: Fitness function evolution over 500 generations with different crossover probabilities

Crossover operation allows the solution to converge, pulling the population towards a local optimum. The probability associated to it modulates the convergence over a local optimum. This parameter plays an important role, finding the most suitable value contributes to increase the chances of finding an admissible solution.

For crossover probability of **70%** and **85%**, the final fitness values were very similar. This difference cannot be considered significant for two different reasons. First because the fitness values after 500 generations were 11269.2 and 11317.4 which are very close to each other. And secondly because the best initial population's individuals fitnesses were 8207.8 and 8502.0, whose difference is larger than the actual final fitness values.

It is important to analyze the fitness achieved, however it is crucial to present the actual results. Since it only was considered classroom view parameters, it is only relevant to show the comparison between *phenotypes* classroom timetables of generations 0 and 500 with the considered most suitable parameters within this test set: **30% of the total population tournament size**, **5% of mutation probability** and **85% of crossover probability**.

In figures 5.4 and 5.5, a comparison between the generation 0 and 500 of **B001 classroom** is presented. This classroom has a total capacity of 184 seats, every lesson in both timetables written with blue text represents a lesson with B001 classroom marked as preferred or alternative on input data. At the bottom right corner there is indicated the input data information about how many students are taking that particular lesson. If the value is written in red, it means that the value on input data had value 1 and the new default value is set to twenty as it was mentioned above.

Figure 5.4: B001 classroom initial population

In figure 5.4, it is noticeable that only six of twenty nine lessons are not placed neither in one preferred nor on a alternative classroom. This six lessons do not have many students to justify their allocation to such a big classroom, they are just wasting space that may be useful for other lessons.

In figure 5.4, the lessons that are allocated on B001 classroom and have it as a preferred or alternative classroom (name written in blue), have a big amount of students, and totally justify their allocation in this classroom, although, for example lessons **TSIN(2015-2)**, **LAIG(2015-2)** and **PROG1(2015-2)**, have a small amount of students to justify their allocation in this classroom. This is happening because their number of students was incorrect. However they can have a large amount of students enrolling them and justify that way the allocation in a larger classroom like B001.

Horas	Segunda	Terça	Quarta	Quinta	Sexta
08:00 - 08:30				DACO(2015 - 2) TP1 33	LAIG(2015 - 2) T1 20
08:30 - 09:00		LCOM(2015 - 2) T1 168	LTW(2015 - 2) T1 160		
09:00 - 09:30					
09:30 - 10:00					
10:00 - 10:30	M II(2015 - 2) T1 20		MDIS(2015 - 2) T1 200		RCOM(2015 - 2) T1 147
10:30 - 11:00					
11:00 - 11:30					
11:30 - 12:00		MDIS(2015 - 2) T1 200			
12:00 - 12:30					
12:30 - 13:00			LSDI(2015 - 2) TP1 141	LSDI(2015 - 2) TP2 141	
13:00 - 13:30	AM I(2015 - 2) TP1 22	SCON(2015 - 2) T1 62			
13:30 - 14:00			CPIN(2015 - 2) T1 220	QMP(2015 - 2) T1 167	PROG1(2015 - 2) T1 20
14:00 - 14:30		ECAC(2015 - 2) TP1 22			
14:30 - 15:00			PEST(2015 - 2) T1 120		
15:00 - 15:30	EDA(2015 - 2) TP1 24				TSIN(2015 - 2) T1 20
15:30 - 16:00			IELE(2015 - 2) TP1 40		
16:00 - 16:30					
16:30 - 17:00					
17:00 - 17:30		DT(2015 - 2) TP1 20			
17:30 - 18:00					
18:00 - 18:30	FISI2(2015 - 2) T1 180		TSIN(2015 - 2) T1 20	QMP(2015 - 2) T1 167	LSDI(2015 - 2) TP1 141
18:30 - 19:00					
19:00 - 19:30			PEST(2015 - 2) T1 120		
19:30 - 20:00					
20:00 - 20:30					

Figure 5.5: B001 classroom generation 500

In generation 500, the fittest individual's B001 classroom, have almost the same amount of lessons that do not have this classroom as preferred or alternative (5) and a smaller amount of total allocated lessons (26). This means that the solution did not get better, it may got worst.

There is an explanation for this phenomenon. When analyzing input information, the data about where the lesson should be placed (preferential or alternative classrooms) is more accurate when compared to the lessons number of enrolled students. The score about the free seats will be often wrong in reality because some lessons student number are erroneous and lessons that should be placed on this classrooms that have a large number students end being placed in smaller classrooms, because in theory they fit there.

The mistake on these tests, is that the fitness function parameters were not balanced by their relevance to the problem. It is essential to balance the fitness function scores of each parameter according to its relevance. In this test set, it was given more importance to free seats rate of a given lesson allocation on a classroom than the recommendation where that lesson should be placed. This decision produced individuals that had higher fitnesses, but not a real higher quality in the end.

If the input information about the number of students of each lesson was right this solution had potential to work, although changing all this information manually is unfeasible given the amount of available time and the chance of being impossible to get more accurate information.

5.3.1.2 Experiment 2

This test was performed in order to correct the explained problem present on the test described in 5.3.1.1. The considered **population size** is **360** and the new set of parameters is specified on tables 5.3 and 5.4:

Levels	Score
< 0	$0.2 \times free\ rate$
$> 0 \wedge \leq 0.05$	2
$> 0.05 \wedge \leq 0.15$	10
$> 0.15 \wedge \leq 0.3$	10
$> 0.3 \wedge \leq 0.5$	6
$> 0.5 \wedge \leq 0.7$	3
> 0.7	1

Table 5.3: Fitness function free rate scores

Parameter	Score
Preferential Classroom	30
Alternative Classroom	20

Table 5.4: Fitness function classroom scores

In order to balance the importance level of each parameter on the considered fitness function. As in the test described on 5.3.1.1, the free seats related parameters were overvalued, and the parameters related to preferential or alternative classrooms were undervalued. This test tries to balance these two fitness function parameters by increasing the score of most credible data (preferential and alternative classrooms) and decreasing the score of a less credible or important parameter (free seats). These actions allow to redefine the objective function in order to describe the problem more accurately. Expression 5.3 it is specified the fitness function used to perform this test.

$$\sum_{i=1}^{pcl} P_i + \sum_{i=1}^{acl} A_i + \sum_{i=1}^{tl} F_i + \sum_{i=1}^{tln} FN_i \times 0.2 \tag{5.3}$$

P = score of a lesson that is placed in one of its preferential classrooms
A = score of a lesson that is placed in one of its alternative classrooms
F = correspondent score of a classroom free rate level where a lesson is allocated
FN = number of negative free seats of a classroom with a lesson allocated
pcl = number of lessons assigned to their preferential classrooms
acl = number of lessons assigned to their alternative classrooms
tl = total number of lessons
tln = total number of lessons allocated in classrooms with negative free seats

Note that in the expression 5.3, the last member that was present on last test's expression 5.1 was eliminated since it was overvaluing the information relative to amount of free seats. The theoretical function's maximum considering the parameters values presented on tables 5.3 and 5.4 is presented bellow:

$$\sum_{i=1}^{pcl} 30 + \sum_{i=1}^{tl} 10 = 47210 \tag{5.4}$$

pcl = number of lessons assigned to their preferential classrooms
tl = total number of lessons

There is a total of 1766 lessons to be allocated, where 985 of them declare their preferential classrooms. There are no lessons that declare one or more alternative classrooms without declaring preferential ones. So the ideal is to place the 985 lessons on their preferential classrooms. As it was computed in the last test 5.3.1.1, the free rate is 0.1111(1) that fits in third table level with score 10 for each allocated lesson.

The same GA variables were considered, compared with experiment described in 5.3.1.1. In this test, the fitness function evolution pattern and pace were similar. Therefore it is not useful to present and analyze the generated charts. It is important to focus on final fitness function values and on comparison of generated generation 0 and 500 *phenotypes*.

It is worth pointing that the better solution was achieved with a **tournament size** of 50% of the total population size, a **mutation** and **crossover probability** of 10% and 70% respectively. After 500 generation of evolution process, the best solution achieved a total fitness score of 33756.

In order to compare this experiment with the one on 5.3.1.1, the same classroom timetable will be compared on its generation 0 and 500.

Figure 5.6: B001 classroom generation 0

Starting by analyzing the generation 0 timetable of B001 classroom, there are five lessons in this classroom that do not have it as preferential or alternative classroom. Three (**A(2015-2)**, **EMMI(2015-2)** and **TCAL(2015-2)**) of these five lessons do not justify its placement on such a big classroom. Two of them have wrong students information. The partial fitness of this classroom is 678.

Horas	Segunda	Terça	Quarta	Quinta	Sexta
08:00 - 08:30	AMAT3(2015 - 2) T1 63	SHP(2015 - 2) T1 122			CPIN(2015 - 2) T1 220
08:30 - 09:00					
09:00 - 09:30	HECI(2015 - 2) T1 57		LSDI(2015 - 2) T1 141		
09:30 - 10:00				MNUM(2015 - 2) T1 20	LCOM(2015 - 2) T1 168
10:00 - 10:30			M II(2015 - 2) T1 20		
10:30 - 11:00	A(2015 - 2) T1 20				
11:00 - 11:30					
11:30 - 12:00					
12:00 - 12:30					
12:30 - 13:00				ELEC2(2015 - 2) T1 20	
13:00 - 13:30	LTW(2015 - 2) T1 160	EMMI(2015 - 2) TP1 62	AMAT3(2015 - 2) T1 63		
13:30 - 14:00					LAIG(2015 - 2) T1 20
14:00 - 14:30				LSDI(2015 - 2) TP2 141	
14:30 - 15:00			FISI2(2015 - 2) T1 180		
15:00 - 15:30	FOBR(2015 - 2) T1 48	LSDI(2015 - 2) TP2 141			
15:30 - 16:00				TSIN(2015 - 2) T1 20	
16:00 - 16:30					TCAL(2015 - 2) T1 20
16:30 - 17:00			QMP(2015 - 2) T1 167		
17:00 - 17:30	PROG1(2015 - 2) T1 20				
17:30 - 18:00				TCOM(2015 - 2) T1 160	
18:00 - 18:30					
18:30 - 19:00					
19:00 - 19:30		PROG1(2015 - 2) T1 20	MDIS(2015 - 2) T1 200		MDIS(2015 - 2) T1 200
19:30 - 20:00					
20:00 - 20:30					

Figure 5.7: B001 classroom generation 500

After 500 generations, there are still five lessons allocated in B001 classroom that do not have it as preferential or alternative classroom. There is, like in the generation 0 timetable, a total of 21 lessons that are placed on one of their preferential or alternative classroom. It can be concluded that the available seats utilization on B001 classroom were optimized since the achieved score was 701. In 500 generation the score increase was nearly 3,3% when comparing with generation 0. However the score increase in this classroom was bellow the average of the total score increase from generation 0 to 500 that was nearly 15%.

To conclude, it was crucial to balance the parameters associated scores in order to balance their emphasis degree and obtain a significantly more accurate solution.

5.3.2 Testing considering only class view individuals

In this section, the genetic algorithm will be tested considering only class view parameters. In this case the number of overlapped lessons is the only testable parameter.

In the following tests the considered **population size** is **360 individuals**. Like the previous one, this is an isolated test, that considers only one fitness function parameter: the score of a lesson being overlapped by one or more lessons.

Two tests were made, both tests consider a tournament size of 30% of the entire population and 10% of mutation probability. The variable considered in this test is the **tournament size** which assumes 30% and 50% of the population size. On the other hand, the **overlapped lessons score** considered was **-1**. In figure 5.8, the results of these two tests are presented.

In expression 5.5 it is represented the considered fitness function to perform this experiment. This expression, is a portion of the complete fitness function that represents the class view individuals only.

$$\sum_{i=1}^{tol} E_i \tag{5.5}$$

E = score of an overlapped lesson
tol = total number of overlapped lessons

The fitness function represented in 5.5 along with assumed score for each overlapped lesson of -1 represents the total amount of overlapped lessons. It is logical that it is desired to maximize this function to 0. Achieving a maximum score of 0, means that no overlapped lessons were found in the considered individual.

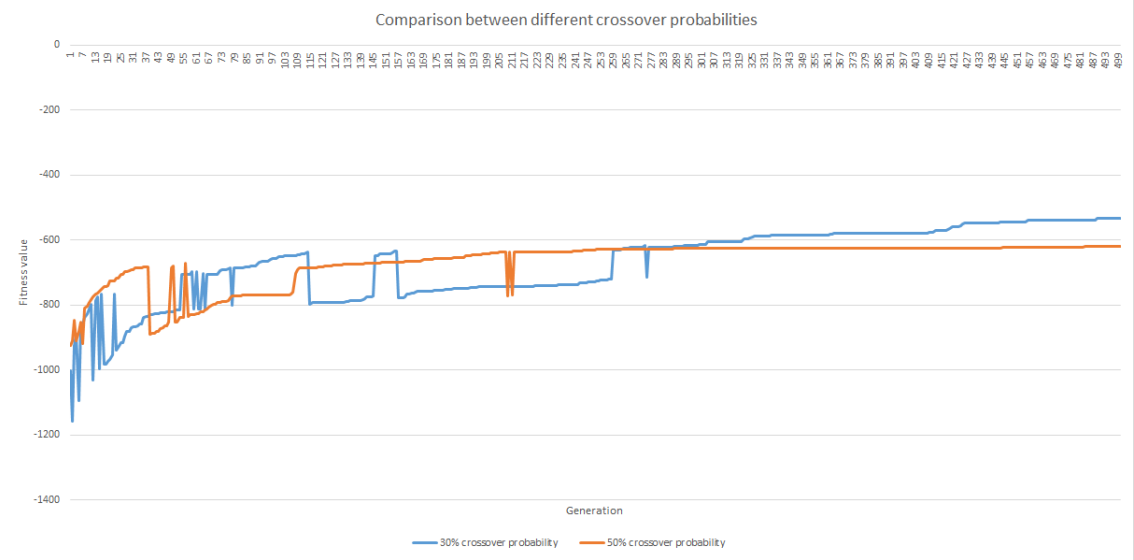

Figure 5.8: Evolution of fitness value over 500 generations

Starting with the test that considers the tournament size with value of 30% of the entire population, the best individual on initial population started with a fitness value of -1003. This fitness score, in this particular case means that in this individual there are 1003 overlapped lessons. The evolution of the fitness function over the 500 generations presents some fitness increase/decrease peaks, however in generation 500, it gives a solution with less 47% of overlapped lessons.

The test with a tournament size with value of 50% of the entire population, was not as effective as last one. It presented some fitness increases/decrease peaks over the 500 generations like the

previous solution. However, due to a higher *selection pressure*, the final result was not as satisfactory as the last one. This test started with a better initial solution with fitness value of -926, and after 500 generations, only 33% of overlapped lessons were eliminated.

Generally both solutions continue to improve over time, although the improvement pace becomes too slow and the progress was stopped due to lack of time since each 500 generation takes about 3 hours to complete.

A recurrent problem on both tests is the peaks of fitness values. This fact is not favorable since it wastes time to find the lost fittest individual, that was not selected for breeding. A convenient solution for this problem is the implementation of elitism, to avoid losing the set of best solutions like it happened in both experiment's tests.

After presenting and analyzing the evolution of the fitness function, it is important to analyze the actual results. In figures 5.9 and 5.10, two classes timetables of the best individual generated by the best test with tournament size with 30% of population size will be presented and compared.

These class timetables belong to *Mestrado Integrado Em Engenharia Informática e Computação* cycle of studies, curricular year 4 and class 5 (**4MIEIC05**). This class has a total of twelve weekly lessons, that should not be overlapped.

Figure 5.9: 4MIEIC05 class timetable generation 0

On generation 0, from a total of twelve lessons, there are three that are overlapped, two **AIAD(2015-2)** theoretical lessons and one **MFES(2015-2)** theoretical lesson.

Figure 5.10: 4MIEIC05 class timetable generation 500

Analyzing the same class timetable on generation 500 *phenotype* (**4MIEIC05**) class represented in figure 5.10, the overlapped lessons were completely eliminated. The final value of the fitness function was -532, which means that in the individual where this class timetable belongs, there are 532 lessons that still overlap.

After executing this set of tests, some conclusions must be pointed out. There is a clear reduction of overlapped lessons when comparing generation 0 and 500. This reduction pace could be higher if elitism was implemented. At the end, even with thousand generations it would be difficult to find a perfect case scenario solution without overlapped lessons and the maximum fitness value of 0.

5.3.3 Testing considering both classroom and class views individuals

The results of executed tests that comprise classroom and class individuals will be presented and analyzed. Both tests present on figure 5.11 were executed considering a **population size of 360 individuals** and **85% of crossover probability** with variable mutation probability and tournament size.

Levels	Score
< 0	$0.2 \times free\ rate$
$> 0 \wedge \leq 0.05$	2
$> 0.05 \wedge \leq 0.15$	10
$> 0.15 \wedge \leq 0.3$	10
$> 0.3 \wedge \leq 0.5$	6
$> 0.5 \wedge \leq 0.7$	3
> 0.7	1

Table 5.5: Fitness function free rate scores

Parameter	Score
Preferential Classroom	30
Alternative Classroom	20
Overlapped lesson	-10

Table 5.6: Fitness function classroom scores

Tables 5.5 and 5.6 show the fitness scores that were considered for running these tests. The classroom individual scores were maintained from experiment 2 5.3.1.2, however to balance the class with classroom parameters, the score of each overlapped lessons was set to -10.

$$\sum_{i=1}^{pcl} P_i + \sum_{i=1}^{acl} A_i + \sum_{i=1}^{tl} F_i + \sum_{i=1}^{tln} FN_i \times 0.2 + \sum_{i=1}^{tol} E_i \tag{5.6}$$

P = score of a lesson that is placed in one of its preferential classrooms
A = score of a lesson that is placed in one of its alternative classrooms
F = correspondent score of a classroom free rate level where a lesson is allocated
FN = number of negative free seats of a classroom with a lesson allocated
E = score of an overlapped lesson
pcl = number of lessons assigned to their preferential classrooms
acl = number of lessons assigned to their alternative classrooms
tl = total number of lessons
tln = total number of lessons allocated in classrooms with negative free seats
tol = total number of overlapped lessons

Expression 5.6, shows the fitness function considered to run this experiment tests. The theoretical maximum is achieved when all lessons that have one or more listed preferential classrooms are allocated in one of them. Given that all lessons that declare an alternative classroom also declare a preferential one, in the ideal solution no lessons are allocated in their listed alternative classrooms. The free rate level as it was calculated in previous tests, is $0.1111(1)$, which fits on the third level with a correspondent fitness score value of 10. In the ideal individual there are no lessons with a

negative number of free seats, since there are enough resources to avoid it. And finally, the total amount of overlapped lessons should be 0. There are only 985 of the total of 1766 lessons that declare a list of preferential classrooms. The theoretical fitness function maximum is presented on expression.

$$\sum_{i=1}^{pcl} 30 + \sum_{i=1}^{tl} 10 = 47210 \tag{5.7}$$

pcl = number of lessons assigned to their preferential classrooms
tl = total number of lessons

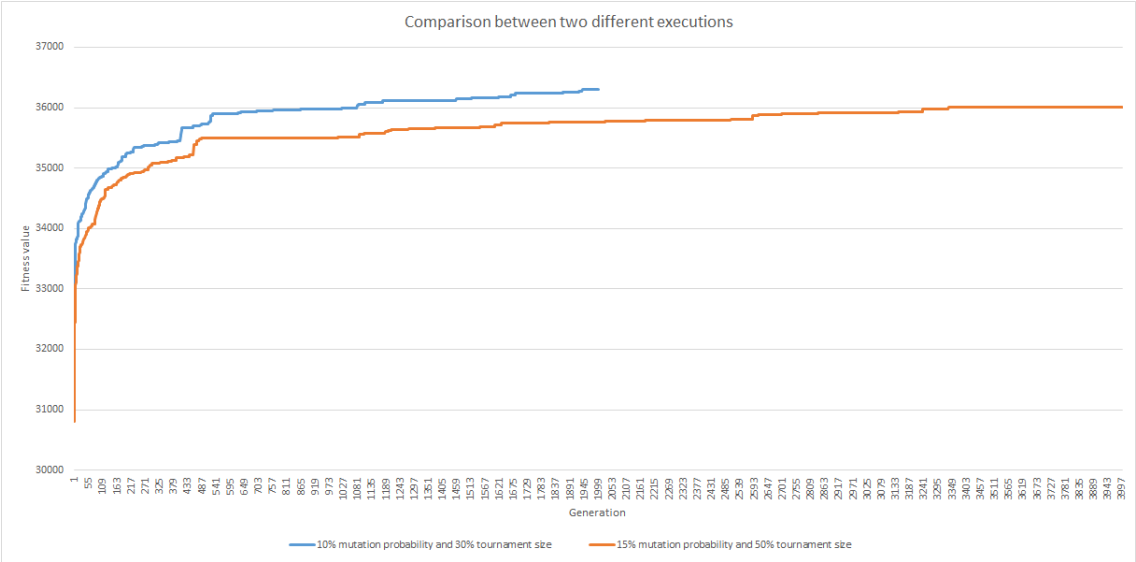

Figure 5.11: Evolution of fitness on two different executions

The test that achieved a higher fitness value was the one that considered 10% mutation probability and 30% of the total population size tournament size. Due to time restrictions, only two thousand generations were produced, with a total execution time of 14 hours. The final achieved fitness was 36301.8.

The second test was executed with 15% mutation probability, a tournament size of 50% of the total population size and four thousand generations were bred. However, even with more two thousand generations compared with the first test, the final results were worse. This test never outperformed the first one in the entire fitness function evolution. After nearly 28 hours of execution time it achieved a final fitness of 36015.8.

Next classroom and class timetables from the test that only executed two thousand generations but achieved a higher fitness will be analyzed.

The best results using a mutation probability of 10% and a tournament size of 30% of the total population size had a **lowest solution exploration flexibility** and a **lowest selection pressure** as

well. This combination of factors, for this problem with the considered fitness function parameters achieved better results.

Figure 5.12: B001 classroom generation 0

Despite that, it is mainly through experimentation that in genetic algorithms, it is possible to discover the best combination of parameters that achieves the best solution in the shortest amount of time. Hence the second test with 15% of mutation probability and 50% of tournament size was executed, for testing if it achieved a most satisfactory solution. However the **selection pressure revealed to be too high** which decreased the population's diversity. Even a larger **exploration flexibility** was not able to keep the population evolving at the desired pace.

The two **B001** classroom timetables represented of figures 5.12 and 5.13 exhibit numerous similarities in terms of their lesson content. The generation 0, has twenty five lessons that are already placed on one of their listed preferential or alternative classrooms. This result is totally expected since the initial population is not completely random generated for starting with a more optimal initial solution. Considering generation 2000, there are twenty seven lesson that are already on one of their listed preferential or alternative classrooms.

The fact that in generation 2000, **B001** classroom timetable there are two additional lessons that have this classroom as preferential or alternative, means that they possibly came from classroom that were not the most suitable to them. Thereby this 2000 generation B001 timetable was improved regarding this aspect.

Figure 5.13: B001 classroom generation 2000

Another relevant information is the number of free seats that each lesson leaves when allocated on B001 classroom. Given the amount of allocated lessons it is not simple to compare this fitness function portion, instead, the total fitness of each classroom will be computed. If the difference between them is higher than 60, it means that the free seats level was improved as well. The previous value is 60, because a lesson allocated in a preferential or alternative classroom gives the individual a score of 30, however in generation 2000 there are more two lessons allocated in one of their preferential or alternative classrooms compared with generation 0 (30×2).

The generation 0 of B001 timetable achieves a total fitness of 846.8, and on the homologous classroom the generation 2000 achieves a total fitness of 912.8. Since $912.8 - 846.8 = 66$, it is possible to conclude that the lessons placement was improved in two lessons leading to an increase of 60 on fitness score. The remaining 6 score points were achieved on increasing the free rate quality of lessons allocated on B001 classroom .

After analyzing **B001** classroom timetable above, the **4MIEIC05** class timetables will be analyzed.

Figure 5.14: 4MIEIC05 class timetable generation 0

In generation 0, from the total of twelve lessons allocated, two of them: **MFES(2015-2)** and **SINF(2015-2)** are overlapping (figure 5.14). Since a satisfactory solution was achieved on generation 0, with the **4MIEIC05** class timetable producing only two overlapped lessons, after two thousand generations one overlapped lesson was found, meaning that the one overlapped lesson was suppressed through GA processing as it is possible to see in figure 5.15.

Figure 5.15: 4MIEIC05 class timetable generation 2000

Both individual views were improved when comparing generation zero with two thousand. However as figure 5.11 proves, the final and better achieved fitness was 36301.8 from a theoretical maximum of 47210. It is known that it may be impossible to reach this maximum, although the final solution fitness is more than 10000 fitness score points apart from it.

The evolution pace started to decrease with the increase of the generations. This aspect can be justified with a possible convergence towards a local maximum, however not even the mutation operator was able to pull the solution out of it in order to allow new search spaces to be explored. If there is not enough genetic variability, an increase on population size should improve it and apply an increase on evolution pace while evolving generations towards the global maximum.

5.3.4 Comparison between CPU and GPU crossover operator execution times

The comparison between execution times of order one crossover operator running on CPU and GPU will be tested and analyzed. The CPU implemented operator is running serial on single *core*. Following the execution times are presented on table 5.7 and on figure 5.16.

	Execution time (ms)	
Population Size	**CPU serial crossover**	**GPU parallel crossover**
10	15	60
20	41	344
100	164	353
200	340	464
300	540	405
360	579	421

Table 5.7: Crossover times comparison

It is possible to observe that for smaller populations, the CPU execution times are smaller compared with GPU's. On table 5.7 the GPU parallel crossover starts to achieve more performance when the population size reaches 300 individuals. Analyzing the two series present in figure 5.16 the CPU execution times tend to increase as the population size increases. On GPU side, for a small population size, the tendency is to increase the execution time as the population size increases. However, when the population size reaches two hundred individuals, the execution time tends to drop and stabilize.

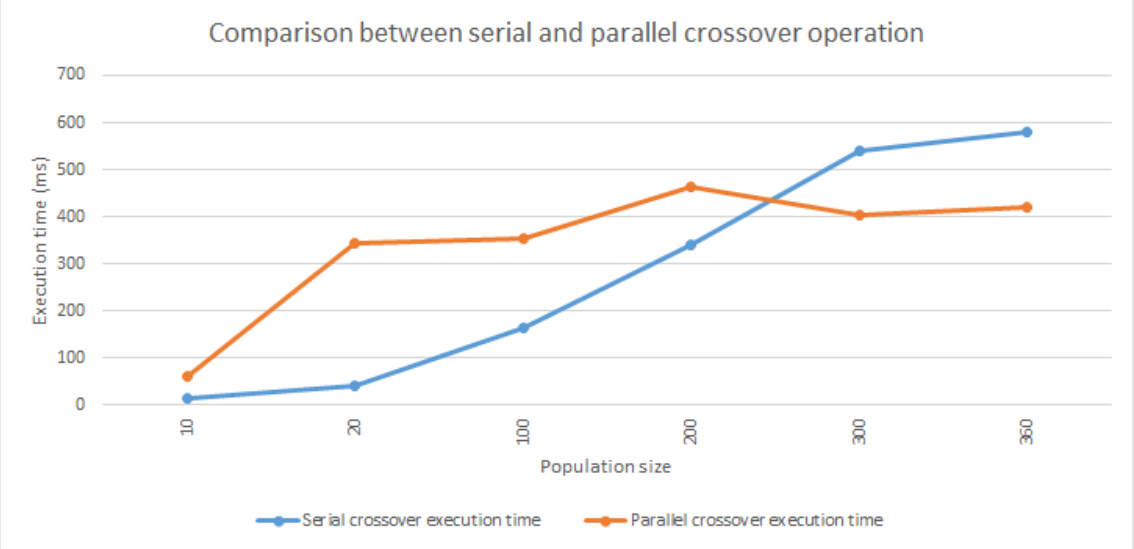

Figure 5.16: CPU and GPU crossover execution times

The CPU execution times will always tend to increase. In this case, only one core is being used and no parallelization is being applied. On GPU side, the data needs to be copied from the *host memory* to *device memory* and vice-versa. Usually memory transfer operations are highly time consuming even for small amounts of data. In theory if a GPU is computing a small chunk of data, the slight advantage that it may have, will be wasted on data transfers operations. However, when a big amount of data is sent to GPU, the data transfers operations consumed time ends up to pay-off the big amount of data that was processed in a faster way.

Execution times, on GPU for 500 generations, take nearly 4 hours of processing time. It was found that execution time grows linearly with the increase of population size, if 1000 generations are considered it will be needed 8 hours of execution time considering classroom and class individual views fitness function parameters. Each generation takes nearly 29 seconds to be generated.

Crossover time are just a portion of the GA process, with a population size of 360 individuals it takes around 0.4 seconds to be fully processed. There are still selection, mutation, fitness computation and swath repair mechanism operators that are executed every generation. Mutation and swath repair mechanism, are simpler operators when compared to crossover, however they present intradependences that do not allow to execute a high amount of threads like it is on crossover operator. This intradependences heavily increase these operators execution times and consequentially the overall generation computing process times are increased too.

5.3.5 Genetic algorithm's operators execution time comparison

The comparison between GA's operators parallel execution times will be performed. For the two performed experiences, **population sizes** of **360** and **100** individuals were considered. The considered **mutation probability** and **crossover probability** was **10%** and **70%** respectively and lastly the considered **tournament size** was **30%** of the total population size. In tables 5.8 and 5.9 are presented the fitness function parameters that were considered for both experiences.

Levels	Score
< 0	$0.2 \times free\ rate$
$> 0 \wedge \leq 0.05$	2
$> 0.05 \wedge \leq 0.15$	10
$> 0.15 \wedge \leq 0.3$	10
$> 0.3 \wedge \leq 0.5$	6
$> 0.5 \wedge \leq 0.7$	3
> 0.7	1

Table 5.8: Fitness function free rate scores

Parameter	Score
Preferential Classroom	30
Alternative Classroom	20
Overlapped lesson	-10

Table 5.9: Fitness function classroom scores

In figures 5.17 and 5.18 the portion of the total generation time that is used by each GA operator is presented. A total of five hundred generations was considered and the arithmetic average of each GA's operators execution time was computed.

For the experiment that considered a **population size** of **100** individuals, the **total setup time** that includes generating the initial population and allocating needed resources on GPU global

memory took around 3 minutes, the **GA execution time** took approximately 66 minutes which gives a **total execution time** of 69 minutes.

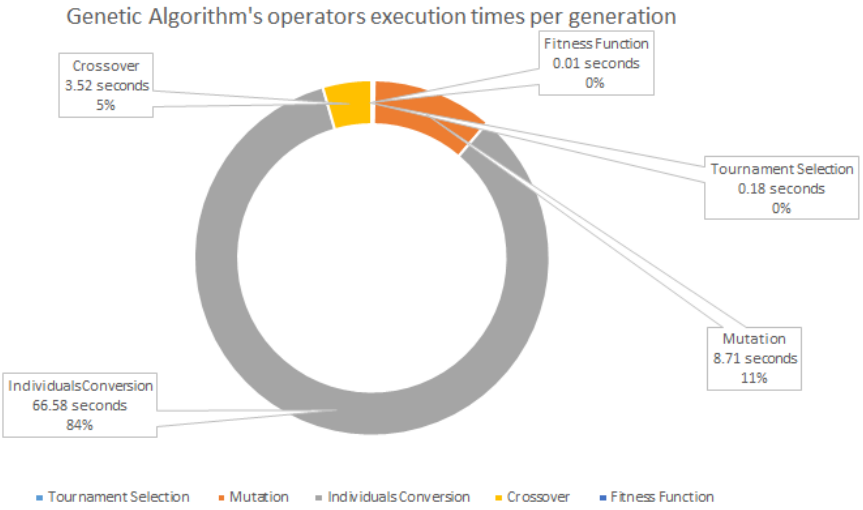

Figure 5.17: GA operators execution times considering 100 individuals

Analyzing the figure 5.17, it is visible that the two operations that use a larger execution time portion are the **individuals conversion** and the **mutation** operators. These two operators tend to use a larger time portion because there are some intra-dependencies that prohibit to raise the amount of threads that can be used which may lead to race conditions.

Operation	Standard deviation (s)
Tournament selection	0.005982
Mutation	0.023439
Individuals conversion	0.039694
Fitness function	0.001163
Crossover	0.007252
Total generation time	0.043900

Table 5.10: GA operations standard deviation considering 100 individuals

Analyzing table 5.10 it is possible to deduce that all considered operators do not vary considerably between generations. Even the **individuals conversion** operator that takes an average of 66.58 seconds per generation, has a standard deviation of 0.039694 seconds.

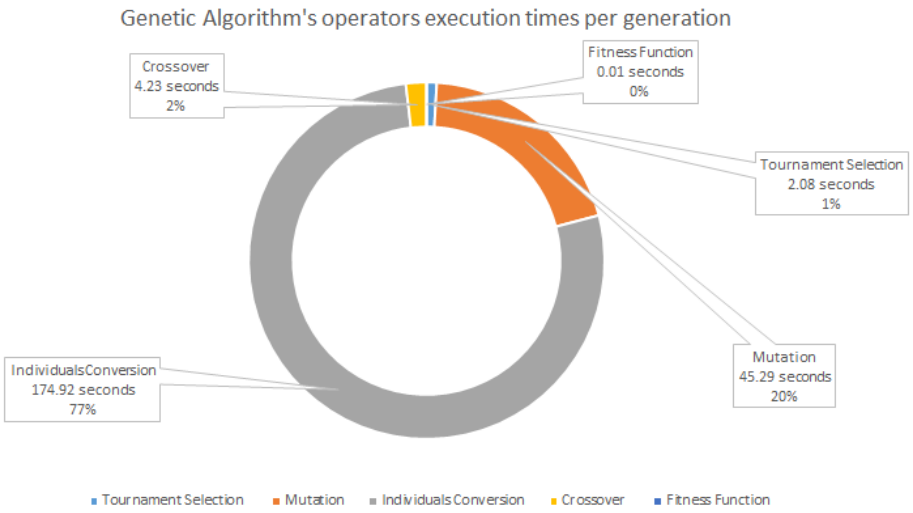

Figure 5.18: GA operators execution times considering 360 individuals

When considering a **population size** of **360** individuals, the total **setup time** took around 10 minutes, the **GA execution time** took approximately 206 minutes which gives a **total execution time** of 216 minutes.

When the population size is raised to 360 individuals, the overall GA operators execution time tend to increase in a non proportional way. However the less parallelized operators continue to use a much larger generation time portion.

Operation	Standard deviation (s)
Tournament selection	0.074765
Mutation	0.027173
Individuals conversion	0.134550
Fitness function	0.002228
Crossover	0.008753
Total generation time	0.163253

Table 5.11: GA operations standard deviation considering 360 individuals

Analyzing the table 5.11, like the experiment that considered one hundred individuals, all considered operators do not vary considerably between different generations.

The operation that takes a larger amount of each generation is surely the **individuals conversion** operation. Since this is not a default genetic algorithm's operation, it needs to be executed in order to evaluate the class individual view.

Chapter 6

Conclusions and future work

This thesis goal was very well defined: to investigate, design and implement a suitable GPGPU solution to tackle the FEUP's faculty timetabling generation problem. The viability of the implemented solution should be tested in order to recognize its potential for being explored in the future. It was designed and implemented a GA using CUDA parallel framework. The algorithm uses the input data provided by FEUP and it was able to produce timetables that respect a series of defined constraints.

There were performed several independent experiments in order to evaluate different genetic algorithm fitness function parts. Lastly a crossover operator execution time comparison was performed.

In first experiment 5.3.1, the classroom individual view was tested in isolation. This experiment proved that it is crucial to have a well defined notion of the importance level of each fitness function parameter in considered problem. Therefore the GA will evolve the individuals according to the defined fitness function. In case of the fitness function parameters were not well defined, the final achieved solution, even with a high fitness and a good evolution pace, will generate a *phenotype* that represents the incorrectly defined fitness function. However with balanced fitness function parameters values, it was possible to achieve a better classroom timetabling with an improved lesson allocation.

The second experiment 5.3.2, isolates the class view individual portion, considering only one fitness function parameter, the number of overlapping lessons. In the best solution, after 500 generations, the amount of overlapped lessons was reduced nearly 47% compared with generation 0. In this experiment, only two GA parameters values combinations were tested with each population evolving 500 generations. With a most suitable set of fitness function parameters values combination and higher number of generations could lead to better result. However the results were satisfying given the amount of evolved generations.

On third experiment 5.3.3, the two fitness function parts were joined and both individual views were considered. The two GA tested parameters combinations achieved different evolution paces. The best GA parameters combination achieved after 2000 generations the solution with higher fitness score.

The final classroom view was identical to its generation 0 homologous, however with small quality increase. On class view timetable, from a timetable with two overlapped lessons on generation 0, after 2000 generations one overlapped lessons was eliminated. In this experiment the total achieved performance was lower when comparing with two previous experiments 5.3.1, 5.3.2 that tested isolated fitnesses portions.

Due to hardware restrictions, it was only possible to test with populations with a maximum of 360 individuals. The last isolation tests had a satisfactory performance because the population genetic variability was enough to provide a faster evolution considering the problems complexity. However when considering all restrictions the problem complexity increases significantly, and it is required a higher population genetic variability to conduct the population towards a global maximum at a higher rate. The lack of variability does not reduce the computation time needed for each generation. However with higher genetic variability a higher evolution rate can be reached requiring a smaller amount of generations to produce the same solution quality.

On the fourth experiment a comparison among CPU single core and GPU order one crossover operator was performed. For the GPU implemented crossover operator to worth, a high amount of data should be sent for being processed on GPU avoiding copying data from *Host* memory to *Device* global memory and vice-versa.

The proposed objectives were fulfilled, with positive results produced. However better results could be achieved if more tests were conducted with different fitness function and GA parameters values combinations along with a larger number of individuals.

6.1 Future work

The development of a software for generating timetables with so many constraints and entities involved, is considered a complex task that requires a big team with more available time and resources to produce a more mature software solution. Despite that the developed implementation proves some important concepts. There are some aspects that could be improved allowing to explore this subject deeper and find out with more confidence if this approach is suitable for being used to produce real timetables for real institutions like FEUP.

A critical barrier found in this dissertation was the difficulty of eliminating entirely overlapped lessons present in class view timetable. It was shown on results presented in chapter 5 that it is possible to mitigate the amount of existent overlapped lessons on class view timetable. However this solution does not satisfy the requirements of a real timetable generation system like FEUP's case. A solution that solves entirely this problem, by eliminating any overlapped lesson in class view timetable while not reducing the performance, is crucial to prove the concept of this approach.

Apart from previous problem there are some improvements that will enable the algorithm to find superior solutions while using a smaller amount of computation time.

A possible improvement is the implementation of elitism, where a small group of the fittest individuals are preserved directly for the next generation. This solution avoids spending precious computation time in finding solutions that were already found, but not selected as parents for

breeding, causing their quality genes to be lost. It is a simple improvement that could potentially lead to better results with less amount of generations.

When considering a large amount of fitness function parameters, or in other words a large amount of constraints, it is ideal to use a larger population size. It could be a major way of increasing the initial genetic variability and diversity of population. However this genetic diversity needs to be maintained over generations in order to avoid the solution to overlook a local optimum and converge to it. The population variability can be maintained by testing more selection operators with different parameters so that a exaggerated selection pressure over population is avoided, since it is the main cause of premature convergence.

With a larger population and variability maintained over generations, the genetic algorithm is more suitable to embrace a more complex problem. The problem of professors allocation can be tackled, for FEUP's case there is a node *GrupoProfessores* on *FileAulas.xml* for each lesson. It indicates the unique code of each professor, and the professor availability in number of turns. If the previous problem of avoiding all lessons on class view timetable of being overlapped was solved, the same solution, with an extra level of complexity, could be applied to solve the possible overlapped lessons on professors view timetable. The professors distribution over the possible lessons turns, is a fairly simple problem compared with the overlapping lessons problem and it could play an important role to solve the overlapping problem, since professors could inter-change their turns in order to reduce the amount of overlapped lessons on their timetable view.

Another big improvement that could reduce the complexity of the problem and lead to a faster and superior solution, is to have lessons allocation divided between morning and afternoon. Where some pre-determined semesters of a cycle of studies are always taught at morning and others at afternoon. This system is already implemented by FEUP's timetable generation system, and helps to reduce the problem complexity and the lessons collision probabilities.

On a genetic algorithm implementation it is possible to explore adaptive mutation and crossover probabilities in order to maintain the population diversity while sustaining the GA capacity of converging to a global optimum.

In this implementation only was possible due to time restrictions to explore **order one crossover** method. However different crossover methods could be implemented and tested. Techniques like **order multiple crossover**, **cycle crossover**, **edge recombination**, **PMX crossover** should be implemented and tested against execution times and solution's quality.

This thesis presented a broad topic, although some of this thesis decisions made it narrower. However there are still an enormous amount of features that could be added and improvements to already implemented methods that could be made. Research work can lead to improvement which offers always the possibility to create a more mature software that can lead to higher quality solutions and lower the amount of time needed when compared with traditional timetable generation methods.

References

[Abe91] J. Abela. A parallel genetic algorithm for solving the school timetabling problem. Technical report, Division of Information Technology, C.S.I.R.O, 1991.

[ALN13] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013.

[BBB+12] Ruibin Bai, Jacek Blazewicz, Edmund K. Burke, Graham Kendall, and Barry McCollum. A simulated annealing hyper-heuristic methodology for flexible decision support. *4or*, 10(1):43–66, 2012.

[Dye] Daniel W. Dyer. A practical guide to the watchmaker framework. http://watchmaker.uncommons.org/manual/. Accessed: 2016-06-04.

[Edi] Biography.com Editors. The darwin biography. http://www.biography.com/people/charles-darwin-9266433#synopsis. Accessed: 2016-06-03.

[FBP13] Pedro Fernandes, Armando Barbosa, and Carla Sofia Pereira. BTTE - An automated timetabling software for Higher Education. 2013.

[GD91] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[GPG] GPGPU.org. Gpgpu.org :: General-purpose computation on graphics processing units. http://gpgpu.org. Accessed: 2016-03-08.

[Gro] The Khronos Group. The khronos group - a non-profit industry consortium to develop, publish and promote open standard, royalty-free media authoring and acceleration standards for desktop and handheld devices, combined with conformance qualification programs for platform and device interoperability. https://www.khronos.org/opencl/. Accessed: 2016-03-04.

[Hol75] John H Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, Oxford, England, 1975.

[Jeb] Khalid Jebari.

[KGG+14] Vasileios Kolonias, George Goulas, Christos Gogos, Panayiotis Alefragis, and Efthymios Housos. Solving the examination timetabling problem in GPUs. *Algorithms*, 7(3):295–327, 2014.

[LB08] Milena Lazarova and Plamenka Borovska. Comparison of parallel metaheuristics for solving the TSP. ... *of the 9th International Conference on* ..., page II.12, 2008.

[LZL09] Yongkai Liu, Defu Zhang, and Stephen C.H. Leung. A simulated annealing algorithm with a new neighborhood structure for the timetabling problem. *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation - GEC '09*, page 381, 2009.

[MBL+09] Ogier Maitre, Laurent A. Baumes, Nicolas Lachiche, Avelino Corma, and Pierre Collet. Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1403–1410, New York, NY, USA, 2009. ACM.

[Mic] Microsoft. The hlsl language. https://msdn.microsoft.com/en-us/library/windows/desktop/bb509561(v=vs.85).aspx. Accessed: 2016-06-03.

[Nvia] Nvidia. The cg toolkit. https://developer.nvidia.com/cg-toolkit. Accessed: 2016-06-03.

[Nvib] Nvidia. The nvcc compiler. http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/#axzz4AWkBbQ4c. Accessed: 2016-06-03.

[Nvic] Nvidia. Parallel programming and computing platform | cuda | nvidialnvidia. http://www.nvidia.com/object/cuda_home_new.html. Accessed: 2016-03-06.

[Obi] Marek Obitko. Genetic algorithms tutorials. http://obitko.com/tutorials/genetic-algorithms. Accessed: 2016-06-03.

[oCS] Carnegie Mellon University School of Computer Science. What is a genetic algorithm. https://www.cs.cmu.edu/Groups/AI/html/faqs/ai/genetic/part2/faq-doc-2.html. Accessed: 2016-06-03.

[Opea] OpenACC.org. Frequently asked questions. http://www.openacc.org/faq-page. Accessed: 2016-03-08.

[Opeb] OpenGL. The glsl language. https://www.opengl.org/documentation/glsl/. Accessed: 2016-06-03.

[Pil13] Nelishia Pillay. A comparative study of hyper-heuristics for solving the school timetabling problem. *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference on - SAICSIT '13*, page 278, 2013.

[PJS10] Petr Pospichal, Jiri Jaros, and Josef Schwarz. Parallel genetic algorithm on the cuda architecture. In *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation - Volume Part I*, EvoApplicatons'10, pages 442–451, Berlin, Heidelberg, 2010. Springer-Verlag.

[PU94] Alberto Colorni Politecnico and Marco Dorigo Universit. . (July), 1994.

[puga] pugixml.org. pugixml 1.7 quick start guide. http://pugixml.org/docs/quickstart.html. Accessed: 2016-02-15.

[pugb] pugixml.org. pugixml github repository. https://github.com/zeux/pugixml. Accessed: 2016-02-15.

[RA01] Marcus Randall and David Abramson. A general meta-heuristic based solver for combinatorial optimisation problems. *Computational Optimization and Applications*, 20(2):185–210, 2001.

[Rag13] Rushil Raghavjee. A Study of Genetic Algorithms for Solving the School Timetabling Problem. pages 193–199, 2013.

[SAD03] Kate A. Smith, David Abramson, and David Duke. Hopfield neural networks for timetabling: formulations, methods, and comparative results. *Computers & Industrial Engineering*, 44(2):283–305, 2003.

REFERENCES

Appendix A

XML data input files

The following appendix show some samples of the XML files that were used as the genetic algorithm input data.

A.1 *AreasCientificas.xml*

```
<ns1:AreasCientificas xmlns:ns1="urn:spaceAreasCientificas">
  <AreaCientifica>
    <Nome>Departamento de Engenharia Fisica</Nome>
  </AreaCientifica>
  <AreaCientifica>
    <Nome>Faculdade de Engenharia da Universidade do Porto</Nome>
  </AreaCientifica>
  <AreaCientifica>
    <Nome>Departamento de Engenharia e Gestao Industrial</Nome>
  </AreaCientifica>
  <AreaCientifica>
    <Nome>Departamento de Engenharia Eletrotecnica e de Computadores</Nome>
  </AreaCientifica>
  <AreaCientifica>
    <Nome>Seccao de Matematica</Nome>
  </AreaCientifica>
  <AreaCientifica>
    <Nome>Seccao de Materiais e Processos Tecnologicos</Nome>
  </AreaCientifica>
  <AreaCientifica>
    <Nome>Seccao de Vias de Comunicacao</Nome>
  </AreaCientifica>
  <AreaCientifica>
    <Nome>Seccao de Estruturas</Nome>
  </AreaCientifica>
</ns1:AreasCientificas>
```

A.2 *Caracteristicas.xml*

```
<ns1:Caracteristicas xmlns:ns1="urn:spaceCaracteristicas">
  <Caracteristica>
    <Nome>Anfiteatros - 184 lugares</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Anfiteatros - 53 lugares</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Anfiteatros - 60 lugares</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Anfiteatros - 99 lugares</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Ensino</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Salas Desenho</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Salas PC's 20 PC's 30 lugares</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Salas PC's - 12 Pc's e 24 lugares</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Salas PC's - 14 Pc's e 24 lugares</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Salas PC's - 8 Pc's e 24 lugares</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Salas Pc's - Arcview</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Salas TP - 28 lugares</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Salas TP - 32 lugares</Nome>
  </Caracteristica>
  <Caracteristica>
    <Nome>Salas TP - 40 lugares</Nome>
  </Caracteristica>
</ns1:Caracteristicas>
```

A.3 *Categorias.xml*

```
<ns1:Categorias xmlns:ns1="urn:spaceCategorias">
  <Categoria>
    <Nome>Assistente</Nome>
  </Categoria>
  <Categoria>
    <Nome>Assistente Convidado</Nome>
  </Categoria>
  <Categoria>
    <Nome>Direccao intermedia de 1 grau</Nome>
  </Categoria>
  <Categoria>
    <Nome>Especialista de Informatica Grau 3 Nivel 1</Nome>
  </Categoria>
  <Categoria>
    <Nome>Investigador Auxiliar</Nome>
  </Categoria>
  <Categoria>
    <Nome>Investigador Principal</Nome>
  </Categoria>
  <Categoria>
    <Nome>Presidente do Conselho Pedagogico</Nome>
  </Categoria>
  <Categoria>
    <Nome>Professor Associado</Nome>
  </Categoria>
  <Categoria>
    <Nome>Professor Associado Convidado</Nome>
  </Categoria>
  <Categoria>
    <Nome>Professor Auxiliar</Nome>
  </Categoria>
  <Categoria>
    <Nome>Professor Auxiliar Convidado</Nome>
  </Categoria>
    <Categoria>
    <Nome>Professor Catedratico</Nome>
    </Categoria>
    <Categoria>
      <Nome>Professor Catedratico Convidado</Nome>
    </Categoria>
    <Categoria>
      <Nome>Pro-Reitor</Nome>
    </Categoria>
</ns1:Categorias>
```

A.4 *Cursos.xml*

```
<ns1:Cursos xmlns:ns1="urn:spaceCursos">
  <Curso>
    <Nome>MIEA - Mestrado Integrado em Engenharia do Ambiente</Nome>
    <Sigla>MIEA</Sigla>
    <Codigo>726</Codigo>
  </Curso>
  <Curso>
    <Nome>MESG - Mestrado em Engenharia de Servicos e Gestao</Nome>
    <Sigla>MESG</Sigla>
    <Codigo>727</Codigo>
  </Curso>
  <Curso>
    <Nome>MIB - Mestrado Integrado em Bioengenharia</Nome>
    <Sigla>MIB</Sigla>
    <Codigo>728</Codigo>
  </Curso>
  <Curso>
    <Nome>LCEEMG - Licenciatura em Ciencias de Engenharia - Engenharia de Minas e
        Geo</Nome>
    <Sigla>LCEEMG</Sigla>
    <Codigo>738</Codigo>
  </Curso>
  <Curso>
    <Nome>MEMG - Mestrado em Engenharia de Minas e Geo-Ambiente</Nome>
    <Sigla>MEMG</Sigla>
    <Codigo>739</Codigo>
  </Curso>
  <Curso>
    <Nome>MIEC - Mestrado Integrado em Engenharia Civil</Nome>
    <Sigla>MIEC</Sigla>
    <Codigo>740</Codigo>
  </Curso>
  <Curso>
    <Nome>MIEEC - Mestrado Integrado em Engenharia Electrotecnica e de Computadores
        </Nome>
    <Sigla>MIEEC</Sigla>
    <Codigo>741</Codigo>
  </Curso>
  <Curso>
    <Nome>MIEIC - Mestrado Integrado em Engenharia Informatica e Computacao</Nome>
    <Sigla>MIEIC</Sigla>
    <Codigo>742</Codigo>
  </Curso>
</ns1:Cursos>
```

A.5 *Disciplinas.xml*

```
<ns1:Disciplinas xmlns:ns1="urn:spaceDisciplinas">
<Disciplina>
  <Nome>EA0007 - Quimica Ambiental I</Nome>
  <Sigla>QA I(2015 - 2)</Sigla>
  <Codigo>366242</Codigo>
  <AreaCientifica>Departamento de Engenharia Quimica</AreaCientifica>
</Disciplina>
<Disciplina>
  <Nome>EIG0045 - Analise Matematica III</Nome>
  <Sigla>AM III(2015 - 2)</Sigla>
  <Codigo>366091</Codigo>
  <AreaCientifica>Seccao de Matematica</AreaCientifica>
</Disciplina>
<Disciplina>
  <Nome>EIG0040 - Estrategia e Competitividade Empresarial</Nome>
  <Sigla>ECE(2015 - 2)</Sigla>
  <Codigo>366120</Codigo>
  <AreaCientifica>Departamento de Engenharia e Gestao Industrial</AreaCientifica>
</Disciplina>
<Disciplina>
  <Nome>EIG0031 - Marketing</Nome>
  <Sigla>MK(2015 - 2)</Sigla>
  <Codigo>366111</Codigo>
  <AreaCientifica>Departamento de Engenharia e Gestao Industrial</AreaCientifica>
</Disciplina>
<Disciplina>
  <Nome>EIG0030 - Logistica</Nome>
  <Sigla>L(2015 - 2)</Sigla>
  <Codigo>366110</Codigo>
  <AreaCientifica>Departamento de Engenharia e Gestao Industrial</AreaCientifica>
</Disciplina>
<Disciplina>
  <Nome>EIG0023 - Microeconomia</Nome>
  <Sigla>MI(2015 - 2)</Sigla>
  <Codigo>366102</Codigo>
  <AreaCientifica>Departamento de Engenharia e Gestao Industrial</AreaCientifica>
</Disciplina>
<Disciplina>
  <Nome>EIG0015 - Estatistica I</Nome>
  <Sigla>E I(2015 - 2)</Sigla>
  <Codigo>366090</Codigo>
  <AreaCientifica>Departamento de Engenharia e Gestao Industrial</AreaCientifica>
</Disciplina>
</ns1:Disciplinas>
```

A.6 *Edificios.xml*

```
<ns1:Edificios xmlns:ns1="urn:spaceEdificios">
  <Edificio>
    <Nome>B - Aulas iv</Nome>
  </Edificio>
  <Edificio>
    <Nome>E - Quimica</Nome>
  </Edificio>
  <Edificio>
    <Nome>J - Electrotecnia Norte</Nome>
  </Edificio>
  <Edificio>
    <Nome>H - Civil Norte</Nome>
  </Edificio>
  <Edificio>
    <Nome>B - Aulas ii</Nome>
  </Edificio>
  <Edificio>
    <Nome>I - Electrotecnia</Nome>
  </Edificio>
  <Edificio>
    <Nome>F - Minas e Metalurgia</Nome>
  </Edificio>
  <Edificio>
    <Nome>L - Mecanica</Nome>
  </Edificio>
  <Edificio>
    <Nome>M - Mecanica Norte</Nome>
  </Edificio>
  <Edificio>
    <Nome>G - Civil</Nome>
  </Edificio>
</ns1:Edificios>
```

A.7 *FileAulas.xml*

```
<ns1:FileAulas xmlns:ns1="urn:spaceFileAulas">
  <AulasDisciplina>
    <CodigoDisciplina>364801</CodigoDisciplina>
    <Aula>
      <Nome>TP1</Nome>
      <Tipologias>
        <Nome>TP</Nome>
```

```xml
      <NumSlots>4</NumSlots>
    </Tipologias>
    <NumTurnos>3</NumTurnos>
    <Repeticao>2</Repeticao>
    <Turno>
      <NomeTurma>MIEA101</NomeTurma>
      <NomeTurma>MIEA102</NomeTurma>
      <NomeTurma>MIEA103</NomeTurma>
      <NomeTurma>1EMM01</NomeTurma>
      <NomeTurma>1EMM02</NomeTurma>
      <NomeTurma>1EMM03</NomeTurma>
      <NomeTurma>1LCEEMG01</NomeTurma>
      <NomeTurma>1LCEEMG02</NomeTurma>
      <NomeTurma>1LCEEMG03</NomeTurma>
      <NumAlunos>48</NumAlunos>
    </Turno>
    <Turno>
      <NomeTurma>MIEA101</NomeTurma>
      <NomeTurma>MIEA102</NomeTurma>
      <NomeTurma>MIEA103</NomeTurma>
      <NomeTurma>1EMM01</NomeTurma>
      <NomeTurma>1EMM02</NomeTurma>
      <NomeTurma>1EMM03</NomeTurma>
      <NomeTurma>1LCEEMG01</NomeTurma>
      <NomeTurma>1LCEEMG02</NomeTurma>
      <NomeTurma>1LCEEMG03</NomeTurma>
      <NumAlunos>48</NumAlunos>
    </Turno>
    <Turno>
      <NomeTurma>MIEA101</NomeTurma>
      <NomeTurma>MIEA102</NomeTurma>
      <NomeTurma>MIEA103</NomeTurma>
      <NomeTurma>1EMM01</NomeTurma>
      <NomeTurma>1EMM02</NomeTurma>
      <NomeTurma>1EMM03</NomeTurma>
      <NomeTurma>1LCEEMG01</NomeTurma>
      <NomeTurma>1LCEEMG02</NomeTurma>
      <NomeTurma>1LCEEMG03</NomeTurma>
      <NumAlunos>48</NumAlunos>
    </Turno>
    <GrupoSalas>
      <Sala>
        <Nome>B001</Nome>
        <Alternativa>1</Alternativa>
      </Sala>
      <Sala>
        <Nome>B002</Nome>
        <Alternativa>1</Alternativa>
      </Sala>
```

```
57          <Sala>
58            <Nome>B003</Nome>
59            <Alternativa>1</Alternativa>
60          </Sala>
61          <Sala>
62            <Nome>B004</Nome>
63            <Alternativa>1</Alternativa>
64          </Sala>
65          <Sala>
66            <Nome>B005</Nome>
67            <Alternativa>1</Alternativa>
68          </Sala>
69          <Sala>
70            <Nome>B006</Nome>
71            <Alternativa>1</Alternativa>
72          </Sala>
73          <Sala>
74            <Nome>B007</Nome>
75            <Alternativa>1</Alternativa>
76          </Sala>
77          <Sala>
78            <Nome>B008</Nome>
79            <Alternativa>0</Alternativa>
80          </Sala>
81          <Sala>
82            <Nome>B009</Nome>
83            <Alternativa>0</Alternativa>
84          </Sala>
85          <Sala>
86            <Nome>B010</Nome>
87            <Alternativa>1</Alternativa>
88          </Sala>
89          <Sala>
90            <Nome>B011</Nome>
91            <Alternativa>1</Alternativa>
92          </Sala>
93          <Sala>
94            <Nome>B012</Nome>
95            <Alternativa>1</Alternativa>
96          </Sala>
97          <Sala>
98            <Nome>B013</Nome>
99            <Alternativa>1</Alternativa>
100         </Sala>
101         <Sala>
102           <Nome>B014</Nome>
103           <Alternativa>1</Alternativa>
104         </Sala>
105         <Sala>
```

```
106           <Nome>B015</Nome>
107           <Alternativa>1</Alternativa>
108         </Sala>
109         <Sala>
110           <Nome>B016</Nome>
111           <Alternativa>1</Alternativa>
112         </Sala>
113         <Sala>
114           <Nome>B017</Nome>
115           <Alternativa>1</Alternativa>
116         </Sala>
117         <Sala>
118           <Nome>B018</Nome>
119           <Alternativa>0</Alternativa>
120         </Sala>
121         <Sala>
122           <Nome>B019</Nome>
123           <Alternativa>0</Alternativa>
124         </Sala>
125         <Sala>
126           <Nome>B020</Nome>
127           <Alternativa>1</Alternativa>
128         </Sala>
129         <Sala>
130           <Nome>B021</Nome>
131           <Alternativa>1</Alternativa>
132         </Sala>
133         <Sala>
134           <Nome>B022</Nome>
135           <Alternativa>1</Alternativa>
136         </Sala>
137         <Sala>
138           <Nome>B023</Nome>
139           <Alternativa>1</Alternativa>
140         </Sala>
141         <Sala>
142           <Nome>B024</Nome>
143           <Alternativa>0</Alternativa>
144         </Sala>
145         <Sala>
146           <Nome>B025</Nome>
147           <Alternativa>0</Alternativa>
148         </Sala>
149         <Sala>
150           <Nome>B026</Nome>
151           <Alternativa>1</Alternativa>
152         </Sala>
153         <Sala>
154           <Nome>B027</Nome>
```

```
          <Alternativa>1</Alternativa>
        </Sala>
        <Sala>
          <Nome>B028</Nome>
          <Alternativa>1</Alternativa>
        </Sala>
        <Sala>
          <Nome>B029</Nome>
          <Alternativa>1</Alternativa>
        </Sala>
        <Sala>
          <Nome>B030</Nome>
          <Alternativa>0</Alternativa>
        </Sala>
        <Sala>
          <Nome>B031</Nome>
          <Alternativa>0</Alternativa>
        </Sala>
        <Sala>
          <Nome>B033</Nome>
          <Alternativa>1</Alternativa>
        </Sala>
        <Sala>
          <Nome>B034</Nome>
          <Alternativa>1</Alternativa>
        </Sala>
        <Sala>
          <Nome>B035</Nome>
          <Alternativa>1</Alternativa>
        </Sala>
      </GrupoSalas>
      <GrupoProfessores>
        <CodigoProfessor>211908</CodigoProfessor>
        <NumTurnos>2</NumTurnos>
      </GrupoProfessores>
      <GrupoProfessores>
        <CodigoProfessor>211157</CodigoProfessor>
        <NumTurnos>1</NumTurnos>
      </GrupoProfessores>
    </Aula>
  </AulasDisciplina>
</ns1:FileAulas>
```

A.8 *PlanosCurriculares.xml*

```
<ns1:PlanosCurriculares xmlns:ns1="urn:spacePlanosCurriculares">
  <PlanoCurricular>
    <Nome>Plano Oficial a partir de 2008/2009 - 1</Nome>
    <Codigo>1575-1</Codigo>
    <Ano>1</Ano>
    <CodigoCurso>454</CodigoCurso>
    <DisciplinaDoPlano>
      <CodigoDisciplina>365524</CodigoDisciplina>
    </DisciplinaDoPlano>
    <DisciplinaDoPlano>
      <CodigoDisciplina>365484</CodigoDisciplina>
    </DisciplinaDoPlano>
    <DisciplinaDoPlano>
      <CodigoDisciplina>365503</CodigoDisciplina>
    </DisciplinaDoPlano>
    <DisciplinaDoPlano>
      <CodigoDisciplina>365541</CodigoDisciplina>
    </DisciplinaDoPlano>
    <DisciplinaDoPlano>
      <CodigoDisciplina>365502</CodigoDisciplina>
    </DisciplinaDoPlano>
  </PlanoCurricular>
  <PlanoCurricular>
    <Nome>Plano Oficial a partir de 2008/2009 - 2</Nome>
    <Codigo>1575-2</Codigo>
    <Ano>2</Ano>
    <CodigoCurso>454</CodigoCurso>
    <DisciplinaDoPlano>
      <CodigoDisciplina>365520</CodigoDisciplina>
    </DisciplinaDoPlano>
    <DisciplinaDoPlano>
      <CodigoDisciplina>365519</CodigoDisciplina>
    </DisciplinaDoPlano>
    <DisciplinaDoPlano>
      <CodigoDisciplina>365518</CodigoDisciplina>
    </DisciplinaDoPlano>
    <DisciplinaDoPlano>
      <CodigoDisciplina>365506</CodigoDisciplina>
    </DisciplinaDoPlano>
    <DisciplinaDoPlano>
      <CodigoDisciplina>365523</CodigoDisciplina>
    </DisciplinaDoPlano>
  </PlanoCurricular>
  <PlanoCurricular>
    <Nome>Plano Oficial a partir de 2008/2009 - 3</Nome>
    <Codigo>1575-3</Codigo>
```

```
47      <Ano>3</Ano>
48      <CodigoCurso>454</CodigoCurso>
49      <DisciplinaDoPlano>
50        <CodigoDisciplina>365517</CodigoDisciplina>
51      </DisciplinaDoPlano>
52      <DisciplinaDoPlano>
53        <CodigoDisciplina>365512</CodigoDisciplina>
54      </DisciplinaDoPlano>
55      <DisciplinaDoPlano>
56        <CodigoDisciplina>365483</CodigoDisciplina>
57      </DisciplinaDoPlano>
58      <DisciplinaDoPlano>
59        <CodigoDisciplina>365511</CodigoDisciplina>
60      </DisciplinaDoPlano>
61      <DisciplinaDoPlano>
62        <CodigoDisciplina>365467</CodigoDisciplina>
63      </DisciplinaDoPlano>
64      <DisciplinaDoPlano>
65        <CodigoDisciplina>365510</CodigoDisciplina>
66      </DisciplinaDoPlano>
67      <DisciplinaDoPlano>
68        <CodigoDisciplina>365509</CodigoDisciplina>
69      </DisciplinaDoPlano>
70    </PlanoCurricular>
71  </ns1:PlanosCurriculares>
```

A.9 *Professores.xml*

```
1   <ns1:Professores xmlns:ns1="urn:spaceProfessores">
2     <Professor>
3       <Nome>******** *********</Nome>
4       <Sigla>***</Sigla>
5       <Codigo></Codigo>
6       <Categoria>Professor Catedratico</Categoria>
7       <AreaCientifica>******** *********</AreaCientifica>
8       <DiasLivres>2</DiasLivres>
9       <LimiteMaximo>a7</LimiteMaximo>
10      <LimiteConsecutivo>5</LimiteConsecutivo>
11    </Professor>
12    <Professor>
13      <Nome>******** *********</Nome>
14      <Sigla>***</Sigla>
15      <Codigo>******</Codigo>
16      <Categoria>Professor Associado</Categoria>
17      <AreaCientifica>******** *********</AreaCientifica>
18      <DiasLivres>2</DiasLivres>
```

```
    <LimiteMaximo>7</LimiteMaximo>
    <LimiteConsecutivo>5</LimiteConsecutivo>
  </Professor>
  <Professor>
    <Nome>******** *********</Nome>
    <Sigla>***</Sigla>
    <Codigo>******</Codigo>
    <Categoria>Professor Auxiliar Convidado</Categoria>
    <AreaCientifica>
    ******** *********
    </AreaCientifica>
    <DiasLivres>2</DiasLivres>
    <LimiteMaximo>7</LimiteMaximo>
    <LimiteConsecutivo>5</LimiteConsecutivo>
  </Professor>
  <Professor>
    <Nome>******** *********</Nome>
    <Sigla>***</Sigla>
    <Codigo>******</Codigo>
    <Categoria>Assistente Convidado</Categoria>
    <AreaCientifica>******** *********</AreaCientifica>
    <DiasLivres>2</DiasLivres>
    <LimiteMaximo>7</LimiteMaximo>
    <LimiteConsecutivo>5</LimiteConsecutivo>
  </Professor>
  <Professor>
    <Nome>******** *********</Nome>
    <Sigla>***</Sigla>
    <Codigo>******</Codigo>
    <Categoria>Professor Catedratico</Categoria>
    <AreaCientifica>******** *********</AreaCientifica>
    <DiasLivres>2</DiasLivres>
    <LimiteMaximo>7</LimiteMaximo>
    <LimiteConsecutivo>5</LimiteConsecutivo>
  </Professor>
</ns1:Professores>
```

A.10 *RestricaoObjectivo.xml*

```
<ns1:RestricaoObjectivo xmlns:ns1="urn:spaceRestricaoObjectivo">
  <AulasContiguas>
    <Objectivo>
      <Aula1>
        <CodigoDisciplina>364804</CodigoDisciplina>
        <NomeAula>PL1</NomeAula>
        <NomeTurno>PL1</NomeTurno>
```

```
      </Aula1>
      <Aula2>
        <CodigoDisciplina>364804</CodigoDisciplina>
        <NomeAula>PL1</NomeAula>
        <NomeTurno>PL2</NomeTurno>
      </Aula2>
    </Objectivo>
  </AulasContiguas>
  <OrdemTipologias>
    <Objectivo>
      <Tipologia1>T</Tipologia1>
      <Tipologia2>PL</Tipologia2>
      <CodigoDisciplina>365723</CodigoDisciplina>
      <CodigoDisciplina>365736</CodigoDisciplina>
      <CodigoDisciplina>365652</CodigoDisciplina>
      <CodigoDisciplina>365750</CodigoDisciplina>
      <CodigoDisciplina>365682</CodigoDisciplina>
      <CodigoDisciplina>365702</CodigoDisciplina>
      <CodigoDisciplina>365737</CodigoDisciplina>
      <CodigoDisciplina>368699</CodigoDisciplina>
      <CodigoDisciplina>365679</CodigoDisciplina>
      <CodigoDisciplina>365645</CodigoDisciplina>
      <CodigoDisciplina>365647</CodigoDisciplina>
      <CodigoDisciplina>365699</CodigoDisciplina>
      <CodigoDisciplina>365683</CodigoDisciplina>
      <CodigoDisciplina>365745</CodigoDisciplina>
      <CodigoDisciplina>365703</CodigoDisciplina>
      <CodigoDisciplina>365729</CodigoDisciplina>
      <CodigoDisciplina>367743</CodigoDisciplina>
      <CodigoDisciplina>365642</CodigoDisciplina>
    </Objectivo>
  </OrdemTipologias>
</ns1:RestricaoObjectivo>
```

A.11 *Salas.xml*

```
<ns1:Salas xmlns:ns1="urn:spaceSalas">
  <Sala>
    <Nome>B001</Nome>
    <Codigo>73141</Codigo>
    <Edificio>B - Aulas i</Edificio>
    <Piso>0</Piso>
    <Capacidade>184</Capacidade>
    <Caracteristica>Anfiteatros - 184 lugares</Caracteristica>
  </Sala>
  <Sala>
```

```
    <Nome>B003</Nome>
    <Codigo>73143</Codigo>
    <Edificio>B - Aulas i</Edificio>
    <Piso>0</Piso>
    <Capacidade>184</Capacidade>
    <Caracteristica>Anfiteatros - 184 lugares</Caracteristica>
  </Sala>
  <Sala>
    <Nome>B002</Nome>
    <Codigo>73144</Codigo>
    <Edificio>B - Aulas i</Edificio>
    <Piso>0</Piso>
    <Capacidade>184</Capacidade>
    <Caracteristica>Anfiteatros - 184 lugares</Caracteristica>
  </Sala>
  <Sala>
    <Nome>B222</Nome>
    <Codigo>73146</Codigo>
    <Edificio>B - Aulas iii</Edificio>
    <Piso>2</Piso>
    <Capacidade>32</Capacidade>
    <Caracteristica>Salas TP - 32 lugares</Caracteristica>
  </Sala>
</ns1:Salas>
```

A.12 *Tipologias.xml*

```
<ns1:Tipologias xmlns:ns1="urn:spaceTipologias">
  <Tipologia>
    <Nome>E</Nome>
  </Tipologia>
  <Tipologia>
    <Nome>L</Nome>
  </Tipologia>
  <Tipologia>
    <Nome>O</Nome>
  </Tipologia>
  <Tipologia>
    <Nome>OT</Nome>
  </Tipologia>
  <Tipologia>
    <Nome>P</Nome>
  </Tipologia>
  <Tipologia>
    <Nome>PL</Nome>
  </Tipologia>
```

```
  <Tipologia>
    <Nome>S</Nome>
  </Tipologia>
  <Tipologia>
    <Nome>T</Nome>
  </Tipologia>
  <Tipologia>
    <Nome>TC</Nome>
  </Tipologia>
  <Tipologia>
    <Nome>TP</Nome>
  </Tipologia>
</ns1:Tipologias>
```

A.13 *Turmas.xml*

```
<ns1:Turmas xmlns:ns1="urn:spaceTurmas">
  <Turma>
    <Nome>1MEB01</Nome>
    <Codigo>206105</Codigo>
    <CodigoPlanoCurricular>2636-1</CodigoPlanoCurricular>
  </Turma>
  <Turma>
    <Nome>2MEB01</Nome>
    <Codigo>206106</Codigo>
    <CodigoPlanoCurricular>2636-2</CodigoPlanoCurricular>
  </Turma>
  <Turma>
    <Nome>PRODEI01</Nome>
    <Codigo>205888</Codigo>
    <CodigoPlanoCurricular>2511-1</CodigoPlanoCurricular>
  </Turma>
  <Turma>
    <Nome>PRODEI04</Nome>
    <Codigo>205891</Codigo>
    <CodigoPlanoCurricular>2511-1</CodigoPlanoCurricular>
  </Turma>
  <Turma>
    <Nome>PRODEI02</Nome>
    <Codigo>205889</Codigo>
    <CodigoPlanoCurricular>2511-1</CodigoPlanoCurricular>
  </Turma>
  <Turma>
    <Nome>PRODEI03</Nome>
    <Codigo>205890</Codigo>
    <CodigoPlanoCurricular>2511-1</CodigoPlanoCurricular>
```

```
  </Turma>
  <Turma>
    <Nome>PRODEB1</Nome>
    <Codigo>206117</Codigo>
    <CodigoPlanoCurricular>2498-1</CodigoPlanoCurricular>
  </Turma>
  <Turma>
    <Nome>PRODEB2</Nome>
    <Codigo>206118</Codigo>
    <CodigoPlanoCurricular>2498-2</CodigoPlanoCurricular>
  </Turma>
  <Turma>
    <Nome>PRODEB3</Nome>
    <Codigo>206119</Codigo>
    <CodigoPlanoCurricular>2498-3</CodigoPlanoCurricular>
  </Turma>
  <Turma>
    <Nome>1MIETE</Nome>
    <Codigo>206109</Codigo>
    <CodigoPlanoCurricular>2488-1</CodigoPlanoCurricular>
  </Turma>
</ns1:Turmas>
```

Appendix B

XML data input files description

The following appendix show a more detailed description of files presented on appendix A.

B.1 *AreasCientificas.xml*

It lists all the scientific areas that coexist in FEUP ecosystem. Each professor must have only one correspondent scientific area. *AreaCientifica* node has a child node called *Nome* which has a child text node that represents one of many FEUP's scientific areas.

B.2 *Caracteristicas.xml*

This file lists all classroom features present in entire FEUP campus. Inside each *Caracteristica* node, there is a child node designated *Nome* which has a child text node that represents one of many classroom possible features.

For example it can mention that a classroom feature is having a given number of available computers, it is a drawing classroom or even that it is a auditorium classroom.

B.3 *Categorias.xml*

This XML file, maintains information about different groups/categories of professors that may be present in FEUP's ecosystem.

Inside each *Categoria* node, there is a child node named *Nome* which like the previous files has a text node child that represents one of many categories that a teacher may belong to.

B.4 *Cursos.xml*

Cursos.xml holds information about each course/cycle of studies available for FEUP's students.

Each *Curso* node, has three children nodes:

- *Nome*: Has a text node child that holds information about the course's name;

- *Sigla*: Has a text node child which has acronym from which the course is known;
- *Codigo*: Lastly, *Codigo* node holds like the previous ones a text node child with a unique code that identifies the course.

B.5 *Disciplinas.xml*

Disciplinas.xml holds information about all the existent course units in FEUP.

Each *Disciplina* node, has five children nodes:

- *Nome*: Has a child text node that holds information about the course unit name;
- *Sigla*: Has a child text node which has acronym from which the course unit is known;
- *Codigo*: *Codigo* node holds a child text node with a unique code that identifies the referred course unit;
- *Importancia*: It is a optional field and represents the importance degree of a given course unit;
- *AreaCientifica*: Lastly, *AreaCientifica* node holds a child text node with the complete name of the scientific area where the referred course unit belongs. This scientific area complete name must be present on *AreasCientificas.xml* file.

B.6 *Edificios.xml*

This file lists all the buildings that compose FEUP faculty. Each *Edificio* node, has a child text node with the identification name of one of the several existent FEUP's buildings. An optional node is *Zona*, that represents the campus where the building is inserted.

B.7 *FileAulas.xml*

This file lists all the lessons that must be taught referent to all faculty courses. In this file lessons are organized by course units represented by the *AulasDisciplina* node. This node has two possible children, *CodigoDisciplina* has the course unit that each following defined lessons belong to, and one or more *Aula* node that represents a given lesson. It is worthy to explain the *Aula* node in more detail, it may have eight children nodes:

- *Nome*: Holds information about the lesson name;
- *Tipologias*: Has two child nodes, *Nome* holds the typology name and *NumSlots* holds the number of slots that are required for the lesson on timetable;
- *NumTurnos*: Has information about the lesson's number of turns;

- *Repeticao*: Represents the number of repetitions per week that a lesson may have;
- *Turno*: May have several *NomeTurma* nodes that represent the classes that are present in a given turn as well as a node *NumAlunos* that informs about the turn's number of students;
- *GrupoProfessores*: Is an optional node that informs, using professors unique identification code, the set of professors that are allowed to teach the given lesson;
- *GrupoAulas*: Is an optional node that informs, using classrooms names, the set of classrooms that are suitable to teach the referred lesson. This classrooms are marked as preferential or alternative.

B.8 *PlanosCurriculares.xml*

This file holds information about each course's syllabus in FEUP's system. There is one entry for each year of each syllabus.

Each *PlanoCurricular* node has five direct children nodes:

- *Nome*: Has a child text node that holds the syllabus name;
- *Codigo*: Has a child text node with a unique code that identifies the syllabus itself. This code is composed by two parts separated by dash sign: the first part is the actual identification code, the second part represents the syllabus year;
- *Ano*: Holds the same information as the second part of the *Codigo* node, the syllabus year;
- *CodigoCurso*: *CodigoCurso* node holds like the previous ones a child text node with the correspondent course's code that belongs to it;
- *DisciplinaDoPlano*: This last node can appear more than one time. It references the discipline codes by using a child node with the course unit's unique identification code that belong to the syllabus in the referred year. There may be an optional field named *Opcional*, that represents the possibility of a given course unit being facultative in the given curricular plan.

B.9 *Professores.xml*

Professors.xml contains information about each professor that reside in FEUP's ecosystem.

Each *Professor* node has ten children nodes:

- *Nome*: Has a child text node that holds the professor's name;
- *Sigla*: Has a child text node that contains the professor acronym, which may not be unique;
- *Codigo*: Has a unique code that identifies the professor;

- *Categoria*: Holds information about the professor's category. It should be one present on *Categorias.xml* file;
- *AreaCientifica*: Holds information about the professor's scientific area, the information present in this node should be one present as well on *AreasCientificas.xml* file;
- *Vinculo*: It is an optional field and expresses the professor's contract type;
- *DiasLivres*: This node represents the number of days off per week (days without teaching classes) that each teacher may have;
- *Antiguidade*: This node represents the level of service time that the professor has with the institution;
- *LimiteMaximo*: Represents the maximum number of lesson hours per day;
- *LimiteConsecutivo*: The last node holds information about the maximum number of hours of successive lessons per day.

B.10 *RestricaoObjectivo.xml*

RestricaoObjectivo.xml file contains information about existent restrictions and objectives that may exist and be considered for timetabling construction process.

There are six possible parameters to which restrictions and objectives may be defined: **contiguous lessons**, **separated lessons**, **overlapped lessons**, **lessons not taught in the same day**, **lessons not taught in the same day with same typology**, **free days** and **typology order**.

Each of this six parameters can have any number of children nodes named *objectivo* and *restricao*.

- **Contiguous lessons**:

  This parameter is represented by *AulasContiguas* node. The child nodes of restriction and objective nodes are *Aula1* and *Aula2* nodes, this two nodes represent two different lessons.

  Both nodes *Aula1* and *Aula2* hold the correspondent discipline code, that must be defined in *disciplines.xml* file, the lesson name and the lesson turn.

- **Separated lessons**:

  This parameter is represented by *AulasSeparadas* node, the child nodes of restriction and objective nodes are *Aula1* and *Aula2* nodes, this two nodes represent two different lessons.

  Both nodes *Aula1* and *Aula2* hold the correspondent discipline code, that must be defined in *disciplines.xml* file, the lesson name and the lesson turn.

- **Overlapped lessons**:

This parameter is represented by *AulasSobrepostas* node, the child nodes of restriction and objective nodes are *Aula1* and *Aula2* nodes, this two nodes represent two different lessons.

Both nodes *Aula1* and *Aula2* hold the correspondent discipline code, that must be defined in *disciplines.xml* file, the lesson name and the lesson turn.

- **Lessons not taught in the same day**:

  This parameter is represented by *AulasNaoLeccionadasMesmoDia* node, the child nodes of restriction and objective nodes are one or multiple *Aula* nodes that represent a given lesson.

  Node *Aula* hold the correspondent discipline code, that must be defined in *disciplines.xml* file and the lesson name.

- **Lessons not taught in the same day with same typology**:

  This parameter is represented by *AulasNaoLeccionadasMesmoDiaTipologia* node. It is only allowed to define restrictions inside this parameter. Inside of each *restricao* node there's one node *tipologia* which defines a typology and subsequently, the disciplines identified by their unique codes.

  AulasNaoLeccionadasMesmoDiaTipologia node allows to define a set of lessons with a given typology that belong to a given unit course and should not be taught in the same day.

- **Free days**:

  This parameter is represented by *DiasLivres* node, in which only restrictions are allowed to be defined. Inside of *Restricao* node must be defined one or more *CodigoProfessor* nodes. They identify the professor by its unique identification code present on *Professores.xml*.

- **Typology order**:

  This parameter is represented by *OrdemTipologias* node, the child nodes of restriction and objective nodes are *Tiplogia1* and *Tipolgia2* nodes as well as one ore more *CodigoDisciplina* nodes.

  Tiplogia1 and *Tipolgia2* nodes represent two different typologies which must be defined in *Tipologias.xml*. The *CodigoDisciplina* node, represents the disciplines by their own unique code.

B.11 *Salas.xml*

This file lists all the classrooms that are available for teaching lessons on FEUP's infrastructure.

Each *Sala* node represents one classroom and it has eight children nodes:

- *Nome*: Has a child text node that holds the classroom's name;
- *Codigo*: Contains a unique code that identifies the classroom;

- *Edificio*: Holds a building name that must be present on *Edificios.xml* file;
- *Piso*: Contains information about the building's floor where the classroom is located;
- *Capacidade*: Holds information about the classroom's capacity in number of seats;
- *CapacidadeExame*: Holds information about the classroom's capacity in number of seats considering that it will be used for examinations;
- *Caracteristicas*: Can appear more than once, informs about a classroom feature, that must be present on *Caracteristicas.xml* file;
- *MargemAceitacao*: Represents the margin that is acceptable when a classroom capacity is exceeded.

B.12 *Tipologias.xml*

It lists all the typologies that a lesson may belong to. Inside each *Tipologia* node, there is a child node called *Nome* which child text node represents one of many possible typologies. There is also an optional node called *Descricao*, it may present a little description about the given typology.

B.13 *Turmas.xml*

Lastly, *Turmas.xml* lists all the classes that exist inside FEUP's ecosystem. Inside each *Turma* node, there are three children nodes:

- *Nome*: Has a child text node that holds the class name;
- *Codigo*: Contains a unique code that identifies the class itself;
- *CodigoPlanoCurricular*: Holds the curricular plan code and its correspondent year separated by a dash.