HEC MONTRÉAL

A heuristic approach based on local search for Course Timetabling and Examination Timetabling at HEC Montréal

par

Su-Min Tan

Sylvain Perron
HEC Montréal
Directeur de recherche

Gilles Caporossi
HEC Montréal
Codirecteur de recherche

Sciences de la gestion
(Spécialisation Data Science and Business Analytics)

Mémoire présenté en vue de l'obtention
du grade de maîtrise ès sciences en gestion
(M. Sc.)

Décembre 2022

Résumé

Les problèmes des horaires sont couverts par une multitude de disciplines. L'intérêt particulier porté au sujet des horaires en recherche peut être attribué à la complexité notoire de ces problèmes et à l'impact des horaires répandu dans l'ensemble des industries. Ce mémoire se centre sur les problèmes des horaires dans le domaine de l'éducation. De nombreuses institutions dépendent de processus manuels dans la création des horaires, favorisant un mode simple et pratique. Toutefois, ce type de processus est essentiellement fondé sur la reconduction et modification d'un horaire précédent pour s'adapter aux nouvelles demandes. Alors, en présence d'un nombre élevé de changements, ces tâches deviennent considérablement difficiles à exécuter. De plus, les institutions adoptant un processus manuel risquent de préserver les erreurs d'un horaire à l'autre. Avec la croissance et l'évolution des institutions, les méthodes traditionnelles, comme les processus manuels dans la création des horaires, deviennent difficiles à appliquer aux situations modernes. Alternativement, on remarque que l'automatisation de tels processus devient graduellement plus facile avec le progrès de la technologie. Ainsi, nous consacrons ce mémoire à aider HEC Montréal à automatiser son processus de création des horaires, mettant en pratique diverses techniques d'optimisation. À HEC Montréal, des différences significatives de contexte poussent à ce que la création des horaires de cours et celle des examens soient traitées comme des sous-problèmes. Dans ce mémoire, on étudie la revue littéraire des problèmes des horaires, contextualisant l'instance en jeu pour explorer l'ensemble des méthodes de solution. Nous justifions notre choix d'adopter une approche heuristique basée sur la recherche locale pour développer un algorithme capable de résoudre les deux sous-problèmes. De nombreux modèles dérivant de différentes stratégies pour améliorer le temps de calcul général de l'algorithme sont présentés. À travers ce mémoire, nous proposons ultimement un algorithme robuste pourvu de simplicité et flexibilité, favorisant une application à de multiples problèmes dans le cadre des horaires.

Mots clés :

problèmes d'horaires universitaires, optimisation, heuristique, recherche locale

Abstract

Scheduling problems are discussed amongst many research disciplines due to their high complexity and heavy impact on almost all industries. This thesis focuses on a subcategory of scheduling problems: timetabling problems in the educational field. While many institutions rely on manual processes for their scheduling activities in favor of simplicity and convenience, these methods have a few drawbacks. In fact, manual timetabling, which is mainly built upon the modification of a previous timetable to comply to new requests, can amount to a tedious task when the number of changes is large. Moreover, the institutions face a great risk of unknowingly carrying forward existing issues. As institutions grow and evolve, the use of traditional means, such as manual timetabling, becomes harder to adapt to modern situations. With today's level of technology, the automation of such processes is greatly facilitated. In this thesis, we seek to help the university of HEC Montréal to automate its timetabling process by leveraging optimization techniques. At HEC Montréal, due to the contextual differences involved in course lectures and examinations, Course Timetabling and Examination Timetabling are treated as sub-problems. Through this thesis, we study the state-of-the-art of timetabling problems, and we contextualize the problem instance at hand to explore the many possible solving methods. We justify the heuristic approach adopted to develop an algorithm based on local search that can solve both Course Timetabling and Examination Timetabling problems. This algorithm is put through numerous tests with different strategies to improve its overall computational time. Ultimately, we propose an algorithm that shows great simplicity and flexibility, in addition to robustness, allowing for application to a variety of problems in the scheduling sphere.

Keywords:

university timetabling, optimization, heuristics, local search

Table of Contents

List of Tables and Figures

1 Introduction

Scheduling is a widely discussed topic as it essentially affects a variety of fields, including business, sports, and education. From employee scheduling to job scheduling, nearly all industries are subject to these time-management problems. The objective of a scheduling problem is presented as follows:

> to solve practical problems relating to the allocation, subject to constraints, of resources to objects being placed in space-time, using or developing whatever tools may be appropriate. The problems will often relate to the satisfaction of certain objectives. (Wren, 1996)

Timetabling problems are the most prominent scheduling problems in the education industry. The quality of the course schedules offered by universities and other higher education institutions can have a material effect on their reputation. These course schedules not only contribute to the overall satisfaction of students and lecturers, but they also help attract prospective talent to the institution. Because of the complex nature of the problems at hand, research efforts in the field have yet to find a general method able to efficiently solve any timetabling problem despite numerous attempts to do so. In fact, the high complexity in this problem led to the conduction of research in a multitude of disciplines, such as artificial intelligence and operational research. The state-of-the-art of timetabling problems includes operational research-based techniques, metaheuristic methods, intelligent novel methods, and distributed multi-agent system-based applications, as we will later discuss in this thesis.

Timetabling is a process that must be completed either at a semestrial or annual frequency for universities. The process becomes increasingly complicated and time-consuming as the institution grows. With the constant expansion of the university, we have been tasked by HEC Montréal, a business school, to automate the process of timetabling for course lectures and examinations. Timetabling problems considerably differ depending on the institution, which calls for different solving methods to be used for different cases. According to McCollum (2006), a great amount of literature is available on timetabling problems, so much that there seems to be a gap between the theory and the practice of this subject as most published papers discuss specific applications. In this thesis, we propose a heuristic algorithm based on local search that is simple and flexible, making it not only applicable to HEC Montréal's specific Course Timetabling and Examination

Timetabling, but also to other problems in the scheduling sphere. The algorithm was developed on Python, but it can be replicated in most other programming languages.

This thesis is divided into four parts. The first chapter will be dedicated to the review of literature on the topic of timetabling problems in the educational field. We will elaborate on the theoretical definition of timetabling, the computational complexity, and the techniques commonly used to solve timetabling problems. A second chapter will serve to formulate the problem, expanding on the context of the problem instance of HEC Montréal, as well as the frameworks created to represent the timetables. The third chapter will introduce the different components of the algorithm proposed. The fourth chapter will present the experimental results of the models given the data simulated, as well as the analysis of said results. Finally, the last chapter will conclude with a discussion of the usefulness of the algorithm we propose for HEC Montréal.

2 Review of literature and problem

2.1 Introduction to timetabling problems

According to Wren (1996), timetabling problems are a special case of scheduling problems. To be exact, timetabling is defined as "the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy as nearly as possible a set of desirable objectives" (Wren, 1996). In both scheduling and timetabling, the resource allocation is restricted by hard and soft constraints. Hard constraints refer to requirements that must be respected; they define the feasibility of a solution. Soft constraints are not related to feasibility. Rather, they refer to preferences and determine the quality of a solution. The number of constraints, as well as the distinction between hard and soft constraints, are greatly influenced by the formulation of the problem instance.

Timetabling problems in the educational field can be divided into three main classes: School Timetabling, Course Timetabling, and Examination Timetabling. Because universities often rely on a combination of Course Timetabling and Examination Timetabling, they are together referred to as university timetabling problems. School Timetabling, also commonly called class-teacher timetabling, is defined as "the weekly scheduling for all the classes of a school, avoiding teachers meeting two classes at the same time, and vice versa" (Schaerf, 2005). Course Timetabling corresponds to "the weekly scheduling for all the lectures of a set of university courses, minimizing the overlaps of lectures of courses having common students" (Schaerf, 2005). Finally, Examination Timetabling is "the scheduling for the exams of a set of university courses, avoiding overlap of exams of courses having common students, and spreading the exams for the students as much as possible" (Schaerf, 2005). However, real-world problems are complex and may fit loosely between multiple classes.

There have been differences in the advancement of research across educational timetabling classes. As put forth by Oude Vrielink et al. (2019), research progress in School Timetabling problems has historically been slower relative to university timetabling problems. This could be attributable to methodological differences in the way studies are conducted. The authors explained that School Timetabling studies are conducted for specific institutions, while Course Timetabling and

Examination Timetabling studies rely on widely adopted benchmarks, allowing for easier comparisons.

Organizational structures of schools and universities often influence the solving methods of timetabling problems. This is reflected in the typology of Course Timetabling problems. Course Timetabling problems can either be post-enrollment-based or curriculum-based, depending on the timing of student enrollment. According to Abdullah et al. (2012), the main difference between the two types of Course Timetabling is that "the post-enrollment course timetabling problems concentrate on students' preferences' […] while curriculum-based course timetabling focuses on lecturers' preferences." In a post-enrollment-based setting, the timetable is set after the students' enrollment. The timetable needs to allow all students to attend the courses in which they have enrolled. In a curriculum-based setting, the timetable is set before the students' enrollment. Students are organized into groups and must follow a predefined set of compulsory and optional courses. Due to their structure, curriculum-based timetables can be defined as "the weekly assignment of a set of lectures for several university courses to specific timeslots and rooms, where conflicts between courses are set according to curricula published by the university and not on the basis of enrollment data" (Abdullah et al., 2012).

2.2 Timetabling: an NP-complete problem

Timetabling problems are hard combinatorial problems. Hoos and Stützle (2004) defined combinatorial problems as problems that "involve finding groupings, orderings or assignments of a discrete, finite set of objects that satisfy certain conditions or constraints." In some cases, a combinatorial problem can pose as a search problem, where the goal is to find a feasible solution. In other cases, it can be viewed as an optimization problem, where an objective function is associated to the solution. The quality of a solution is defined by the value of the objective function, and the best solution is the one with the highest quality. The Boolean satisfiability problem (SAT) and the travelling salesman problem (TSP) are examples of well-known combinatorial problems; Hoos and Stützle (2004) provided additional information on these problems' combinatorics.

Hoos and Stützle (2004) added that "combinations of *these solution components* form the potential solutions of a combinatorial problem." In an educational timetabling context, the solution

components are course lectures or examinations to be assigned to specific timeslots and rooms. This creates a very large number of candidate solutions to explore. In fact, "for most combinatorial optimization problems, the space of potential solutions for a given problem instance is at least exponential in the size of that instance" (Hoos & Stützle, 2004).

Computational complexity is used to classify problems in terms of the amount of resources required to run their algorithms. Mainly, there are two classes of problems: P and NP. The P class refers to problems that are solvable in deterministic polynomial time, meaning that they are efficiently solvable, whereas the NP class relates to problems that are solvable in nondeterministic polynomial time but are verifiable in deterministic polynomial time. This implies that all P problems are contained in NP (or $P \subseteq NP$), because everything that is solvable in deterministic polynomial time is also ultimately solvable in nondeterministic polynomial time. Moreover, NP-hard problems are problems that are at least as hard as the hardest NP problems. In such way, NP-complete problems are ultimately problems that are both NP and NP-hard. In simpler terms, they are the hardest problems contained in the NP class. Timetabling problems are generally considered as NP-complete (Cooper & Kingston, 1995).

Because the solutions of NP-complete problems can be verified in deterministic time, the relationship between P and NP brings us to an important topic in computational complexity theory: the P vs. NP problem. The question is such that if the solution to a problem can be verified in deterministic polynomial time, is it also possible to solve it in deterministic polynomial time? In other words, we want to know if $P = NP$ is true. If $P = NP$ was proven to be true, it would mean that every problem verifiable in deterministic time can also be solvable in deterministic time. Hence, we would be proving that all NP problems can be solved efficiently in a computational complexity standpoint. Hoos and Stützle (2004) stated that "it suffices to find a polynomial time deterministic algorithm for one single NP-complete problem to prove that $P = NP$." However, to this day, no efficient algorithm to solve a NP-complete problem has yet been found. This just shows how complex timetabling problems truly are and how difficult they are to solve.

2.3 Solving techniques explored

Numerous methods have been studied in attempt to find optimal solutions for educational timetabling instances. A survey by Babaei et al. (2015) breaks down these methods into the following categories: operational research-based techniques, metaheuristic methods, intelligent novel methods, and distributed multi-agent system-based applications. The next sections are dedicated towards the different methods' description and application on diverse educational timetabling problems with a focus on university timetabling, spanning Course Timetabling and Examination Timetabling.

2.3.1 Operational research techniques

Operational research (OR) relates to the application of mathematical principles to management or business problems, mainly to improve decision-making. The most explored OR methods for timetabling problems include graph colouring (GC), integer programming or linear programming (IP/LP) and constraint satisfaction programming (CSP).

2.3.1.1 Graph colouring (GC)

Welsh and Powell (1967) suggested reducing timetabling problems to graph colouring. The GC problem consists of an assignment of colours to vertices of a graph such that no pair of vertices connected by an edge share a colour. In a scheduling context, the vertices are events, the edges are event conflicts, and the colours are timeslots. To illustrate, let us suppose that the only events are course lectures in the Course Timetabling case. If two vertices are connected by an edge, the corresponding courses have common students or lecturers and thus should not have lectures simultaneously. Because they should be assigned to different timeslots, the corresponding vertices' colours must be different. Given a graph G, this problem typically looks for a solution with the least number of colours, i.e., with the chromatic number of G, but Welsh and Powell (1967) introduced a relaxation with the idea of adding an upper bound to the chromatic number. This upper bound allows to simplify the resolution of GC for instances where the minimum number of colours is not necessarily sought after. In fact, rather than a timetable with the minimum number of timeslots, timetabling problems generally look for a timetable with no event conflict that fit within a given number of timeslots. GC has also successfully been applied to School Timetabling (De Werra, 1985), Examination Timetabling (Welsh & Powell, 1967; Redl, 2004) and Course

Timetabling (Dandashi & Al-Mouhamed, 2010; Razak et al., 2010). Graph colouring is primarily used for timetabling problems that are predominantly restricted by event conflicts. However, because there are constraints that are outside of the graph colouring's scope in higher education timetabling problems, heuristics incorporating graph colouring are more commonly used. Asham et al. (2011) proposed a heuristic combining graph colouring and genetic algorithm to decompose the Course Timetabling problem.

2.3.1.2 Integer programming and linear programming (IP/LP)

In a timetabling context, Babaei et al. (2015) referred to linear programming as an "efficient assigning of limited resources to the specified activities in order to maximize the interest and minimize the cost." In fact, integer programming (IP) and linear programming (LP) correspond to mathematical optimization of problems. In integer programming, the variables only take integer values. In linear programming, the problem is represented with linear relationships. And linear integer programming (ILP) combines the two. The mathematical approach has been successfully applied to a wide range of real-world problems across business, medicine, and many more. Instances of Course Timetabling have been solved with IP by Aubin and Ferland (1989), Daskalaki et al. (2004) and Bakir and Aksop (2008). Although IP/LP have proven to be quite successful in solving combinatory optimization problems, computational difficulties may occur. In fact, IP/LP is a method that has shown great success in finding globally optimal solutions at the expense of considerably slow convergence time. Therefore, this approach is mostly used for small-sized problems because the heavy computations in larger-sized problems may require more powerful computers, which are not always accessible. For large scale problems, heuristics based on IP/LP are more frequently used. In the same vein, Daskalaki and Birbas (2005) presented a relaxation procedure to the optimization of the IP formulation of a university timetabling problem. In this two-stage relaxation procedure, the constraints requiring the heaviest computations are relaxed at the first stage, and then recovered at the second stage. This allows the model to find an initial solution more easily at the first stage, and then find an optimal solution at the second stage.

2.3.1.3 Constraint satisfaction programming (CSP)

Many operational research problems, including timetabling problems, fall within the general framework of constraint satisfaction programming. CSP relates to computer programming, rather

than mathematical programming. The general model for constraint-satisfaction programming is as follows: $CSP = (V, D, C)$ where $V = \{v\_1, v\_2, \dots, v\_n\}$ is a finite set of variables, $D = \{D\_1, D\_2, \dots, D\_n\}$ is a finite set of domains where D_i is a set of values for the variable v_i, and $C = \{c\_1, c\_2, \dots, c\_n\}$ is a finite set of constraints restricting the variable values (Müller, 2005). The goal is to find a consistent assignment, i.e., to assign values to each variable such that all constraints are satisfied. Generally, CSP can be solved with either consistency techniques or search algorithms. Amongst consistency techniques, arc-consistency is a process that "ensures any valid value in the domain of a single variable has a valid match in the domain of any other variables in the problem" (Zhang & Lau, 2005). Most search algorithms used to solve CSP are systematic, meaning that they start with an empty solution and systematically assign possible values to the variables. Because they can go through the entirety of the possible values, they must find a solution, or prove that there is no solution. Müller (2005) proposed an iterative forward search algorithm, which combines systematic search to the local search approach, to solve the CSP formulation of a curriculum-based course timetabling problem at Purdue University.

2.3.2 Metaheuristic methods

Metaheuristics methods are methodological approaches based on an iterative process in the quest of improvements. These methods are known to return good quality solutions without proof of optimality. They are mainly split into two types of methods: single-based and population-based. Single solution-based metaheuristics use a single candidate solution to analyze the problem, rather than a population. A single candidate solution is initially chosen according to some criteria and evolves iteratively. Essentially, single-based metaheuristics are local search algorithms. Population-based methods choose multiple candidate solutions to form an initial population. At each iteration, a selection mechanism is used on the current population to select the best candidate solutions. The selected candidate solutions undergo some changes such that there is improvement.

Amongst metaheuristics methods, variable neighbourhood search (VNS), tabu search algorithm (TS), simulated annealing (SA) and genetic algorithm (GA) are most used to solve timetabling problems.

2.3.2.1 Variable neighbourhood search (VNS)

Variable neighbourhood search is an approach based on guided change of neighbourhoods in the search process. It "explores increasingly distant neighbourhoods of the current [best] solutions, and jumps from this solution to a new one if and only if an improvement has been made" (Mladenovic & Hansen, 1997). Many variants of VNS have been developed, such as variable neighborhood descent (VND) which uses multiple neighbourhoods in the local search, and variable neighbourhood decomposition search (VNDS) which systematically decomposes large scale problems into subproblems. According to Hansen et al. (2017), basic VNS is composed of three stages: *shaking*, *local search* and *move*. *Local search* allows to find the optimum in the neighbourhood of a solution. *Move* compares the neighbourhood's local optimum to the current best solution. Finally, *shaking* is used to prevent local optimum traps. Basic VNS starts with a defined set of neighbourhood structures in a specific order. The VNS systematically searches through every neighbourhood of the current best solution in a predefined order. If a local optimum that is better than the current best solution is found, it replaces the latter. Then, the neighbourhood search starts over with the new current best solution, and so until no further improvement can be found. An instance of Examination Timetabling has been solved using VNS by Ayob et al. (2006). Borchani et al. (2017) developed a VND to solve a Course Timetabling problem instance. Many have developed hybrid heuristics combining a variant of VNS with other methods such as genetic algorithm (Burke et al. 2010) and tabu search (Vianna et al., 2020).

2.3.2.2 Tabu search (TS)

Tabu search is a metaheuristic "based on steepest descent search, as it tends to explore the search space by not re-interpreting recent moves" (Oude Vrielink et al., 2019). In fact, TS utilizes a tabu list, i.e., a list of moves that are prohibited. A move generally stays within the tabu list for a given number of iterations, which corresponds to the tabu tenure. Aspiration criteria can also be used, where a move leading to a certain candidate solution that meets the aspiration criteria can be accepted regardless of the tabu list. This aspect is mostly used to prevent stagnation from an overwhelming tabu list. The algorithm starts with a single candidate solution as the current best solution. It generates a set of neighbours (neighboring candidate solutions to the current best), and neighbours created from moves within the tabu list are removed from the set. Then, if the best neighbour out of the set betters the current best solution, it replaces the latter. The tabu list is

updated by removing all moves that is past the tabu tenure. Also, the last move used to obtain the current best candidate solution is added to the list to prevent cycling. Using TS, Alvarez et al. (2002), Hertz (1991) and Lu and Hao (2010) each presented a solution to different course instances, whereas Di Gaspero and Schaerf (2000) solved an examination instance.

2.3.2.3 Simulated annealing (SA)

According to Oude Vrielink et al. (2019), simulated annealing "aims to search for a wider area of search space in which worse steps are accepted and allows for a more extensive search for the optimal solution [than other single-based metaheuristics]." The SA algorithm is a probabilistic process with a variable temperature parameter mimicking the annealing process of metalworking. The temperature has a high value at the beginning of the search process and progressively decreases according to a given reduction rule. This parameter affects the acceptance probability of a candidate solution. If the candidate solution is not deteriorating, the probability of acceptance is 1, else the acceptance is determined by a function of the variation in cost and the current temperature. In fact, the function tells us that, at high temperature, the algorithm focuses on exploration, and at low temperature, it focuses on improvements. The algorithm is initiated with a candidate solution. At each iteration, a candidate solution is randomly generated from the current solution. If the candidate solution is better than the current solution, it replaces the latter. Else, a number between 0 and 1 is randomly generated, and is compared to the acceptance probability. If the number generated is lower than the acceptance probability computed, the candidate solution is accepted, and replaces the current solution. In the context of educational timetabling, simulated annealing has been successfully used for both examinations (Burke et al., 2004; Thompson & Dowsland, 1996) and course lectures (Bellio et al., 2016).

2.3.2.4 Genetic algorithms (GA)

Genetic algorithm is a population-based approach inspired by the concept of natural selection and evolution. The initialization of the GA refers to generation of several individuals (candidate solutions) as part of the initial population. The population maintains a fixed size throughout the algorithm. Each individual has a chromosome which is represented by a set of genes (parameters). A fitness evaluation is used to assess the quality of each individual. In the selection process, individuals are chosen to enter the mating pool according to their fitness score. Then, the mating

process is performed by randomly choosing parents (pairs of individuals) from this subpopulation. Depending on a certain crossover probability, there might be an exchange of genes between the parents to create an offspring, otherwise no change is made. The new offspring, after crossover, is randomly chosen to perform mutation, which corresponds to modification of some genes. Mutation is needed to ensure diversity in the population, or to avoid early state convergence. Genetic algorithm techniques have been used to solve university timetabling problems, as demonstrated by Burke et al. (1994), Erben and Keppler (1995) and Alsmadi et al. (2011).

2.3.3 Intelligent novel methods

Babaei et al. (2015) refer to hybrid methods, fuzzy approach, hyper heuristics, knowledge-based methods, and clustering algorithms as intelligent novel methods. These methods have been more recently developed and more frequently used nowadays. The following sections are dedicated towards hybrid algorithms and fuzzy approach in the context of educational timetabling.

2.3.3.1 Hybrid algorithms

Hybrid algorithms combine multiple approaches to solve a common problem. They aim to benefit from the different attributes each method offers by compensating their individual shortcomings. The use of hybrid algorithms is becoming increasingly popular because of their good performance in solving complex problems. A few examples of hybrid algorithms have already been given in the previous sections. In fact, these algorithms are very diverse, mainly because their design highly depends on the problem instance. For example, local search algorithms and evolutionary approaches can be combined (Abdullah et al., 2007; Kohshori & Abadeh, 2012), and tabu search and variable neighborhood search can be hybridized as well (Muklason et al. 2019).

2.3.3.2 Fuzzy approach

The fuzzy approach makes use of fuzzy logic, which is born from the observation that decision is frequently made on uncertain and imprecise information. Novak et al. (2012) informally defined fuzzy logic as “a special many-valued logic addressing the vagueness phenomenon and developing tools for its modeling via truth degrees taken from an ordered scale. It is expected to preserve as many properties of classical logic as possible.” Because fuzziness still needs to be contained in logic, methods with a fuzzy approach must follow a fuzzy system. Research has shown that fuzzy

approaches constitute very effective and relevant methods for real-world cases (Asmuni et al., 2009; Asmuni et al., 2005; Chaudhuri & Kajal, 2010; Golabpour et al., 2008).

2.3.4 Distributed multi agent system-based applications

The approach based on multi-agent architecture has been gaining more attention recently, partly due to its innovating way to decompose large instances of complex timetabling problems into smaller subproblems. In a multi-agent system, multiple agents cooperate in a computerized system to solve problems that are too complex to solve individually (Obit et al., 2011). An agent is anything that can observe the environment through sensors, and that can perform actions upon the environment. In the context of a decentralized university where each department create their own course schedule by negotiating for resources with the other departments, Di Gaspero et al. (2004) proposed a system based on a three-agent architecture. Each department has three cooperative agents: *Solver* for local search of departmental timetable, *Negotiator* for resource negotiation between departments, and *Manager* for information management.

2.4 Comparison between methods

Educational timetabling problems have been widely studied because of their complexity, and many approaches have been proposed to solve these problems. There is not a universally best method to solve university Course Timetabling and Examination Timetabling problems, as each method has its strengths and weaknesses. According to Chen et al. (2021), operational research-based methods, single-based metaheuristics, and hybrid algorithms are the most popular solving techniques specifically employed in real-world Course Timetabling.

Operational research-based methods, such as GA, IP/LP and CSP, are rather easy to implement, but they do not particularly show good efficiency in solving university timetabling problems. To be more precise, they do not perform well on very large instances since the complexity of these methods increases as the number of students and universities increases (Babaei et al., 2015). In fact, they seem good at generating feasible solutions but lacking at finding high quality solutions, especially in a large-scale context. As for metaheuristics, these methods have been increasingly popular in timetabling research, mainly because they can easily adapt to different constraints to generate high quality solutions and cover a very wide variety of optimization problems. But it was

noted that single-based metaheuristics, such as VNS, TS, and SA, mainly focus on exploitation rather than exploration, while population-based algorithms, like GA, are prone to premature convergence and tend to require a lot of computational time. It seems like recent research has been focused on single-based rather than population-based metaheuristics (Oude Vrielink et al., 2019). The attempts to find a synergy between exploitation and exploration have attracted attention to hybrid algorithms. While hybrid algorithms are more costly and harder to implement than metaheuristics, they generally perform well in the exploration of solution search space.

3 Problem formulation

3.1 Context

HEC Montréal is a business school in Montreal, Quebec, Canada founded in 1907. It offers over 1,200 courses across more than 100 programs, at the undergraduate, graduate, and post-graduate level. There are courses available for the bachelor's degree (BAA) and certificate (CERT) at the undergraduate level; Master of Science (MSC), Master of Business Administration (MBA) and specialized graduate diploma (DESS) at the graduate level; and Doctor of Philosophy (PHD) at the post-graduate level. Courses are offered primarily in French, but they can be available in English and Spanish. In the fall of 2020, HEC Montréal reported a student body size of 15,180, including 4,144 international students from 144 countries. At this scale, the effective scheduling of course lectures and examinations has become an increasingly challenging task for HEC Montréal. We express the situation as a timetabling problem and seek to help the institution derive more efficient scheduling outcomes by leveraging optimization techniques. Here, Course Timetabling and Examination Timetabling are treated as subproblems. We start by describing HEC Montréal's current timetabling process, to then expand on the distinctive characteristics of Course Timetabling and Examination Timetabling in this specific context.

3.1.1 HEC Montréal's current timetabling process

There are three main actors involved in HEC Montréal's current timetabling process: the academic departments, the school administration, and the school organization department. Each of the eleven academic departments manages its course offering depending on its estimated demand, as well as its department lecturers' availabilities. The school administration is responsible for approving the changes put forward by the academic departments. And ultimately, the school organization department is responsible for creating the timetables.

At HEC Montréal, both Course Timetabling and Examination Timetabling are manual processes; the previous year's semestrial timetables are reused and manually adjusted to reflect the required changes. This process takes over five months to complete and produces six timetables, i.e., one course timetable and one examination timetable for each of the three semesters (fall, winter, and summer) offered each year.

We now provide insight into how HEC Montréal operationalizes its current yearly scheduling process. The school organization department primarily uses *Oracle PeopleSoft*, a Human Capital Management (HCM) software suite, to manage course scheduling. The process begins in January, where the school administration holds a meeting to discuss changes on the course curricula. The school organization modifies the course offering on *Peoplesoft* considering the changes discussed by the school administration. Through *Peoplesoft*, the school organization department extracts all course labels from each semester of the previous year. The next step is to extract all schedules (i.e., represented in date and time) and room assignments of every course's lecture and examination for each semester of the previous year. Here, the MBA program is treated separately because of structural differences. Then, the school organization department uses *Textpad*, a word editor, and *Microsoft Excel* spreadsheets to manually unassign rooms, such that only the schedules are retained. In cases where a course needs a specific room, the desired room is also preserved. This is commonly the case for online courses where a virtual room is always assigned. This step essentially allows for room assignment to start anew for the upcoming year. Then, the lectures and examinations are separately imported back into *Oracle PeopleSoft*. For each of the three semesters, the school organization updates the course information (e.g., course title, repertory number, and maximal registrations). Subsequently, a manual smoothing is performed to remove exemptions. An exemption is a lecture that is prevented from happening at its usual time by events such as holidays and lecturer unavailability; it is specific to a semester. For lectures, this manual smoothing allows for a course to be represented by a single timeslot: it reduces the date-time schedule into a weekly schedule where each course is assigned to a timeslot within a week. The course and examination schedules are then shared through *SharePoint* to the other parties (i.e., the academic departments and the school administration) in mid-February. Then the academic departments have four weeks to request for changes, additions, and removals of courses, which will need the approval of the administration. It is also through *SharePoint* that any preference is communicated. The preferences of a course or examination can relate to timeslots, rooms, and other factors, such as other related courses, location of building and room layout. Once mid-March, the school organization updates the exemption table for the upcoming year. The schedules and the exemption tables are imported to *Timetabler* as an intermedium to *Enterprise*, where they are merged to create three semestrial date-time schedules for the upcoming year. Then, the schedules are transferred

back to *Oracle PeopleSoft.* An *Excel* forecasting tool is used to validate the availability of rooms for each timeslot. It basically compares the number of rooms available to the number of courses assigned for each timeslot on different levels (e.g., room layout to course's requested layout). Manual adjustments for additional requests from *SharePoint* are made until consensus, which allows the schedules to be definitive for mid-May.

The process described above is categorized as a manual one. Manual timetabling processes remain widely used by institutions to solve scheduling problems. These manual timetabling techniques offer some benefits, mostly centered around simplicity and convenience. In instances that call for a small number of year-to-year adjustments, it can be significantly simpler to directly modify an existing timetable to satisfy the new requests, as opposed to creating an entirely new timetable. This leads many educational institutions to rely on manual processes for their scheduling activities, as those generally display low year-to-year variability. However, manual timetabling also faces limitations. It can prove tedious when a large number of adjustments are required, and institutions face the risk of carrying forward solvable issues that can first appear unavoidable or too difficult to solve manually. The advancement of computer technology greatly facilitates the development and implementation of automated timetabling solutions, allowing more institutions to enjoy its numerous benefits, namely reduced completion time, personnel employed, and human error involved through manual timetabling process.

3.1.2 Course Timetabling context at HEC Montréal

Course Timetabling at HEC Montréal is curriculum-based. We define a curriculum as a pre-determined set of compulsory and optional courses. The use of curricula allows to produce course timetables before the moment students enroll in courses. Each student follows a given curriculum depending on their program of choice and relevant academic progress. Students generally need to complete all compulsory courses and most optional courses in their curriculum. Each course is associated to a number of credits for its completion, and students are fully responsible for enrolling in the appropriate courses to obtain the required number of credits for graduation.

HEC Montréal must provide a course timetable that is entirely conflict-free for every curriculum at each semester to ensure that students can feasibly obtain all necessary credits within their

program's expected time frame. To help achieve this, it employs three distinct types of constraints: exclusivity groupings, timeslot preferences and similarity groupings.

Exclusivity groupings ensure that courses within a grouping are not assigned to a same timeslot. These groupings are created based on the curricula to make sure that the students can enroll in all their courses. No differentiation is made between compulsory courses and optional courses within exclusivity groupings. This is because although not all optional courses are required, they need to be available for all students following the curriculum. Since HEC Montréal's room capacity is limited, a course is generally offered more than once per semester. Courses that are offered more than once in a semester are referred as multi-sectional courses; otherwise, they are called uni-sectional courses. Exclusivity groupings mainly concern uni-sectional courses within a curriculum. This type of grouping is also used to prevent students from enrolling in both a course and its prerequisite course during the same semester. In parallel, the allocation of lecturers to courses at HEC Montréal is completely managed by the academic departments in preparation to the Course Timetabling process. Because lecturers are pre-assigned to courses, it is important for the course timetable to be conflict-free from the lecturers' standpoint as well. Additional exclusivity groupings are formed to ensure that the lecturers can feasibly attend to all their allocated courses. This is mostly intended for courses that can only be taught by specific lecturers. In addition to exclusivity groupings, timeslot preferences are introduced, mainly to provide a satisfactory teaching schedule to lecturers. Academic departments can also request some courses to be scheduled at a certain time of the day and/or on a certain day of the week through timeslot preferences. Finally, similarity groupings are used to ensure that specific courses are assigned to the same timeslot. In fact, some compulsory courses, commonly core courses, are shared between curricula, and to facilitate lecturer allocation, multi-sectional compulsory courses are generally requested to be scheduled at a same time, given that the lecturers are different.

In addition to the scheduling constraints laid out above, the timetables must also consider room constraints: room capacity, room occupancy, room layout (layout preference and layout necessity) and building location of room. Capacity refers to the constraint that a room assigned to a given course must have the physical capacity to seat the number of students. Because the number of students enrolled to a course is not yet available at the time of the timetable's creation, HEC

Montréal uses course norms as proxies. The norm of a course corresponds to the maximal number of students allowed to be enrolled, itself based on historical data and adjusted according to projections for the current semester. The norm is determined before initiating the timetabling process. In addition to room capacity, HEC Montréal seeks to maximize occupancy rates by allocating courses with small norms to low-capacity rooms and vice-versa. Two constraints can be derived from room layouts: layout preference and layout necessity. While basic layout can be requested by a lecturer's preferences, special layout can be required by the academic department because of a course's nature. Courses with layout necessities (i.e., courses requiring a room with a certain special layout) need more attention than courses with layout preferences (i.e., courses requesting a room with a preferred basic layout). It is important that the only courses assigned to rooms with special layout are courses that require the specific layout in question. The trading room, negotiation room and computer labs are rooms with special layouts, whereas mobile, multi-level, and mobile islands pose as basic layout. Finally, HEC Montréal's two buildings, Côte-Sainte-Catherine (CSC) and Decelles (DEC), are taken into consideration. Lecturers and academic departments may demand for certain courses to be located at a specific building.

3.1.3 Examination Timetabling context at HEC Montréal

We now turn to examination timetables, with an emphasis on the contrast with the course timetable process outlined above. Examination timetables are created separately from course timetables, and are usually constructed at a later stage. Examination periods and lecture periods do not overlap, and examinations have different constraints from course lectures. Examination periods generally last for about two weeks when lectures do not occur; the midterm period comes at the middle of the semester, around the eighth week, whereas the final period begins after the last day of lecture. Generally, the midterms are split according to programs: the first week is dedicated towards BAA and DESS, and the second week is for MSC, CERT and PHD. It is important to note that examinations are overseen by supervisors, and not by lecturers. This leads to a different dynamic in the constraints. Furthermore, since Examination Timetabling is a problem of smaller scale than Course Timetabling, it can support a larger number of constraints, as well as more precise ones.

HEC Montréal aims to avoid examination conflicts between core courses and other compulsory courses of a curriculum. This implies that courses follow similar timeslot exclusivity groupings

for lectures and examinations. It is also possible for academic departments to request that several courses do not have their examinations at the same time, creating additional timeslot exclusivity groupings. Because examinations of a course tend to be similar through all its variants, it is necessary that the examinations of the same courses, no matter the language variant, are scheduled at the same time. This causes similarity groupings to be different from lectures to examinations. In comparison to Course Timetabling, the constraints in Examination Timetabling are more specific to the academic program. In BAA, students of the core curriculum cannot have more than one examination per day. The courses involved form one-day exclusivity groupings. In DESS, examinations of uni-sectional courses must be scheduled at the same time of the day as their lectures, and examinations of multi-sectional courses must be scheduled in the evening or during weekends. In CERT, examinations also need to be scheduled in the evening or during the weekends. In MSC and PHD, for uni-sectional courses, the examinations need to be at the same timeslot as lectures. Also, for uni-sectional courses taught in English, the final examinations need to be in the morning. The MSC and PHD courses with lectures in the evening need to schedule the examinations for an evening timeslot. These requests are managed through timeslot preferences. Timeslot prohibitions are used for MSC and PHD finals where a timeslot (morning or afternoon) without examination on the first day is demanded. MSC and PHD also require that there is a day without examinations between two final examinations of core courses, and it is managed through two-day exclusivity groupings.

Concerning the room assignment for examinations, it is required that for each seat occupied, one seat is left open. This implies that the capacity of a room is cut down in half for examinations. Because the institution is limited in large rooms, the courses are split into smaller divisions (called sub-sections) beforehand, and these sub-sections are the events assigned to time-rooms. As mentioned, examinations are not supervised by lecturers. However, their assistance might be required, suggesting that examinations of the same courses must be within a same building for easier accessibility. For examinations at HEC Montréal, it is also common for there to be a time interval (typically a minimum of 48 hours) between the exam date and the last lecture of the course. Finally, in contrast to Course Timetabling where room layout preferences come from the lecturer, the room layouts required in Examination Timetabling are primarily influenced by the form of the examinations.

3.2 Framework

We have established that course lectures and examinations at HEC Montréal are subject to different sets of constraints. As such, we chose to express course lectures and examinations separately, effectively treating them as sub-problems. In addition, we note that course lecture timetables and examination timetables operate in different timeframes. We settle upon representing them using distinct timetabling frameworks. The following sections describe the differences between the frameworks used to represent a course timetable and an examination timetable.

3.2.1 Course framework: a standardized weekly timetable

We now discuss the logical use of a standardized weekly timetable for courses. Diving deeper in the construction of such a framework, we touch on the difference between typical and atypical courses, and discuss its building blocks which rely on the concept of time-rooms.

Course lectures at HEC Montréal are recurrent events. They occur repeatedly at a weekly interval, but can be exceptionally interrupted by exemptions. As a reminder, an exemption is a lecture that is prevented from happening at its scheduled time by an unusual event (e.g., holidays or lecturer unavailability). The minimal week-to-week changes in the lectures' recurrency patterns suggest that it is possible to reduce the course schedule to a standardized weekly timetable (as broadly supported by the literature on curriculum-based Course Timetabling). This standardized timetable represents a typical week in a context without exemption, and is solved such that each course is assigned to a specific room at a given timeslot, in accordance with the relevant constraints. This reduced representation allows for an easier solving. From there, the standardized weekly course timetable is expanded in a full course schedule (generally of 12 weeks) and manual adjustments are made to account for and reflect any relevant exemptions.

Courses that fit within the standardized framework are considered typical. We regard a typical course as one with lectures occurring once per week for a period of 12 weeks, occupying the same room at the same time. Courses that come close to this definition may also be considered typical, but require explicit manual adjustments. Courses that do not fit within the proposed framework are considered atypical. Atypical courses generally can be manually incorporated in the final

course schedule because of their small number; this includes special cases where several atypical courses share similar behavior. As an alternative to a manual solution, if a special case is large enough, the algorithm we propose can serve as an inspiration to one that fits a special case's framework. For instance, weekend courses that are recurring in a fixed pattern can fit within a weekend framework, and they can be solved by a similar algorithm.

Each course is assigned to a time-room, i.e., a room at a given timeslot, within the framework. Each lecture timeslot can be represented using two components: day and time. The lectures at HEC Montréal are typically given from Monday to Friday. The typical lecture times are at 8:30 – 11:30, 12:00 – 15:00, 15:30 – 18:30 and 18:45 – 21:45. Typical course lectures have a duration of three hours and time is pre-allocated between lecture times to allow for transportation and break between consecutive courses for both students and lecturers. With that being said, the lecture timetable has 20 timeslots, each with 67 rooms; there is a total of 1,340 time-rooms available for course lectures.

3.2.2 Examination framework: a full-length timetable

Unlike course lectures, examinations are not recurring events within a semester. Because of the absence of recurrent patterns, midterm and final examinations must be represented in full-length timetables. As mentioned previously, the midterm weeks are split in two according to programs. The two midterm examination periods are each 7-days long, and the final examinations period is 14-days long. Since the two midterm examination periods and the final examination period do not overlap, the three of them can be solved separately, so they require separate frameworks.

Examinations take place in rooms that are offered for course lectures, and can occur across weekdays or weekends. However, weekdays and weekends offer a different number of timeslots. Weekdays provide three daily timeslots, whereas weekends provide two daily timeslots. Due to the yearly variability in examination dates, the total number of timeslots per examination period is variable. This carries the implication that the creation of an examination framework requires specific dates as inputs. It would also imply that an examination schedule cannot be reused for another year.

4 Application of solving methods

4.1 General algorithm

Combinatorial optimization problems, such as timetabling problems, are hard to solve, and finding optimal solutions can be time-consuming. The literature review in **Chapter 2** revealed that timetabling problems are considered NP-complete. However, the complexity of a timetabling problem instance heavily depends on the constraints present. So far, we can only strongly assume that the instance described is a problem of NP-complete complexity. The literature review also highlighted that metaheuristics are well-suited to solve a wide variety of combinatorial problems, in part due to their flexibility. These methods can allow for sufficiently good solutions to be generated within a reasonable timeframe for cases where optimal solutions are not necessary. In this context, a sufficiently good solution refers to a conflict-free timetable that generally satisfies students and lecturers' preferences.

To solve HEC Montréal's Course Timetabling and Examination Timetabling problems, we propose a heuristic algorithm based on local search. This heuristic algorithm searches the solution space by only exploring complete and feasible solutions, i.e., complete solutions that satisfy all hard constraints. We clarify on the definition of a complete and feasible solution in **Section 4.2** when we expand on the topic of constraints. Figure 4.1 shows the pseudo-code of the algorithm proposed. The heuristic algorithm starts with the initial state s_0, which corresponds to an initial timetable solution defined by the user. The algorithm uses a penalty function to iteratively evaluate the states. At each iteration, a transformation t is generated and applied on a copy of the current state, creating a temporary state $stemp$. To assess the transformation, the temporary state is evaluated by a penalty function, and its value is compared to the penalty value of the current state (i.e., the state before the transformation). The general rule is that a transformation is accepted if it improves the current value (i.e., the penalty value of the temporary state must be lower than the penalty value of the current state). If the transformation is accepted, the temporary state replaces the current state, else the current state remains unchanged. These steps are repeated until the stopping criteria are fulfilled.

- Let $s = s\_0$
- Repeat
  - Generate transformation t to create a temporary timetable $stemp \leftarrow s + t$
  - If $c(stemp) < c(s)$:
    - $s \leftarrow stemp$
- Until a stopping criterion is fulfilled
- Output: final state s

Figure 4.1 Pseudo-code of the algorithm

While course lectures and examinations have different constraints, they adhere to a same algorithm. The next sections expand on each component of the algorithm used for Course Timetabling and Examination Timetabling.

4.2 Constraints

Soft constraints and hard constraints can vary from a case study to another, and they strongly depend on the problem instance. The algorithm proposed is constructed in a way that the hard constraints are pre-emptively respected, by constantly working with complete and feasible solutions. We impose, as a hard constraint, that every course (lecture or examination) must be assigned to an available time-room. We consider all other constraints as soft constraints, viewing them as goals, and any of their violation carries a penalty. Although some constraints technically pose as hard constraints, we consider them as components of the penalty function (i.e., as soft constraints) to be able to evaluate them. By treating them as soft constraints, the optimization problem is subject to a less strictly constrained search space. This, in turn, allows for the algorithm to freely explore the space and to avoid being trapped in disconnected zones. To most prevent the violation of the soft constraints that technically pose as hard constraints, they are significantly more penalized than other soft constraints.

Since we assume that optimality is not essential, the weights attached to the constraints do not need to be optimal and they can be set manually. When defining the numerical values of a set of weights, two factors need to be considered by the user: scale and trade-off. The scale corresponds to the range of value that the set of weights can take, whereas the trade-off refers to the priorities between the goals. This means that the weights of more important goals need to be set to higher values. In parallel, the user needs to be mindful of the numerical difference between the weights, as it

influences the symmetry of the combinatorial problem. To be more precise, if the difference between the weights is too small, it might increase the number of equivalent solutions. To prevent this, the weight values need to put enough emphasis on the qualitative differences between the goals. Keeping this in mind, we move on to expand on the penalty mechanism of each constraint in Course Timetabling and Examination Timetabling.

4.2.1 Course Timetabling constraints

Course Timetabling at HEC Montréal is focused on the allocation of timeslots and rooms to course lectures. In this special case of allocation problem, there is an interaction of time and space. The course timetables need to respect time constraints (timeslot preference, similarity grouping and exclusivity grouping) and room constraints (room capacity, room occupancy, building preference, layout necessity and layout preference). Thus, there are eight Course Timetabling constraints, and their order of importance (from most important to least important) is as follows: exclusivity grouping, layout necessity, room capacity, similarity grouping, timeslot preference, layout preference, building preference, room occupancy.

4.2.1.1 Timeslot preference

Timeslot preferences allow courses to integrate lecturers' availability and to comply to the academic departments' requests. Each course is allowed up to a definite maximal number of timeslot preferences, and the default number is arbitrarily set to 4. These preferences must be listed in an ascending order of importance (from most important to least important). The penalty weights associated to the timeslot preferences of a course are ascending but cumulative. To evaluate this constraint, the timeslot in which a course is assigned is compared to each of its preference until a preference is satisfied by the timeslot, or until all preference has been compared. Every preference that is not satisfied adds a penalty to the course. This means that satisfying the first preference leads to no penalty and satisfying no preferences cumulatively leads to the highest penalty. The ascending weights put emphasis on the satisfaction of the first preferences, and gradually load the penalty as the preferences are not satisfied.

4.2.1.2 Similarity grouping

Similarity groupings are used for courses of a same grouping to be assigned to a same timeslot. This type of grouping mainly allows the academic departments to manage human resources more easily towards the courses they offer. A mechanism is put in place to make sure that no similarity grouping is repeated. More importantly, the mechanism also ensures that there is no overlapping of grouping. This implies that if a grouping shares courses with another, they ultimately merge into a same grouping. For a similarity grouping, a penalty is applied to every course that is not assigned to the same timeslot as the grouping's majority.

4.2.1.3 Exclusivity grouping

Exclusivity groupings allow courses of the same grouping to be assigned to different timeslots. These groupings are created to ensure that students can enroll to all courses of their curriculum. A mechanism is integrated so that no exclusivity grouping is completely contained within another. If a grouping is completely contained within another, the former is removed. The penalty of a grouping is calculated through the number of unique timeslots within a grouping, which must be equal to the number of courses within a grouping. If not, the difference between the two is penalized.

4.2.1.4 Room capacity and room occupancy

The capacity of a room assigned to a course must be higher or equal to its norm, or else a penalty of fixed value is applied to the course. Additionally, rooms are evaluated on their room occupancy rate. Courses with low occupancy rate are more penalized than courses with high occupancy rate. This is meant to reduce the number of empty seats in occupied rooms, where the number of empty seats corresponds to the difference between the norm of a course assigned and the capacity of the room. For each course, we find the ratio between the number of empty seats and the capacity of the room assigned. This ratio is then multiplied by the room occupancy penalty weight. Because this constraint is of low priority, a very small value is set for the penalty weight.

4.2.1.5 Building preference

The building of the room assigned is compared to the course's building preference. A penalty only happens when the room's building does not respect the course's building preference.

4.2.1.6 Layout necessity and layout preference

The layout of the room assigned needs to satisfy the layout necessity or the layout preference of a course. The satisfaction of a layout necessity differs from that of a preference. It is considered essential to satisfy a layout necessity, whereas the satisfaction of a layout preference is not as important. This implies that a heavier penalty should inevitably be attached to the violation of the layout necessity constraint, rather than the layout preference constraint. Furthermore, the evaluation of a layout necessity also differs from that of a layout preference. For layout preferences, only courses with preferences that are unsatisfied are penalized, whereas for layout necessities, courses with an unsatisfied necessity and courses that wrongfully occupy a room with a special layout are penalized. This is because it is not adequate for a regular course to occupy a room with a special layout.

4.2.2 Examination Timetabling constraints

Like Course Timetabling, Examination Timetabling at HEC Montréal deals with the allocation of timeslots and rooms to examinations. As mentioned in **Section 3.1.3**, rather than courses, sub-sections of courses (i.e., pre-emptively split courses) are affected to the time-rooms. The examination timetables need to respect time constraints (timeslot preference, timeslot prohibition, similarity grouping, timeslot exclusivity grouping, one-day exclusivity grouping and two-day exclusivity grouping) and room constraints (room capacity, building preference, building grouping and layout necessity). For Examination Timetabling, the importance order of the 10 constraints will be as follows: similarity grouping, timeslot exclusivity grouping, layout necessity, timeslot prohibition, room capacity, one-day exclusivity grouping, two-day exclusivity grouping, building grouping, timeslot preference, building preference.

4.2.2.1 Timeslot preference

Timeslot preferences in Examination Timetabling allow courses to request for their examinations to be scheduled at the same time of the day and the same day of the week as their lectures. Like in Course Timetabling, each course has a maximum of 4 preferences as default, and the timeslot preferences, as well as the penalty weights, must be listed in an ascending order of importance.

The timeslot preferences in Examination Timetabling are penalized in the same way as the Course Timetabling.

4.2.2.2 Timeslot prohibition

Timeslot prohibitions are used in two instances for examinations. On one hand, courses can explicitly request their examinations to not be assigned to certain timeslots. This is the case for those that need to be in the evening or during the weekend. On another hand, prohibitions can be used to force a time interval between a course's examination and its last lecture before the examination. The default time interval is 48 hours, but courses may specifically request a different time interval or none. This type of timeslot prohibitions is created using a course's last lecture date before the examination and its time interval input, which then generates a list of timeslots that are prohibited. This ultimately requires the course timetable as an input. A penalty is affected to a course if it is assigned to a timeslot within its prohibitions.

4.2.2.3 Similarity grouping

Similarity groupings are used to ensure that examinations of the same courses happen at the same time. A mechanism automatically creates similarity groupings between the sub-sections of a course, as well as between the same courses. This type of grouping mainly helps to avoid communication between students of the same courses. Because examinations tend to be similar for all variants of a course, all language variants need to be in a similarity grouping. In parallel, they can also be manually created by the user to address different courses. A mechanism is integrated to merge groupings that share courses. The penalty for similarity groupings is affected in the same way as in Course Timetabling.

4.2.2.4 Timeslot exclusivity grouping, one-day exclusivity grouping and two-day exclusivity grouping

Several types of exclusivity groupings are employed in examinations: timeslot exclusivity groupings, one-day exclusivity groupings and two-day exclusivity groupings. The timeslot exclusivity grouping constraint is used to ensure that examinations of courses within a same grouping are assigned to different timeslots. This constraint is equivalent to the exclusivity grouping constraint in Course Timetabling. In one-day exclusivity groupings, the examinations of

the courses within a grouping must be on different days. For two-day exclusivity groupings, there needs to be at least one day without examination between each course of a grouping. Each of these types of grouping have a mechanism preventing repeated groupings, as well as groupings that are completely contained within another.

Exclusivity groupings are formed using the repertory number of the courses. By using repertory numbers rather than sub-sections, a short repertory exclusivity grouping can represent an extensive sub-section exclusivity grouping. Exclusivity groupings for examinations are mostly derived directly from the curricula, leading to a larger number of sub-sections to be involved. Furthermore, it is required for the examinations of multi-sectional courses to be scheduled at the same time, regardless of the lecture schedules. Because their sub-sections belong to a similarity grouping, they cannot be represented individually in an exclusivity grouping. For instance, suppose the courses *X* and *Y* with respectively two and three examination sub-sections, sub-section X_1 must be assigned to the same timeslot as X_2 but they both cannot be assigned to the same timeslot as Y_1, Y_2 and Y_3. While representing an exclusivity grouping with sets of sub-sections (e.g., set *X* and set *Y*) would solve this issue, the use of sub-section sets for the exclusivity constraints becomes much more complicated. Taking the previous example, the exclusivity groupings derived from these sets would be X_1 and Y_1, X_1 and Y_2, X_1 and Y_3, X_2 and Y_1, X_2 and Y_2, X_2 and Y_3. In fact, this representation would call for combinations between sub-sections of different courses through the rule of product. This principle in combinatorics stipulates that between two sets *A* and B with respectively m and n elements, there are $m \times n$ combinations. To avoid wasting computational resource, exclusivity groupings should be expressed in repertory numbers instead, since multi-sectional courses and sub-sections of a course all share the same repertory number. Figure 4.2 illustrates the difference between these two representations of exclusivity groupings through a striking example using three courses (*A, B, C*) with respectively two, three and four sub-sections. It shows that the 24 groupings of sub-sections (on the left of the figure) can be represented by a single exclusivity grouping of three repertory numbers (on the right of the figure). In terms of time complexity, the creation of groupings with sets of sub-sections through combinations is $O(n^k)$ where *n* is the length of the largest set and *k* is the number of sets. This can be entirely eliminated by creating groupings with repertories.

$$\{A_1, A_2\}, \{B_1, B_2, B_3\}, \{C_1, C_2, C_3, C_4\} =$$
$$\{(A_1, B_1, C_1), (A_1, B_1, C_2), (A_1, B_1, C_3), (A_1, B_1, C_4),$$
$$(A_1, B_2, C_1), (A_1, B_2, C_2), (A_1, B_2, C_3), (A_1, B_2, C_4),$$
$$(A_1, B_3, C_1), (A_1, B_3, C_2), (A_1, B_3, C_3), (A_1, B_3, C_4), \qquad \{A, B, C\}$$
$$(A_2, B_1, C_1), (A_2, B_1, C_2), (A_2, B_1, C_3), (A_2, B_1, C_4),$$
$$(A_2, B_2, C_1), (A_2, B_2, C_2), (A_2, B_2, C_3), (A_2, B_2, C_4),$$
$$(A_2, B_3, C_1), (A_2, B_3, C_2), (A_2, B_3, C_3), (A_2, B_3, C_4)\}$$

Figure 4.2 Difference between exclusivity groupings of sub-sections and exclusivity groupings of repertories in Examination Timetabling

Moving on to the topic of evaluation, the different types of exclusivity groupings are evaluated similarly. Regardless of the exclusivity grouping type, for a given grouping, the first step is to count the number of each repertory's sub-sections assigned to each timeslot. In timeslot exclusivity, for a given timeslot, we identify the repertory with the most sub-sections assigned, and the sub-sections that are not part of the majority repertory adds a factor of one to the penalty. In one-day exclusivity, for the timeslots in a given day, we identify the repertory with the most sub-sections assigned, and the sub-sections outside of the majority repertory are penalized by the exclusivity penalty weight. In addition, the sub-section is further penalized by the difference in time distance between its timeslot and the majority repertory's timeslot. If the sub-section is assigned to the repertory's timeslot, the penalty is at its highest, with a factor corresponding to the number of daily timeslots in the framework; and each additional timeslot difference deducts the factor by one. A penalty on the time distance is calculated by multiplying the factor with a distance weight. In two-day exclusivity, for the timeslots in a given two-day period (i.e., a given day and its next consecutive day), we identify the repertory with the most sub-sections assigned, and the sub-sections that are not part of the majority repertory are penalized in a similar manner. The time distance is penalized similarly, with the highest penalty factor corresponding to the number of timeslots within a two-day period.

4.2.2.5 Room capacity

For every student seated in examination, a seat needs to remain empty. This means that the capacity of an examination room corresponds to half of its real capacity (rounded up). In Examination Timetabling, the capacity constraint is not evaluated based on the norm of the course assigned like in Course Timetabling. Rather, the examination room's capacity needs to be higher or equal to a

sub-section's number of seats required. The penalty however takes effect similarly to the Course Timetabling constraint.

4.2.2.6 Building preference and building grouping

While the building preferences are used and evaluated in the same way as Course Timetabling, building groupings play a more important role than the preferences. This is because, although examinations are not supervised by lecturers, some require their assistance. The building preferences allow to make sure that the examinations of the same courses are within a same building such that the lecturer can be available when needed. The penalty for building groupings is evaluated in the same way as the similarity groupings, where buildings are compared rather than timeslots.

4.2.2.7 Layout necessity

The layout required relates to the form of the examination. As a result, the layout preferences have been removed and only the layout necessities remain in Examination Timetabling. This is mainly because layout preferences are related to lecturers' preferences, which are irrelevant in this context. Moreover, it is important to note that the course lectures' layout necessities do not directly translate into the examinations' layout necessities. To illustrate, a course with lectures given in a computer lab may have an examination in paper format, hence it does not require computer lab as a layout. The layout necessity constraint in Examination Timetabling is evaluated like in Course Timetabling.

4.3 Data representation

We now turn to structuring the data. In doing so, we first draw a distinction on data mutability, by making use of tuples (for immutable data) and lists (for mutable data). All tuples and lists contain the same number of elements, with a length corresponding to the total number of time-rooms in the framework. We treat time-room characteristics as immutable (contained in tuples) and course requirements/preferences as mutable (contained in lists). We start by defining a *time-room tuple*, where each element corresponds to a timeslot for a given room (room r at timeslot t). We then define the *characteristic tuples* to store relevant room-related characteristics (e.g., room capacity, layout, and building location). The state is represented by a list, and this *state list* shows the

assignment of courses when compared to the *time-room tuple*. Every room requirement/preference (e.g., norm, layout required, and building required) and timeslot requirement/preference (e.g., timeslot preferences, and similarity groupings) is represented in a list, referred to as *requirement lists*. The representation in tuples and lists allows for easy comparison between characteristics of time-rooms and requirements of courses. Room requirement lists are compared to their corresponding *characteristic tuples* (e.g., norm list compared to room capacity tuple). Timeslot preferences and prohibitions lists can be directly compared to the *time-room tuple*. However, similarity grouping and exclusivity grouping lists call for timeslot comparison within groupings, which potentially leads to multiple comparisons to the *time-room tuple* for a course. Therefore, the evaluation of grouping constraints takes slightly longer to complete. These *requirement lists* are manipulated to follow each course's requirements/preferences through the state changes expressed by the *state list*. This structure allows us to more easily retrieve data of time-rooms and courses affected by each iteration.

Initially, bi-dimensional numPy arrays were used to represent timetables. In such arrays, the first dimension corresponds to the timeslots and the second dimension to the rooms. For instance, the course timetable is represented by a 20×67 array. However, this representation is highly taxing in computational time. This is mainly because only the assignment of courses to time-rooms is represented by the array. The time-room information (e.g., room capacity and room layout) and course information (e.g., course time preferences and similarity groupings) are represented in a different and separate manner than the assignment. Retrieving information in this circumstance is complicated, which ultimately lengthens the calculation of the effects of a state change. While each swap generally involves a very small number of time-rooms (e.g., courses of 2 time-rooms swapped over 1,340 time-rooms), the entirety of a timetable must be evaluated at every state change in this array representation. The complete evaluation of solution leads to a lot of repeated calculations. With the list representation suggested, only the affected courses and time-rooms are evaluated through partial evaluation. This is enabled by the fact that time-room information and course information are represented in the same way as the timetable, allowing direct access through the index. The swap generation returns indexes to be swapped in the *state list*. All *requirement lists* follow the same changes as the *state list*, and the constraints are evaluated at the level of the affected indexes. By avoiding the evaluation of the entire state at every iteration through this

representation of data, we realize a significant economy in evaluation time. Figure 4.3 shows the evaluation plot of a Course Timetabling model using partial evaluation and complete evaluation. This figure allows to compare the algorithm's convergence time using partial evaluation with list representation (illustrated in grey), and complete evaluation with array representation (illustrated in black). We note that the model converges considerably faster when using partial evaluation. The time economy through this choice of evaluation is significant; it reduces the solution evaluation time by a factor of over 1000.

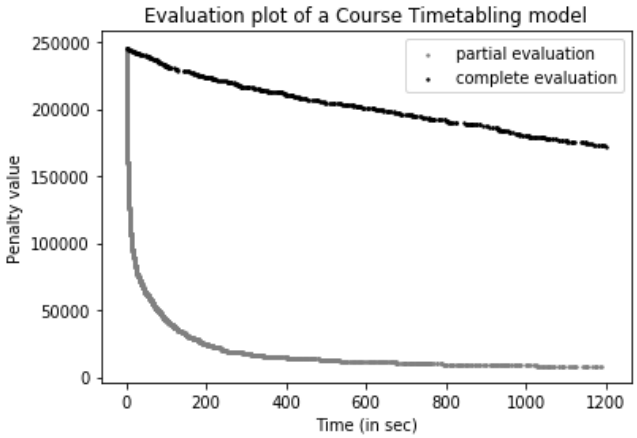

Figure 4.3 Comparison between partial evaluation and complete evaluation with a Course Timetabling problem

4.4 Initialization

The algorithm is initialized with a state defined by the user. This initial state must be a complete and feasible solution. To realize this, three types of initializations are suggested: *last*, *random*, and *random by similarity groupings*. *Last* invokes the use of the previous year's semestrial timetable as the initial state. Because the examination framework differs year-to-year, the *last* initialization is unapplicable to Examination Timetabling. *Random* and *random by similarity groupings* are constructive heuristics; they start off empty and iteratively assign a course to an available time-room until a complete timetable is built as a state. *Random* returns a list of random non-repetitive time-rooms, and they are assigned to the alphabetically sorted courses. *Random by similarity groupings* is done in two phases. First, we assign time-rooms to courses that are part of similarity groupings, in which all courses of a grouping should share a same timeslot. For a grouping, we randomly generate one timeslot with enough available rooms for the number of courses in the grouping. A list of random non-repetitive rooms is then generated. The time-rooms formed from the timeslot and rooms generation must be yet assigned. This step is repeated for all similarity

groupings. Second, we assign time-rooms to courses that are not in any similarity groupings. To do so, a list of random non-repetitive time-rooms which have yet been assigned is generated. The idea behind *random by similarity groupings* initialization is for the algorithm to start off with a timetable that already has the similarity groupings constraint satisfied, that way less time is spent to reunite courses within groupings. The initial *state list* is derived from the list of courses assigned to time-rooms resulting from the type of initialization chosen by the user. The *requirement lists* are, in turn, derived from the *state list.* Because the *state list* and the *requirement lists* must maintain their length, time-rooms with no course assigned are assigned the *None* value at their respective position in the *state list* and the *requirement lists*.

4.5 Transformation

Transformations are modifiers that are applied on a state to create a new one. We only consider swaps for transformations in our algorithm. In this context, we define a swap as an exchange of courses between two or more time-rooms. A time-room within a swap can be empty (no course is currently assigned to the time-room); it would result in a translation of course from a time-room to another. To maintain similarity groupings that were united through the *random by similarity groupings* initialization, the algorithm must support swaps of sets of time-rooms. A swap of time-room sets can be viewed as an exchange of courses between rooms of different timeslots. We will clarify on the idea of swaps of time-room sets when discussing swap generation.

4.6 Swap generation

The following section will explain the two steps entailed in a swap generation: the generation of a list of time-room indexes for each set involved, as well as the generation of a swapping order.

In the generation of a list of time-room indexes for each set of a swap, each set must be generated subsequently to avoid repetition. To do so, we first generate an index number amongst the *time-room tuple*. If the course currently assigned to the time-room (indicated by the *state list*) is part of a similarity grouping (indicated in the *similarity grouping list*), the time-rooms currently assigned to the courses of the grouping become part of the set, if they currently share the same timeslot as the first generated time-room. However, the number of time-rooms in each set must be equal, and this number is determined by the length of the largest set. Additional time-rooms are generated for

each set with insufficient time-rooms. The additional time-rooms, along with the time-rooms of the courses in their similarity groupings, need to be currently assigned to the same timeslot as the set they join, and they cannot exceed the number of time-rooms to fulfill. Figures 4.4 present a flowchart to illustrate this generation process, whereas Figure 4.5 provides a flowchart of the subprocess to generate additional time-rooms for sets with insufficient time-rooms.

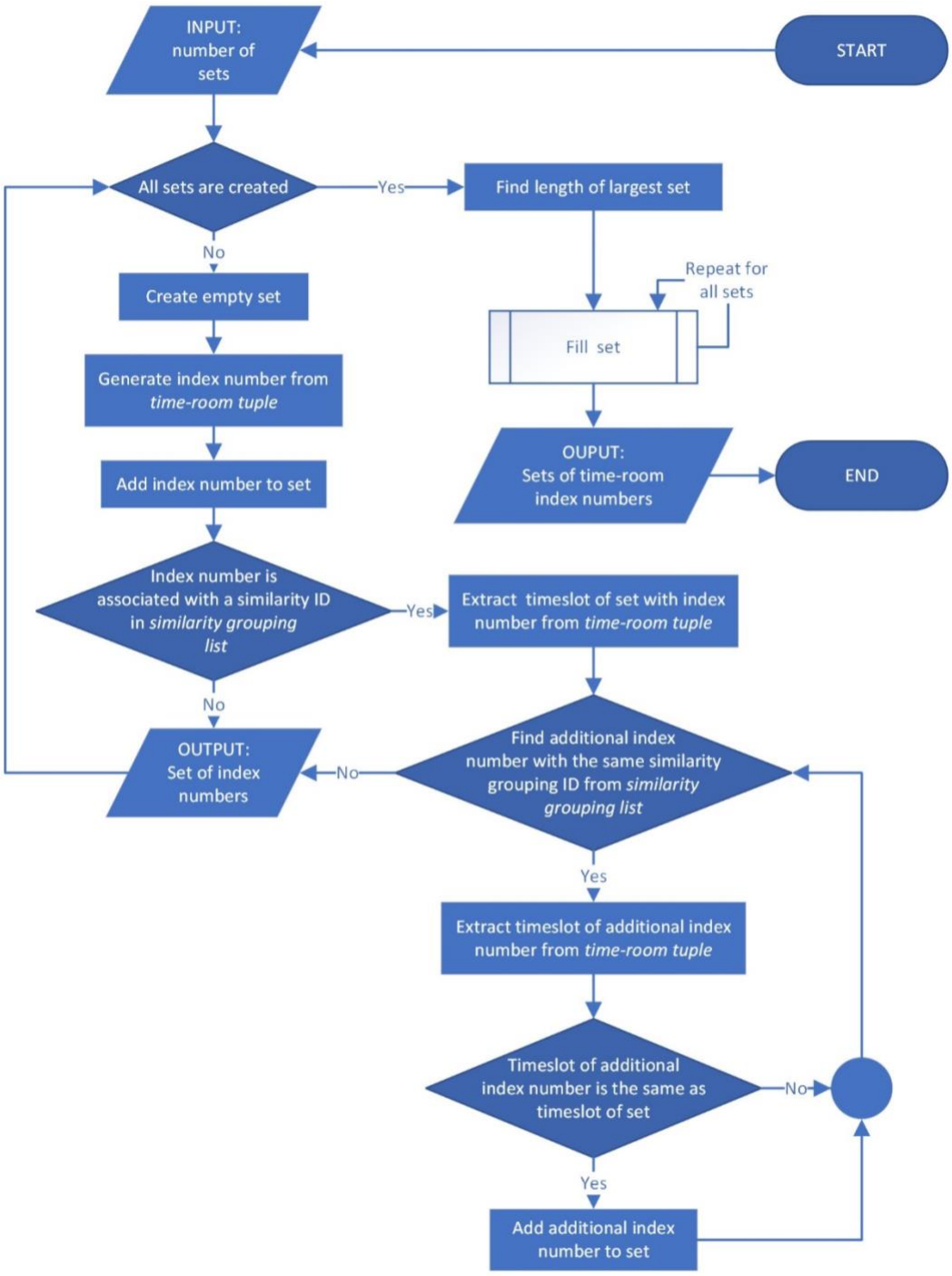

Figure 4.4 Flowchart of the process of swap generation

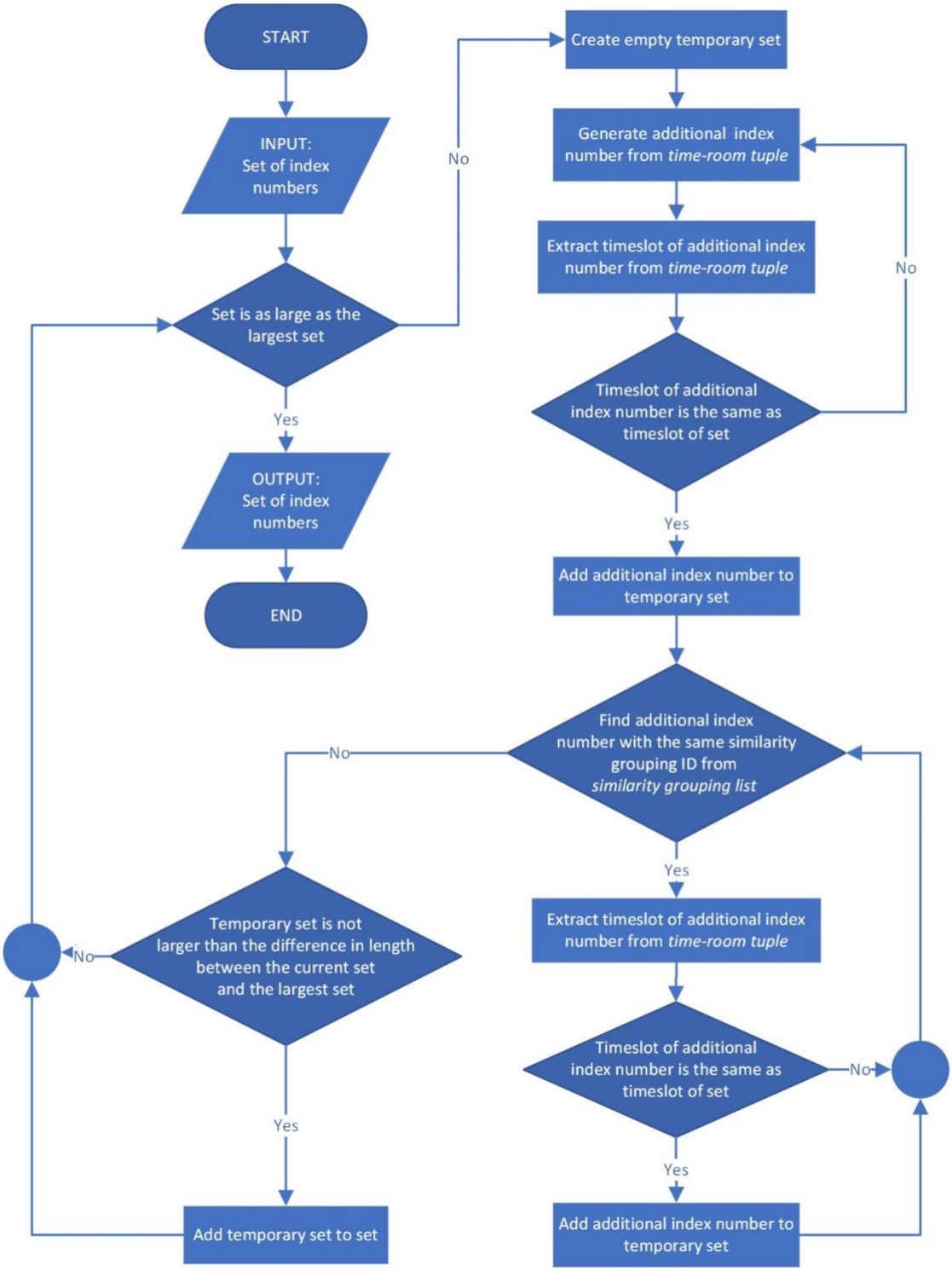

Figure 4.5 Flowchart of the subprocess to fill sets

The generation of a swapping order is needed to enable the possibility to swap multiple (more than two) sets of time-rooms. This order defines the dynamic between the sets, i.e., with which set each is swapped. In such a way, a swap can be considered as a permutation between the time-room sets. For n time-room sets, there are $n! - 1$ possible permutations (without repetitions), excluding the permutation that involves no change. The number of possible permutations becomes increasingly high as the number of time-room sets increases. The time complexity for generating a swapping order from the possible permutations is $O(n!)$, which suggests that the number of sets should be limited to avoid lengthy swap generations. We compare two ways of generating a swapping order: best order and random order. The best order is obtained by evaluating every possible permutation for a given list of time-room indexes and selecting the best resulting one; the random order only calls for a single evaluation of a randomly generated order amongst the possible permutations. We compare the best order and random order methods with two Course Timetabling models, one with 3-set swaps and another with 6-set swaps. Through evaluation plots presented in Figure 4.6, it is evident that the random order method both performs better and converges faster than the best order method in both models. Incidentally, Figure 4.5 shows that a higher number of sets does not guarantee better results. More details on the performance of models with multiple sets will be provided in **Chapter 5.**

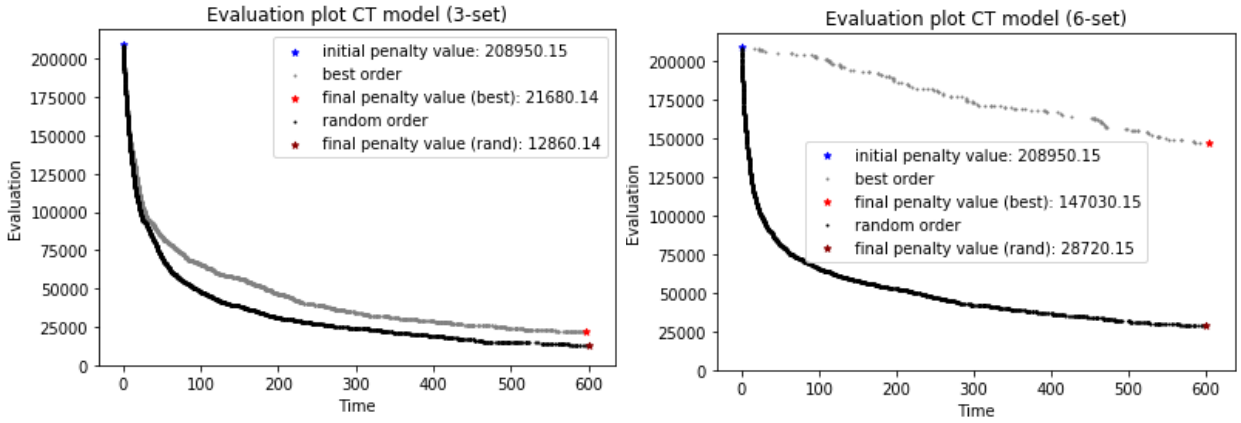

Figure 4.6 Evaluation plot of a 3-set model and 6-set Course Timetabling model using best order and random order methods

We further investigate the difference between the two methods by comparing the number of swaps generated and the average evaluation time in models that use different number of sets, as shown in Table 4.1. Globally, as the number of sets increases, less swaps are generated, and evaluation appears slower. The longer generation and evaluation processes can be explained by the increase

in time-rooms involved through the increase in number of sets. Moreover, as the number of sets grows higher, the difference between the two methods becomes more prominent. While the best order method seems better for the model with 2 sets, the random order method works better for the other models. In the model with 2 sets, the best order method allows for 1.10x more swaps generated and 1.09x faster average evaluation time than the random order method. In models with 3 to 6 sets, we see that the random order method leads to significantly higher number of swaps generated (i.e., respectively 4.16x, 18.5x, 96.37x and 577.81x) and faster average evaluation time (i.e., respectively 5.56x, 24.21x, 121.43x and 718.37x) than the best order method. We attribute this disparity in number to the fact that the best order method requires $n! - 1$ evaluations, making it longer to generate an order when the number of sets is high, whereas the random order method only requires one evaluation regardless of the number of sets. While both methods still require the generation of $n!$ permutations, the difference in execution time resides in the number of evaluations involved in the generation of a swap.

	Number of swaps generated		**Average evaluation time (in seconds)**	
Number of sets	***Best order***	***Random order***	***Best order***	***Random order***
2	346558	313711	0.0011	0.0012
3	55349	230505	0.0100	0.0018
4	10143	188105	0.0581	0.0024
5	1407	135586	0.4250	0.0035
6	182	105161	3.3045	0.0046

Table 4.1 Comparison of best order and random order methods through number of swaps generated and average evaluation time according to number of sets

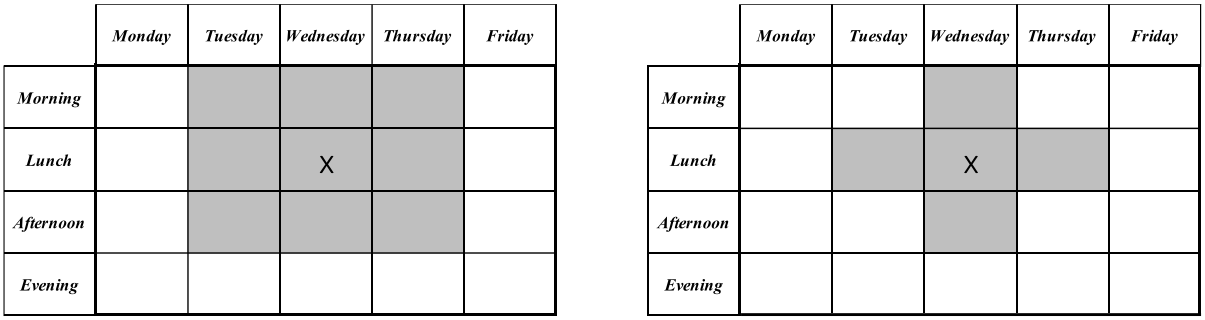

Figure 4.7 Possible movements in two special swapping techniques (box(2,2) and cross(1,1)) through a Wednesday lunch timeslot example

To generate swaps, a fully random mechanism was initially adopted, implying that each part of a swap is generated randomly in a subsequent manner. However, full randomness can potentially lead to slow convergence, especially since combinatorial problems tend to have very large search

spaces. This indicates a need for additional mechanisms to randomness. We notice that most accepted swaps have a pattern: they involve adjacent timeslots. The cause of this is the existing patterns within timeslot preferences; the preferred timeslots of a course are generally very close to one another. For instance, it is significantly more likely for a course lecture with a Monday morning preference to also list a Monday noon (or even a Tuesday morning) preference rather than a Friday evening. Therefore, it would be beneficial to exploit this idea of time distance within a swap in attempt to reduce exploration time.

To control the timeslot distance in a pairwise swap, it is necessary to generate each part of a swap sequentially. This allows to add a bound on the timeslot of the first part of the swap to then generate the timeslot of the counterpart. Several techniques could be developed to generate swaps of time-rooms that are closer in terms of time; we decide to focus on $box(v, h)$ and $cross(v, h)$ swappings. $Box(v, h)$ swapping enables simultaneously bi-dimensional swaps (vertical, horizontal, or across), while $cross(v, h)$ swapping only allows for uni-dimensional swaps (vertical or horizontal). The former allows for courses to simultaneously change timeslot and day, whereas the latter only permits a change of timeslot or day at a time. $Box(v, h)$ swapping adds a bound by completely surrounding a given timeslot, to create a box around the timeslot, and $cross(v, h)$ swapping adds a cross-shaped bound around the given timeslot. The user-defined values of v and h respectively correspond to the vertical units and the horizontal units. Figure 4.7 illustrates the different timeslots enabled (boxes colored in grey) between the two swapping techniques through an example: given a course time-room at Wednesday lunch (indicated by the "x" symbol), a $box(1, 1)$ would allow a time-room to be generated from nine timeslots (between Tuesday to Thursday, from morning to afternoon) as seen on the left of Figure 4.7, and a $cross(1, 1)$ would enable 5 possible timeslots (Tuesday lunch, Wednesday morning to afternoon, and Thursday lunch) as seen on the right of Figure 4.7.

4.7 Types of swaps

In attempt to further the study on swaps, we have conducted an analysis on the types of swaps that are accepted in a baseline model. We categorized the swaps into 3 types: time-room swaps, time swaps, and room swaps. Stacked bar plots and pie charts are used to observe the occurrence of the different types of swaps according to the runtime of the model. Figure 5.2 shows stacked bar plots

of the frequency of the accepted swap types according to the runtime. The frequency is illustrated in four plots, each representing a different interval of runtime, because of the heavy concentration of swaps at the start of the runtime. Figure 4.8 shows that 78.25% of the total swaps (2988 swaps within 600 seconds) accepted in the first 150 seconds. Besides, we note that almost all of the time-room swaps (90.25% out of 667 time-room swaps) and most of the swaps involving only time (77.25% out of 923 time swaps) or room (73.18% out of 1398 room swaps) happen in the first 150 seconds. On another note, the time-room swaps are noticeably less common; they only account for 22.32% of the total swaps. Time swaps and room swaps respectively account for 30.89% and 46.79%. The pie charts in Figure 4.9 allow us to extract more information on the proportions of the types of swaps accepted per runtime interval. It can be observed that, aside from the first 150 seconds where the proportions of time-room, time and room swaps are almost equal, the room swaps are the most frequent within a given time interval, followed by the time swaps and the time-room swaps. The exclusive use of specific types of swaps between certain time intervals in a model could potentially lead to interesting results.

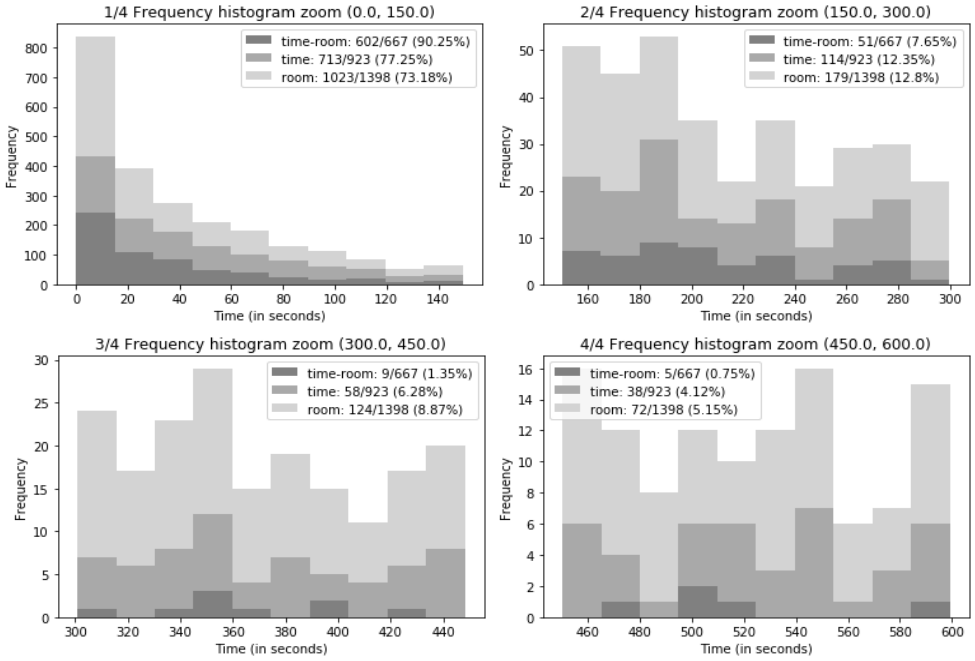

Figure 4.8 Stacked bar plots of swap types per time interval

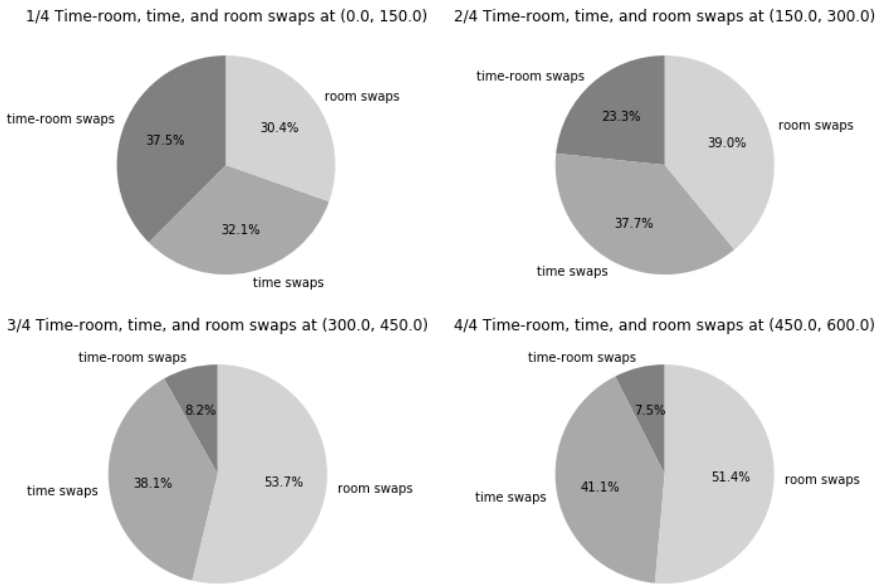

Figure 4.9 Pie charts of swap types per time interval

4.8 Solution evaluation

In basic terms, evaluation compares the current state to the temporary state (i.e., the state after transformation). Due to the data representation proposed, it is not necessary to evaluate the entire timetable. Rather, only the time-rooms affected by the transformation need to be evaluated, once before and once after the transformation. Furthermore, because the constraints are on two distinct dimensions (time and space), evaluation can be done on two levels, and can be calculated using two variables: var_time and var_room. The sum of both results in the total variation in evaluation caused by the transformation generated on the current state. This decomposition in evaluation allows for an economy in evaluation time for some scenarios.

Since Course Timetabling and Examination Timetabling are such large combinatorial problems, they are bound to involve numerous evaluations. To spare time in evaluation and focus on exploring solutions, it is important to identify situations where evaluation is not necessary. The distinction between time and space in the timetabling problems fuels the idea of computing the variation of penalty through two separate variables: var_time and var_room. Here are three scenarios where the time-rooms generated for the transformation allow for a significant economy in evaluation time:

1) The time-rooms are empty.

   If the time-rooms generated are empty, there would be no change: $var\_time = 0$ and $var\_room = 0$. Therefore, no evaluation is necessary.

2) The time-rooms have the same timeslot.

   If the transformation is only a change of room for the same timeslot, only the room constraints would be affected: $var\_time = 0$. Therefore, no evaluation for timeslot is necessary, and if the variation for room shows improvement, the transformation is accepted.

3) The time-rooms have the same room.

   If the transformation is only a change of timeslot for the same room, only the timeslot constraints would be affected: $var\_room = 0$. Therefore, no evaluation for room is necessary, and if the variation for timeslot shows improvement, the transformation is accepted.

4.9 Stopping criteria

Because the optimal solution is not known, and the problem instance might not have a single optimal solution, the algorithm needs a stopping criterion. Two options based on duration are made available: total runtime and plateau time. Both options can also be used simultaneously such that the algorithm can be forcibly stopped after a predefined duration or after it hits a plateau (i.e., a period when no significant change can be found) for a predefined duration. The use of both options allows the algorithm to terminate as soon as it converges, such that the user does not have to wait the remainder of the algorithm duration to extract a solution.

5 Results

5.1 Data simulation

After acquiring knowledge on the general mechanics of the algorithm, we move on to the model applications to solve HEC Montréal's timetabling problem instances. As novel problems, Course Timetabling and Examination Timetabling at HEC Montréal do not have benchmarks and require data simulations for model testing. Most data required for the algorithm (e.g., lecturers' timeslot preferences and layout requirements) were not accessible at the time of the algorithm conception, leading to a need for data simulation. To test for a given past semester, we derive from its semestrial timetable to simulate the data. The past timetable will mainly be used to derive data, rather than to assess as a benchmark. This is because the goal of the algorithm is to generate conflict-free timetables that mostly satisfy students and lecturers' preferences within a reasonable timeframe. Rather than attempting to replicate the past semestrial timetable, we focus on the analysis of the quality aspect of the timetables generated by the algorithm.

To test Course Timetabling models, we use data from the lecture timetable of Fall 2019. While courses that could require layout necessities are directly extracted from the timetable, the preferences on the layout, building and timeslot are only partially derived. This is because it is impossible to pinpoint the courses with these requests solely from the timetable. The norms were not provided, so they had to be simulated. The similarity groupings and exclusivity groupings are formed by comparing timeslots of the uni-sectional courses within a same curriculum. We assume that, for a given curriculum, courses with the same timeslot form a similarity grouping, and courses with different timeslots form an exclusivity grouping. In the resulting simulated data, only 2% of the courses do not have any preferences (other than the norm that is affected to all 802 courses); 51% of the courses require a specific layout, including 4% that indicate a layout necessity; 49% of courses request a building preference, and 93% of them provide at least one timeslot preference; finally, 8% of the courses are part of a similarity grouping, while 18% are in an exclusivity grouping.

To test Examination Timetabling models, we similarly derive data from the examination timetable of Fall 2019. The number of seats required for each sub-section is provided. Because room layouts

concerned in the examination context are special layout (only the computer labs and the trading room), courses requiring a specific layout are directly extracted from the timetable. The preferences of building and timeslot, as well as the prohibitions of timeslot, are only partially derived from the timetable since the courses with such requests cannot completely be identified from the timetable. For timeslot prohibitions, we derive the timeslots that need to be included to respect the time interval from the last lecture date before examination. The academic programs' requirements of course examinations to be scheduled, or not scheduled, at a specific timeslot are also used to create timeslot preferences and prohibitions. The similarity groupings and exclusivity groupings are formed by comparing timeslots of the sub-sections within a same curriculum. We assume that the sub-sections with the same repertory number form a similarity grouping. The building groupings are simply the same as the similarity groupings. Concerning the three types of exclusivity groupings, for a given curriculum, repertories of sub-sections with different timeslots form a timeslot exclusivity grouping, repertories of sub-sections on different days form a one-day exclusivity grouping, and those separated by a day without examination form a two-day exclusivity grouping. The resulting simulated data show only 1% of sub-sections without any preferences (other than the number of seats required for all 995 sub-sections); less than 1% of the sub-sections require a specific layout or request a building preferences while 91% are part of a building grouping; 8% have a timeslot preference and 62% use timeslot prohibitions; finally, 91% of the sub-sections are part of a similarity grouping, whereas 9% are in a timeslot exclusivity grouping, one-day exclusivity grouping and two-day exclusivity grouping.

5.2 Experimental results

This section is dedicated towards the presentation of the different models developed to solve the Course Timetabling and Examination Timetabling problems of HEC Montréal. The models are tested using the simulated data and, to ensure comparability, all models use the same constraint weights (excl. the room occupancy constraint) which are listed in Tables 5.1 and 5.2. The discussion of the results based on a total of 10 runs will be supported by additional tables. Description tables, labelled as 5A, are used to summarize each model by presenting the parameter inputs and a short description of the mechanics involved in the model. The parameters revealed through this table include the type of initialization, the type of swapping, the use of sets of single or multiple time-rooms, the number of sets, and the weight value of the room occupancy constraint.

Result tables, labelled as 5B, report the average, lowest and highest values for initial and final penalty, as well as the average runtime needed to reach 95% and 99% of the improvement accomplished within the defined total runtime. A result table provides insight regarding the performance and convergence of a model, given the inputs presented in its corresponding description table. The result reported in this section specifically concern the Course Timetabling problem, and the conclusions derived from these results are consistent with the Examination Timetabling problem, unless specified otherwise.

Constraint	*Weight*
Timeslot preference	[10, 20 30, 40]
Similarity grouping	200
Exclusivity grouping	500
Capacity	300
Layout necessity	450
Layout preference	35
Building	29

Table 5.1 Course lecture constraint weights

Constraint	*Weight*
Timeslot preference	[5, 10, 20, 60]
Timeslot prohibition	625
Similarity grouping	900
Timeslot exclusivity grouping	800
One-day exclusivity grouping	200 (distance = 1)
Two-day exclusivity grouping	100 (distance = 1)
Capacity	500
Layout necessity	700
Building	30
Building grouping	100

Table 5.2 Examination constraint weights

Considering the different model components, we introduce the concept of phase change, where a model changes one (or multiple) of its components at a certain point of the runtime. Two different approaches can be used for activating a phase: time-based and exploration-based. In the time-based approach, a phase change is triggered at a given runtime. This information is usually defined by a fraction of the total runtime. In the exploration-based approach, the phase change is instead triggered by a certain exploring time (i.e., the duration to find an improving swap). We find that two conditions must be in place to ensure proper functioning of the latter approach: ongoing exploring time and last exploring time. Since the average exploring time depends on the problem

instance, rather than using a fixed number, the ongoing exploring time should be expressed according to the last exploring time. Because the ongoing exploring time is expressed by a factor, the last exploring time needs to be limited by an explicit lower bound. The default lower bound for the last exploring time is 0.01. This is to ensure that very small values of the last exploring time do not affect the condition on the ongoing exploring time. Due to the conditions within the exploration-based approach, a phase change is not guaranteed to happen.

5.2.1 Baseline model

To start off, Course Timetabling and Examination Timetabling at HEC Montréal are novel problems. Because there is no available benchmark, a baseline model is used to compare improvement strategies. The baseline model corresponds to the simplest model developed, given the components presented in **Chapter 4**. As summarised in the description table, Table 5A.1, this baseline model is randomly initialized through *random* initialization, and it randomly generates swaps of single time-rooms, rather than sets of multiple time-rooms, and each swap involves two time-rooms. Additionally, it uses the room occupancy constraint with its default weight value of 10^{-3}. Through the result table, Table 5B.1, we can see that, with a total runtime of 600 seconds, the average penalty value progresses from 214252 to 6001, with 95% of the improvement achieved within the first 118.36 seconds, and 99% within 339.78 seconds in average. The range of penalty values greatly shrinks, with an initial value range of 9860 to a final value range of 825.

5.2.2 Adapted baseline model

We adapt the baseline model to study the joint effectiveness of the *random by similarity grouping* initialization and the use of sets of multiple time-rooms. Table 5A.1 indicates that all inputs, aside from the type of initialization and the use of sets of single or multiple time-rooms, remain unchanged from the baseline model. By pre-emptively regrouping courses within their similarity groupings and allowing them to change time-rooms simultaneously, we believe that an opportunity for a significant time economy can be created. As shown in Table 5B.1, this type of initialization allows the model to start at a lower initial penalty value than the baseline model, with an average value of 212100. Within a similar total runtime, the adapted baseline model successfully reaches a lower final penalty value of 5351 in average. On another note, 95% of the average improvement is accomplished within 127.53 seconds, and 99% of the average improvement within 370.14

seconds out of 600 seconds. The signs of slower convergence can be explained by the better performance of the model. With a range of initial values of 13880 and a range of final values of 1840, the adapted baseline model seems to be much more variable than the baseline model, which shows ranges of 208515 and 825 for its initial and final values. While the range in final penalty values is much larger in the adapted baseline model, the highest penalty value obtained is still lower than the baseline model's, showcasing once again the performance of the adapted baseline model.

Name	***Baseline***	***Adapted baseline***
Initialization	Random	Random by similarity
Type of swapping	Random	Random
Single or multiple	Single	Multiple
Number of sets	2	2
Occupancy	10^{-3}	10^{-3}
Algorithmic details	Simplest model	Baseline model that uses *random by similarity groupings* initialization and sets of multiple time-rooms

Table 5A.1 Description of baseline and adapted baseline models

Name	***Baseline***	***Adapted baseline***
Total runtime	600 sec	600 sec
Average initial value	214252	212100
Highest initial value	218375	218900
Lowest initial value	208515	205020
Range of initial values	9860	13880
Average final value	6001	5351
Highest final value	6515	6440
Lowest final value	5690	4600
Range of final values	825	1840
Average time for 95% of final value	118.36 sec	127.53 sec
Average time for 99% of final value	339.78 sec	370.14 sec

Table 5B.1 Results of baseline and adapted baseline models

5.2.3 Baseline and adapted baseline models with longer runtime

Table 5C.1 presents the results of a single execution of the baseline and adapted baseline models executed for a longer runtime. It lists the total runtime, initial and final penalty values, as well as the time for 95% and 99% of the final value of the single execution of the model. We observe that,

with a total runtime of 1800 seconds, the baseline model does improve. The average final penalty of the baseline model after the extended runtime is 5130, showing a 14.51% decrease in value from the average of the original runtime of 600 seconds. We see that 95% of the improvement is achieved in 157.74 seconds, and 99% of it is achieved in 540.01 seconds. By contrast, there is a noticeably bigger decrease in final penalty value for the adapted baseline model through the extended runtime. The final value reaches 3415, which corresponds to a decrease of 36.18% in value from the average of the original runtime. This large improvement does, however, show an effect on the convergence of the model. In fact, when the model is executed for 1800 seconds, the penalty reaches 95% of its final value at 203.66 seconds and 99% of it at 668.18 seconds.

Name	***Baseline***	***Adapted baseline***
Total runtime	1800 sec	1800 sec
Initial value	218375	208950
Final value	5130	3415
Time for 95% of final value	157.74 sec	203.66 sec
Time for 99% of final value	540.01 sec	668.18 sec

Table 5C.1 Results of baseline adapted baseline with extended runtime

By comparing both models, we can conclude that a longer runtime is as not beneficial to the baseline model, seeing that 99% of its final penalty value can be achieved within the original runtime of 600 seconds. In contrast, the adapted model needs more than 600 seconds to achieve 99% of its final penalty value. With a 36.18% improvement to the former average penalty value, we can argue that additional runtime can be advantageous to the adapted baseline model. We can clearly see that the extended runtime enables penalty to reach lower values than the lowest penalty values of the runs with 600 seconds. Considering that the penalty value can be improved, we begin exploring avenues of improvement strategies as alternatives to a longer runtime. From Table 5B.2, we can see that the adapted baseline model has a lower average initial value, but a much larger range than the baseline model, i.e., average initial values of 214252 and 212100 with ranges of 9860 and 13880. However, considering the lower resulting penalty value with a runtime of 1800 seconds and the lower average penalty value with a runtime of 600 seconds, the next models will mostly continue to use the parameters *random by similarity* initialization and sets of multiple time-rooms, introduced through the adapted baseline model.

5.2.4 Swaps of multiple time-room sets

Next, we turn to swaps between multiple sets of time-rooms, keeping in mind that the number of sets used should be limited, as discussed in **Section 4.1.5**. Table 5A.2 presents three models that operate with more than 2 sets of time-rooms in their swaps. As we can see, only the number of sets differs from the parameters of the adapted baseline model. With 3 sets of time-rooms involved in each swap, Table 5B.2 indicates that the model reaches an average final penalty of 7706, with 95% of improvement found within 195.50 seconds and 99% of it found within 418.05 seconds in average. This model shows both worse performance and convergence than the baseline model. We then use the time-based approach to equally allocate runtime to a phase of 2 sets and another of 3 sets of time-room. With the use of 3 sets as the starting phase, Table 5B.2 reports an average final penalty value of 5765, whereas with the use of 2 sets as the starting phase, it reports an average value of 5856. As the models do not seem to show good performance compared to the adapted baseline model, the following models will simply continue to use swaps of 2 sets.

Name	***3set***	***3set_2set_1/2***	***2set_3set_1/2***
Initialization	Random by similarity	Random by similarity	Random by similarity
Type of swapping	Random	Random	Random
Single or multiple	Multiple	Multiple	Multiple
Number of sets	3	Per phase (3 and 2)	Per phase (2 and 3)
Occupancy	10^{-3}	10^{-3}	10^{-3}
Algorithmic details	Allows swaps of 3 sets of multiple time-rooms	Allows swaps of 3 sets in first ½ of total runtime; and allows only swaps of 2 sets in remaining runtime	Allows only swaps of 2 sets in first ½ of total runtime; and allows swaps of 3 sets in remaining runtime

Table 5A.2 Description of models with swaps of multiple time-room sets

Name	***3set***	***3set_2set_1/2***	***2set_3set_1/2***
Total runtime	600 sec	600 sec	600 sec
Average initial value	211748	208137	209630
Highest initial value	217995	213910	217020
Lowest initial value	204485	202900	203100
Range of initial values	13510	11010	13920
Average final value	7706	5765	5856
Highest final value	8645	6170	6740
Lowest final value	7015	5395	5500
Range of final values	1630	775	1240
Average time for 95% of final value	195.50 sec	220.48 sec	119.22 sec
Average time for 99% of final value	418.05 sec	420.72 sec	288.97 sec

Table 5B.2 Result of models with swaps of multiple time-rooms sets

5.2.5 Special swappings

As mentioned in **Section 4.1.5**, the randomness proposed in the generation of swaps can potentially lead to slow convergence. With the $box(v, h)$ and $cross(v, h)$ swappings introduced previously, we aim to improve swap generation by looking for patterns in the accepted swaps of a model. We suggest an analysis, with the heatmap shown in Figure 5.1, of the accepted swaps in the baseline model discussed in the **Section 5.2.1**. In heatmap, the vertical axis shows the vertical units within the swaps and the horizontal axis shows the horizontal units within the swaps. We can see that the figure shows a strong concentration in the upper left corner. This indicates that the majority of the accepted swaps (70% of the 1089 swaps) involve a short time distance, which is limited to 2 units of time in days and daily timeslots. The heatmap seems to suggest that special swappings such as $box(2, 2)$ and $cross(1, 1)$ may be helpful in reducing the generation of non-improving swaps.

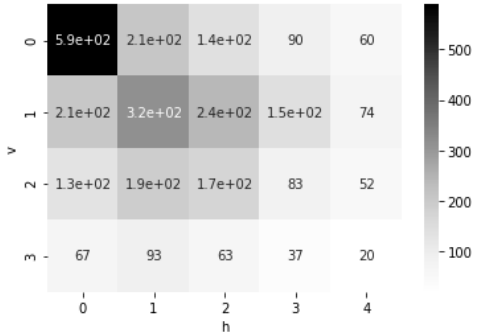

Figure 5.1 Heatmap of time distance in accepted swaps in a baseline model

To integrate this idea, we first determine the impact of special swappings on the adapted baseline model. Two models are presented through Table 5A.3, where each model only uses a single type of special swapping which are $box(2,2)$ and $cross(1,1)$. The remaining of the parameters are unchanged from the adapted baseline model. As reported in Table 5B.3, when using $box(2,2)$, the average final penalty value achieved is 8262, and when using $cross(1,1)$, the average value achieved is 32590. The model with $box(2,2)$ reaches 95% of its improvement in 143.97 seconds and 99% of its average improvement in 370.01 seconds in average. The model with $cross(1,1)$, considering a worse performance, reaches 95% of its average improvement in 79.64 seconds and 99% of in average improvement at 253.75 seconds in average. These models do not yield comparable results to the baseline and adapted baseline models.

Name	$\boldsymbol{Box(2,2)}$	$\boldsymbol{Cross(1,1)}$
Initialization	Random by similarity	Random by similarity
Type of swapping	$box(2,2)$	$cross(1,1)$
Single or multiple	Multiple	Multiple
Number of sets	2	2
Occupancy	10^{-3}	10^{-3}
Algorithmic details	Only uses special swapping $box(2,2)$	Only uses special swapping $cross(1,1)$

Table 5A.3 Description of models that use a single type of special swapping

Name	$\boldsymbol{Box(2,2)}$	$\boldsymbol{Cross(1,1)}$
Total runtime	600 sec	600 sec
Average initial value	211462	210470
Highest initial value	215980	217310
Lowest initial value	208575	203980
Range of initial values	7405	13330
Average final value	8262	32590
Highest final value	9225	34960
Lowest final value	7335	30780
Range of final values	1890	4180
Average time for 95% of final value	143.97 sec	79.64 sec
Average time for 99% of final value	370.01 sec	253.75 sec

Table 5B.3 Results of models that use a single type of special swapping

From the last observations, there is evidence to believe that the exclusive use of special swappings does not improve the algorithm. As random swapping seems to promote better performance and convergence, we move on to develop models that integrate the use of random swapping and special swappings through model phases. Given a list of swapping types ordered by their phase occurrence, two different approaches can be used for phase activation: time-based and exploration-based. A major difference can be perceived between the two approaches: as long as there is runtime allocated to each swapping type phase, all listed swapping types are used in the time-based approach, which is not the necessarily the case in the exploration-based approach. Because it is not possible to predict how many times the conditions for a phase change are satisfied in the exploration-based approach, there is no guarantee that every listed swapping type will be used in this approach.

We first take a look at the time-based approach through the models described in Table 5A.4, which use random swapping and the special swappings $box(2,2)$ and $cross(1,1)$. These models only differ from the adapted baseline model in terms of types of swapping. This particular order in the swapping types promotes a gradual increase of precision in time distance. As shown in the table, we test several runtime allocations to these swapping type phases. For a model that equally allocates the runtime to each type of swapping (⅓ of total runtime to each), the final penalty value is 6187 in average. For a model that allocates ½ of the runtime to random swapping and ¼ to each of the special type of swapping, the final penalty value averages 5234. Finally, for a model that allocates ⅔ of the runtime to random swapping and ⅙ for each of special type of swapping, the final penalty value is 5227 in average. From these models developed, we observe that the last two model beats not only the performance of the baseline model, but also the performance of the adapted baseline model. In addition to the lower average value of the final penalty, a smaller range of final value is also noted. The first model shows a range of 3245, as opposed to the last two models which respectively have a range of 1000 and 800. These results seem to indicate that a model performs better and is less variable when the majority of the total runtime is dedicated towards random swapping.

Name	***Rand_box*(2, 2)_ *cross*(1, 1)_*1/3***	***Rand_box*(2, 2)_ *cross*(1, 1)_*1/2***	***Rand_box*(2, 2)_ *cross*(1, 1)_*2/3***
Initialization	Random by similarity	Random by similarity	Random by similarity
Type of swapping	Per phase (random, $box(2,2)$, $cross(1,1)$)	Per phase (random, $box(2,2)$, $cross(1,1)$)	Per phase (random, $box(2,2)$, $cross(1,1)$)
Single or multiple	Multiple	Multiple	Multiple
Number of sets	2	2	2
Occupancy	10^{-3}	10^{-3}	10^{-3}
Algorithmic details	Time-based model that allocates ⅓ of total runtime to each type of swapping	Time-based model that allocates ½ of total runtime to random swapping and the remaining ¼ to each of special swapping	Time-based model that allocates ⅔ of total runtime to random swapping and ⅙ to each of special swapping

Table 5A.4 Description of time-based models that use three types of swapping

Name	***Rand_box*(2, 2)_ *cross*(1, 1)_*1/3***	***Rand_box*(2, 2)_ *cross*(1, 1)_*1/2***	***Rand_box*(2, 2)_ *cross*(1, 1)_*2/3***
Total runtime	600 sec	600 sec	600 sec
Average initial value	210588	209265	208855
Highest initial value	216325	216595	216240
Lowest initial value	207280	204320	203200
Range of initial values	9045	12275	13040
Average final value	6187	5234	5227
Highest final value	8655	5765	5690
Lowest final value	5410	4765	4890
Range of final values	3245	1000	800
Average time for 95% of final value	137.38 sec	124.95 sec	122.92 sec
Average time for 99% of final value	327.96 sec	306.92 sec	337.22 sec

Table 5B.4 Results of time-based models that use three types of swapping

These results lead us to question the sufficiency of the use of a single special swapping, rather than two special swappings, to incorporate with random swapping. Tables 5A.5 and 5A.6 present three models that use random swapping, and $box(2,2)$ or $cross(1,1)$ swapping, where runtime allocations to random swapping are the first ¼, ½ and ¾ of the total runtime of 600 seconds. The results are reported through Tables 5B.5 and 5B.6. For models that use $box(2,2)$ swapping in addition to random swapping, the average final penalty values are respectively 5890, 4990 and 5015. For models that use $cross(1,1)$ swapping in addition to random swapping, the average final penalty values are respectively 7950, 6039 and 5161. We see that, when $box(2,2)$ is the secondary

swapping type, the models allocating the first ½ and ¾ of the total runtime to random swapping beat the adapted baseline model; when $cross(1,1)$ is the secondary swapping type, only the model that allocates the first ¾ of the total runtime to random swapping beat the adapted baseline model. Additionally, these three models show better performance than the best model combining all three swappings (random, $box(2,2)$ and $cross(1,1)$), which has an average final value of 5227. This indicates that, to compliment random swapping, the use of two special swappings is not necessary, as one special swapping seems sufficient. However, this is conditional to the runtime allocations and the type of special swapping used. From the previous results, we can deduce that a larger proportion of runtime needs to be allocated to random swapping when the secondary swapping type is more precise.

Name	***Rand_box*(2, 2)_*1/4***	***Rand_box*(2, 2)_*1/2***	***Rand_box*(2, 2)_*3/4***
Initialization	Random by similarity	Random by similarity	Random by similarity
Type of swapping	Per phase (random, $box(2,2)$)	Per phase (random, $box(2,2)$)	Per phase (random, $box(2,2)$)
Single or multiple	Multiple	Multiple	Multiple
Number of sets	2	2	2
Occupancy	10^{-3}	10^{-3}	10^{-3}
Algorithmic details	Time-based model that allocates ¼ of total runtime to random swapping and remaining ¾ to special swapping $box(2,2)$)	Time-based model that allocates ½ of total runtime to random swapping and remaining ½ to special swapping $box(2,2)$)	Time-based model that allocates ¾ of total runtime to random swapping and remaining ¼ to special swapping $box(2,2)$)

Table 5A.5 Description of time-based models that use random swapping and special swapping $box(2,2)$

Name	***Rand_box*(2, 2)_*1/4***	***Rand_box*(2, 2)_*1/2***	***Rand_box*(2, 2)_*3/4***
Total runtime	600 sec	600 sec	600 sec
Average initial value	210455	209319	209995
Highest initial value	218910	216290	216850
Lowest initial value	203620	203490	201165
Range of initial values	15290	12800	15685
Average final value	5890	4990	5015
Highest final value	6710	5330	5255
Lowest final value	5160	4595	4750
Range of final values	1550	735	505
Average time for 95% of final value	114.96 sec	120.21 sec	132.51 sec
Average time for 99% of final value	330.20 sec	334.09 sec	330.38 sec

Table 5B.5 Results of time-based models that use random swapping and special swapping $box(2,2)$

Name	***Rand_cross*(1, 1)_*1/4***	***Rand_cross*(1, 1)_*1/2***	***Rand_cross*(1, 1)_*3/4***
Initialization	Random by similarity	Random by similarity	Random by similarity
Type of swapping	Per phase (random, $cross(1,1)$)	Per phase (random, $cross(1,1)$)	Per phase (random, $cross(1,1)$)
Single or multiple	Multiple	Multiple	Multiple
Number of sets	2	2	2
Occupancy	10^{-3}	10^{-3}	10^{-3}
Algorithmic details	Time-based model that allocates ¼ of total runtime to random swapping and remaining ¾ to special swapping $cross(1,1)$)	Time-based model that allocates ½ of total runtime to random swapping and remaining ½ to special swapping $cross(1,1)$)	Time-based model that allocates ¾ of total runtime to random swapping and remaining ¼ to special swapping $cross(1,1)$)

Table 5A.6 Description of time-based models that use random swapping and special swapping $cross(1,1)$

Name	***Rand_cross*(1, 1)_*1/4***	***Rand_cross*(1, 1)_*1/2***	***Rand_cross*(1, 1)_*3/4***
Total runtime	600 sec	600 sec	600 sec
Average initial value	209292	209292	211610
Highest initial value	215100	214815	219750
Lowest initial value	197225	200590	207440
Range of initial values	17875	14225	12310
Average final value	7950	6039	5161
Highest final value	8595	6635	5500
Lowest final value	7055	5375	4780
Range of final values	1540	1260	720
Average time for 95% of final value	103.33 sec	124.92 sec	130.90 sec
Average time for 99% of final value	245.49 sec	295.61 sec	348.85 sec

Table 5B.6 Results of time-based models that use random swapping and special swapping $cross(1,1)$

We move on with the exploration-based approach, bearing in mind one special swapping is sufficient to complement random swapping. Considering the previous observations from the time-based models, to ensure that a majority of the runtime is allocated to random swapping, we also consider adding a condition to limit the change in swapping type to the second ½ of the total runtime. Table 5A.7 presents two exploration-based approach models that use random swapping and $box(2,2)$ swapping. As shown in Table 5B.7, with an ongoing exploring time that must be 50 times superior to the last exploring time, the average final penalty value is 5473. With an ongoing exploring time that must be 100 times superior to the last, the final penalty value averages 5517.

95% of their respective average improvement is reached in 128.33 seconds and 137.75 seconds, whereas 99% of it is reached in 340.97 seconds and 360.75 seconds. These models show better performance and faster convergence than the adapted baseline model.

Name	***Rand_box*(2, 2)_*50***	***Rand_box*(2, 2)_*100***
Initialization	Random by similarity	Random by similarity
Type of swapping	Per phase (random, $box(2,2)$)	Per phase (random, $box(2,2)$)
Single or multiple	Multiple	Multiple
Number of sets	2	2
Occupancy	10^{-3}	10^{-3}
Algorithmic details	Exploration-based model that uses random swapping until past ½ of the total runtime and until ongoing exploring time is a factor of 50 to last exploring time, and then it uses $box(2,2)$ swapping	Exploration-based model that uses random swapping until past ½ of the total runtime and until ongoing exploring time is a factor of 100 to last exploring time, and then it uses $box(2,2)$ swapping

Table 5A.7 Description of exploration-based models that use random swapping and special swapping $box(2,2)$

Name	***Rand_box*(2, 2)_*50***	***Rand_box*(2, 2)_*100***
Total runtime	600 sec	600 sec
Average initial value	211258	209273
Highest initial value	221430	218635
Lowest initial value	201825	200680
Range of initial values	19605	17955
Average final value	5473	5517
Highest final value	6250	5890
Lowest final value	5120	5130
Range of final values	1130	760
Average time for 95% of final value	128.33 sec	137.75 sec
Average time for 99% of final value	340.97 sec	360.75 sec

Table 5B.7 Results of exploration-based models that use random swapping and special swapping $box(2,2)$

5.2.6 Room occupancy

The room occupancy is a constraint that is used to reduce the assignment of small courses (courses with small norms) to large rooms (rooms with large capacities) in Course Timetabling. A relatively small weight value is associated to this constraint. We take interest in this constraint's influence

on the search algorithm, as it stimulates changes but inevitably reinforces the acceptance of solutions that are not significantly improving. As room occupancy is not applicable in examination timetables, this section does not concern Examination Timetabling.

Name	***Baseline no occ***
Initialization	Random
Type of swapping	Random
Single or multiple	Single
Number of sets	2
Occupancy	0
Algorithmic details	Simplest model with the weight of the room occupancy constraint set to 0

Table 5A.8 Description of baseline model without the room occupancy constraint

Name	***Baseline no occ***
Total runtime	600 sec
Average initial value	216975
Highest initial value	220355
Lowest initial value	212375
Range of initial values	7980
Average final value	6444
Highest final value	6745
Lowest final value	6095
Range of final values	650
Average time for 95% of final value	113.32 sec
Average time for 99% of final value	321.12 sec

Table 5B.8 Results of baseline model without the room occupancy constraint

We investigate the usefulness of the room occupancy constraint by nullifying the constraint in the baseline model (setting its weight to 0), as described in Table 5A.8. With the average resulting penalty value of 6444 reported in Table 5B.8, this model does not outperform the baseline model. Moreover, it does not show better convergence than the baseline model, seeing that 95% of the average improvement is achieved within 113.32 seconds, and 99% within the 322.12 seconds. It is evident that the constraint of room occupancy, although associated to a very small weight value, have a great influence on the performance of a model. Rather than assigning a constant value to the room occupancy weight, we suggest creating models with active and inactive phases of the

room occupancy constraint, where the constraint weight takes a small positive value (defaulted to 10^{-3}) in active phases and a value of 0 in inactive phases. The phases can be triggered through a time-based approach and an exploration-based exploration. These phases are used so the algorithm can focus on bigger improvements while preventing disproportionate occupancy assignments.

Through time-based models shown in Table 5A.9, we test the end of the inactive phase and the start of the active phase at ¼, ½, and ¾ of the total runtime. The resulting average penalty values provided in Table 5B.9 are respectively 5170, 5072 and 5379. Through exploration-based models shown in Table 5A.10, we test a factor to the last exploring time of 50, 100 and 150, which respectively yields an average final penalty value of 5009, 5263 and 5191 presented in Table 5B.10. We recognize that, amongst our tests, the exploration-based performant than the time-based ones.

Name	***Occ_1/4***	***Occ_1/2***	***Occ_3/4***
Initialization	Random by similarity	Random by similarity	Random by similarity
Type of swapping	Random	Random	Random
Single or multiple	Multiple	Multiple	Multiple
Number of sets	2	2	2
Occupancy	Per phase (0, 10^{-3})	Per phase (0, 10^{-3})	Per phase (0, 10^{-3})
Algorithmic details	Occupancy penalty weight is set to 0 in first ¼ of the total runtime and then takes a value 10^{-3} for the remaining ¾ of the runtime	Occupancy penalty weight is set to 0 in first ½ of the total runtime and then takes a value 10^{-3} for the remaining ½ of the runtime	Occupancy penalty weight is set to 0 in first ¾ of the total runtime and then takes a value 10^{-3} for the remaining ¼ of the runtime

Table 5A.9 Description of time-based models controlling the room occupancy constraint

Name	***Occ_1/4***	***Occ_1/2***	***Occ_3/4***
Total runtime	600 sec	600 sec	600 sec
Average initial value	211498	210427	208524
Highest initial value	217100	221450	212755
Lowest initial value	206215	203730	201960
Range of initial values	10885	17720	10795
Average final value	5170	5072	5379
Highest final value	5910	5330	6135
Lowest final value	4825	4630	4830
Range of final values	1085	700	1305
Average time for 95% of final value	129.28 sec	134.88 sec	136.85 sec
Average time for 99% of final value	349.40 sec	372.66 sec	362.39 sec

Table 5B.9 Results of time-based models controlling the room occupancy constraint

Name	***Occ_50***	***Occ _100***	***Occ_150***
Initialization	Random by similarity	Random by similarity	Random by similarity
Type of swapping	Random	Random	Random
Single or multiple	Multiple	Multiple	Multiple
Number of sets	2	2	2
Occupancy	Per phase (0, 10^{-3})	Per phase (0, 10^{-3})	Per phase (0, 10^{-3})
Algorithmic details	Occupancy penalty weight is set to 0 until ongoing exploring time is a factor of 50 to last exploring time, and then it takes a value 10^{-3} for the remaining runtime	Occupancy penalty weight is set to 0 until ongoing exploring time is a factor of 100 to last exploring time, and then it takes a value 10^{-3} for the remaining runtime	Occupancy penalty weight is set to 0 until ongoing exploring time is a factor of 150 to last exploring time, and then it takes a value 10^{-3} for the remaining runtime

Table 5A.10 Description of exploration-based models controlling the room occupancy constraint

Name	***Occ _50***	***Occ_100***	***Occ_150***
Total runtime	600 sec	600 sec	600 sec
Average initial value	210790	210024	211203
Highest initial value	215455	216250	219640
Lowest initial value	207245	203215	201755
Range of initial values	8210	13035	17885
Average final value	5009	5263	5191
Highest final value	5985	5670	5645
Lowest final value	4440	4885	4600
Range of final values	1545	785	1045
Average time for 95% of final value	135.12 sec	144.26 sec	125.06 sec
Average time for 99% of final value	347.75 sec	358.66 sec	350.36 sec

Table 5B.10 Results of exploration-based models controlling the room occupancy constraint

5.2.7 Room types

The room constraints presented in **Section 4.1.1** relate to three room characteristics: capacity, layout, and building. When evaluating a room constraint, for a same timeslot, rooms with the same characteristics yield equivalent penalties. Hence, these rooms can be considered equivalent to one another. Instead of defining the rooms by their room characteristics individually, it would be more beneficial to incorporate their combined characteristics. The intention is to use the combined characteristics as a type of screening before evaluation to lower the number of equivalent swaps.

By reducing the combinatorial symmetry of the problem with this method, the algorithm allocates less time in the exploration of equivalent solutions and focuses on improvement. To incorporate combined characteristics of rooms, we join rooms with similar traits to create clusters based on room types. While the layout and building characteristics are qualitative data with few possible values, respectively 6 and 2 possible values for layout and building characteristics, the room capacity is a quantitative data with 26 possible values over 67 rooms. We suggest creating clusters of rooms with similar capacities, rather than equivalent capacities. This leads to a smaller number of room types (16 room types instead of 26 room types out of the 67 rooms), allowing for better reduction of combinatorial symmetry. The effective capacity refers to the capacity representing a room type, whereas the real capacity refers to the true capacity of a room. We consider two options in defining the effective capacity through the rooms within a room type: minimal capacity and maximal capacity. Using the minimal capacity ensures that a course that respects the minimal capacity will inevitably respect the capacity of all time-rooms under this room type. However, it causes courses that require a larger number of seats than the largest room type that uses the minimal capacity to always be wrongly assigned in terms of the capacity constraint. Using the maximal capacity would solve this issue, but it does not guarantee that a course respecting the maximal capacity will inevitably respect the capacity of all time-rooms under this room type. Both options can be beneficial in different instances.

Name	***Roomtype_min***	***Roomtype_max***
Initialization	Random by similarity	Random by similarity
Type of swapping	Random	Random
Single or multiple	Multiple	Multiple
Number of sets	2	2
Occupancy	0	0
Algorithmic details	Uses room types (with room occupancy weight set to 0 and with minimal capacity of room types as effective capacity))	Uses room types (with room occupancy weight set to 0 and with maximal capacity of room types as effective capacity))

Table 5A.11 Description of models only room types with minimal or maximal capacity as effective capacity

Name	***Roomtype_min_all***	***Roomtype_max_all***
Total runtime	600 sec	600 sec
Average initial value	211297	211913
Highest initial value	219400	220715
Lowest initial value	204875	206480
Range of initial values	14525	14235
Average final value	5566	11480
Highest final value	6505	13815
Lowest final value	4620	10335
Range of final values	1885	3480
Average time for 95% of final value	136.74 sec	142.51 sec
Average time for 99% of final value	366.00 sec	353.05 sec

Table 5B.11 Results of models only room types with minimal or maximal capacity as effective capacity

When the model is actively using room types, rather than specific rooms, the tuple that represents the effective capacity must be the room type capacity tuple. The effective capacity of a room type can correspond to the minimal or maximal capacity amongst the room type. Since both have different usage, as mentioned, we put both through tests to assess their effects on the models described in Table 5A.11. In models solely using room types, the average final penalty values reported in Table 5B.11 are respectively 5566 with the minimal capacity and 11480 with the maximal capacity considering an average initial penalty value of 211297 and 211913. Note that to maintain comparability between the models, all penalty values provided in tables labelled as 5B are real values, meaning that the reported values are based on the real capacity, rather than the effective capacity. From these results, the model that uses the minimal capacity clearly shows better performance than the other. Amongst the established tests, with a similar initial penalty value, the models with minimal capacity return a lower final penalty value, with the highest value being more than 1.5x lower than the lowest value of the models with maximal capacity. This reinforces our previous statement, saying that, if a course satisfies the capacity constraint of a room type using minimal capacity, it will necessarily satisfy the capacity constraint of any room within the room type, which is not the case with the maximal capacity. Nonetheless, both models do not beat the adapted baseline model. This could be due to the lack in precision of the capacity, implying that room types are not meant to be used exclusively. Instead, we suggest the use of room types and rooms in phases with this particular order, to allow for reduced combinatorial symmetry as well as precision in the solution. It is important to note that when room types are being used, the

room occupancy constraint must be nullified, since the effective capacity would lead to incorrect calculations of occupancy. For models that alternate between the use of room types and rooms through repeated phases, there must be a re-evaluation of the current state to accurately reflect the room capacity and room occupancy as the effective capacity changes to real capacity or vice-versa. Rather than a complete evaluation of the state at a phase change, by constantly keeping track of the time penalty value (i.e., the penalty values of the time constraints), only the rooms need to be re-evaluated and summed to the time penalty value for an accurate evaluation to carry on.

Name	***Roomtype_max_1/4***	***Roomtype_max_1/2***	***Roomtype_max_3/4***
Initialization	Random by similarity	Random by similarity	Random by similarity
Type of swapping	Random	Random	Random
Single or multiple	Multiple	Multiple	Multiple
Number of sets	2	2	2
Occupancy	Per phase (0, 10^{-3})	Per phase (0, 10^{-3})	Per phase (0, 10^{-3})
Algorithmic details	First phase uses room types (with room occupancy weight set to 0 and with maximal capacity as effective capacity) for first ¼ of total runtime, then the remaining ¾ of runtime is dedicated to second phase which uses rooms (with room occupancy weight set to 10^{-3})	First phase uses room types (with room occupancy weight set to 0 and with maximal capacity as effective capacity) for first ½ of total runtime, then the remaining ½ of runtime is dedicated to second phase which uses rooms (with room occupancy weight set to 10^{-3})	First phase uses room types (with room occupancy weight set to 0 and with maximal capacity as effective capacity) for first ¼ of total runtime, then the remaining ¾ of runtime is dedicated to second phase which uses rooms (with room occupancy weight set to 10^{-3})

Table 5A.12 Description of time-based models that use room types with maximal capacity as effective capacity

Name	***Roomtype_max_1/4***	***Roomtype_max_1/2***	***Roomtype_max_3/4***
Total runtime	600 sec	600 sec	600 sec
Average initial value	210218	212010	208125
Highest initial value	216540	222515	212455
Lowest initial value	205840	205500	201030
Range of initial values	10700	17015	11425
Average final value	5348	5323	5467
Highest final value	5850	5695	5920
Lowest final value	4690	4825	4690
Range of final values	1160	870	1230
Average time for 95% of final value	161.68 sec	244.39 sec	254.39 sec
Average time for 99% of final value	365.51 sec	379.71 sec	470.63 sec

Table 5B.12 Results of time-based models that use room types with maximal capacity as effective capacity

Name	***Roomtype_min_1/4***	***Roomtype_min_1/2***	***Roomtype_min_3/4***
Initialization	Random by similarity	Random by similarity	Random by similarity
Type of swapping	Random	Random	Random
Single or multiple	Multiple	Multiple	Multiple
Number of sets	2	2	2
Occupancy	Per phase (0, 10^{-3})	Per phase (0, 10^{-3})	Per phase (0, 10^{-3})
Algorithmic details	First phase uses room types (with room occupancy weight set to 0 and with minimal capacity as effective capacity) for first ¼ of total runtime, then the remaining ¾ of runtime is dedicated to second phase which uses rooms (with room occupancy weight set to 10^{-3})	First phase uses room types (with room occupancy weight set to 0 and with minimal capacity as effective capacity) for first ½ of total runtime, then the remaining ½ of runtime is dedicated to second phase which uses rooms (with room occupancy weight set to 10^{-3})	First phase uses room types (with room occupancy weight set to 0 and with minimal capacity as effective capacity) for first ¼ of total runtime, then the remaining ¾ of runtime is dedicated to second phase which uses rooms (with room occupancy weight set to 10^{-3})

Table 5A.13 Description of time-based models that use room types with minimal capacity as effective capacity

Name	***Roomtype_min_1/4***	***Roomtype_min_1/2***	***Roomtype_min_3/4***
Total runtime	600 sec	600 sec	600 sec
Average initial value	207231	208558	211743
Highest initial value	212980	214360	218505
Lowest initial value	201200	197665	206620
Range of initial values	11780	16695	11885
Average final value	5374	5459	5515
Highest final value	6255	5990	6065
Lowest final value	4980	5000	5120
Range of final values	1275	990	945
Average time for 95% of final value	148.16 sec	143.13 sec	137.92 sec
Average time for 99% of final value	365.06 sec	379.20 sec	377.30 sec

Table 5B.13 Results of time-based models that use room types with minimal capacity as effective capacity

We propose three time-based models that either use the minimal capacity or the maximal capacity as the effective capacity with the room type phase active until the room phase at ¼, ½ and ¾ of the total runtime, as shown in Tables 5A.11 and 5A.12. With the minimal capacity as the effective capacity, Table 5B.12 reports average final penalty values 5374, 5459 and 5515. With the maximal capacity as the effective capacity, Table 5B.11 reports average final penalty values of 5348, 5323 and 5467. Amongst these models, the model that use maximal capacity with a phase change at ½

of the runtime seems to be the best model. Not only does this model show a lower average penalty value, but it also has the shortest range of final penalty value.

Imposing the maximal capacity as the effective capacity for the next models, we move on to analyze the exploration-based models presented in Table 5A.14. With the phase change triggered by an ongoing exploring time that is a factor of 25 of the last exploring time, the model results provided in Table 5B.14 reveals an average final penalty value of 4970, whereas a factor of 50 would result in an average final penalty value of 5270. 95% of the respective average improvement is obtained within 130.90 seconds and 170.62 seconds, whereas 99% of the respective average improvement is obtained within 361.24 seconds and 345.22 seconds.

Name	***Room type _max_25***	***Roomtype_max_50***
Initialization	Random by similarity	Random by similarity
Type of swapping	Random	Random
Single or multiple	Multiple	Multiple
Number of sets	2	2
Occupancy	Per phase (0, 10^{-3})	Per phase (0, 10^{-3})
Algorithmic details	First phase uses room types (with room occupancy weight set to 0 and with maximal capacity as effective capacity) until ongoing exploring time is a factor of 25 to the last exploring time, then the remaining runtime is dedicated to second phase which uses rooms (with room occupancy weight set to 10^{-3})	First phase uses room types (with room occupancy weight set to 0 and with maximal capacity as effective capacity) until ongoing exploring time is a factor of 50 to the last exploring time, then the remaining runtime is dedicated to second phase which uses rooms (with room occupancy weight set to 10^{-3})

Table 5A.14 Description of exploration-based models that use room types with maximal capacity as effective capacity

Name	***Room type _max_25***	***Roomtype_max_50***
Total runtime	600 sec	600 sec
Average initial value	210704	211007
Highest initial value	216445	216375
Lowest initial value	203715	199390
Range of initial values	12730	16985
Average final value	4970	5270
Highest final value	5245	6225
Lowest final value	4490	4685
Range of final values	755	1540
Average time for 95% of final value	130.90 sec	170.62 sec
Average time for 99% of final value	361.24 sec	345.22 sec

Table 5B.14 Results of exploration-based models that use room types with maximal capacity as effective capacity

5.3 Proposed methodology

The experimental tests from the previous section allow us to identify the best models for course lecture and examination timetables at HEC Montréal, considering the data simulated. Supported by these numerical results, we propose two models to solve the Course Timetabling and Examination Timetabling problems. In the following section, we will explain the reasoning behind our choice of models, and reinforce our propositions with a robustness test. Finally, we present the constraint satisfaction derived from the best model in both contexts.

5.3.1 Best Course Timetabling model

At the extend of the experimental tests, the following models are the most noteworthy for their performance on this instance of Course Timetabling: the time-based models that use random swapping and $box(2,2)$ swapping at ½ and ¾ of the total runtime (see models *rand_box*$(2,2)$*_1/2* and *rand_box*$(2,2)$*_3/4* in Tables 5A.5 and 5B.5), the exploration-based model that controls room occupancy with a factor of 50 (see model *occ_50* in Tables 5A.10 and 5B.10), and the exploration-based models that use room types with the maximal capacity, with a factor of 25 and 50 (see models *roomtype_max_25* and *roomtype_max_50* in Tables 5A.14 and 5B.14).

Amongst this list of good performing models, we identify the exploration-based model that uses room types with the maximal capacity and a factor 25, as the best Course Timetabling model. This

model is built around two phases, where it uses room types (with the maximal capacity of the rooms within a room type as the effective capacity) in the first phase and rooms (using the real capacity of the rooms) in the second phase. The phase change is triggered when the ongoing exploring time exceeds 25 times the last exploring time. To recall this model's specific parameters and results from the tests conducted, refer to Tables 5A.14 and 5B.14. This model is selected as the best model over the others because of its better performance and convergence. In terms of final penalty value, it shows the lowest average, in addition to a relatively small range. From an average initial value of 210704, this model achieves an average final penalty value of 4970, with a range of 755. While the model could not beat the performance of the adapted baseline model that underwent an extended runtime, it still shows better performance than the baseline and adapted baseline models on a similar runtime of 600 seconds. Additionally, given such a low average final penalty value, the time to reach 99% of its final value is considered relatively short, i.e., 361.24 seconds over 600 seconds of total runtime.

To consolidate on the model's robustness, we put this model through 10 runs of 1800 seconds. This second round of testing is done to compare these results to the experimental results achieved within 600 seconds. For both rounds of testing with different total runtime, Table 5.3 presents the average, highest, lowest and range of the initial and final values. From Table 5.3, we can see that the average final penalty value is considerably lower on the second round of testing, i.e., respectively 3346 and 4970. Additionally, the highest final penalty value with a runtime of 1800 seconds is lower than the lowest final penalty value with a runtime of 600 seconds, i.e., respectively 3635 and 4490. This indicates that, all the results from a runtime of 1800 seconds are better than those from a runtime of 600 seconds. However, the comparison in the results of both rounds of testing needs to take into account the difference in initial penalty values as well. Considering the same parameters and data in both rounds of testing, we observe that the average initial penalty value is slightly lower in the second round of testing, i.e., respectively 209501 and 210704. Although it is lower, the average initial penalty value is only lower by 1203, whereas the average final penalty value is lower by 1624. Therefore, while the initial penalty is lower in average, the higher difference in average final values implies that a longer runtime leads to better results. As for the range in values, the range of the initial values is relatively larger on the second round of testing, i.e., respectively 19305 and 12730, and the range of final penalty values is much smaller

than the first round of testing, i.e., respectively 520 and 755. When representing these last numbers as a percentage of their respective average final penalty value, they amount to about the same, i.e., 15.54% and 15.19%. Therefore, we believe that, for the model we propose as the best Course Timetabling model, a longer runtime allows for better results to be generated with nearly no compromise on the variability. We judge that the variability is low for both 600 seconds and 1800 seconds of runtime, implying that a single execution of the model will unlikely return a penalty value that is very far from the average penalty value.

Name	***Roomtype_max_25***	***Roomtype_max_25***
Total runtime	600 sec	1800 sec
Average initial value	210704	209501
Highest initial value	216445	219905
Lowest initial value	203715	200600
Range of initial values	12730	19305
Average final value	4970	3346
Highest final value	5245	3635
Lowest final value	4490	3115
Range of final values	755	520

Table 5.3 Comparison of the best Course Timetabling model's penalty values between 600 seconds and 1800 seconds

To illustrate the constraint satisfaction obtained through the models, we describe the outcome from a single execution of a model with a runtime of 600 seconds. While any good performing model would suffice, we choose to use the model we identified as the best Course Timetabling model, which corresponds to the model that uses room types with the maximal capacity, with a factor of 25. In a single execution of 600 seconds, this model's real penalty value progresses from 206850 to 4595. Table 5.4 serves as a support to report numbers about each type of constraint. For each constraint type, we find the total number of constraints, the number of constraints that is left unsatisfied with the resulting timetable, and the total penalty value accounted by the constraint type. Given the constraint weights listed in Table 5.1 from **Section 5.1**, the resulting timetable fully satisfies the capacity, similarity groupings and exclusivity groupings, with rooms occupied at 85.70% in average. The Table 5.4 shows that all 32 layout necessities are satisfied, considering that there are 80 time-rooms with special layouts and that all courses with the wrong layout necessity assigned to any of those time-rooms are also subject to penalization. The layout and building preferences constraints are satisfied for the most part, with only 5 out of 377 layout requests (1.33%) and 7 out of 390 building location requests (1.79%) unsatisfied. While 113

courses were penalized on the timeslot preference level, only 27 out of the 743 courses with timeslot preferences (3.63%) do not have any of their preferences satisfied. While the timeslot preferences accounts for the majority of the final penalty value, most courses (96.37%) are at least partially satisfied in terms of timeslot preference.

Constraint type	***Total number of constraints***	***Number of unsatisfied constraints***	***Total penalty value***
Timeslot preference	743	113	4280
Similarity grouping	62	0	0
Exclusivity grouping	141	0	0
Capacity	802	0	0
Layout necessity	32 + 80	0	0
Layout preference	377	5	175
Building	390	7	140

Table 5.4 Constraint satisfaction of a resulting course lecture timetable

5.3.2 Best Examination Timetabling model

The models developed for examinations are the same as for course lectures. However, the models related to room occupancy are omitted since this aspect is irrelevant in the examination context. While the results were not explicitly exposed in the previous section, they show similar dynamic between the examination models and the course lecture models. Through Table 5.5, we provide the results of a few selected Examination Timetabling models. This result table provides, for each model listed, the average, highest, lowest and range of the initial and final penalty value, as well as the average time to reach 95% and 99% of the respective final penalty value. These models' parameters are the same as the Course Timetabling models under the same name, and they can be found in description tables labelled as 5A in **Section 5.2**. Note that the occupancy in tables labelled as 5A is unapplicable for the examination context. The similarity in dynamic between the models is illustrated by Table 5.5. By comparing the average final penalty values, we can see that the adapted baseline model is more performant that the baseline model, i.e., respectively 85178 and 475783 in average final penalty value. Additionally, like in Course Timetabling, the time-based models that use random swapping and special swapping $box(2,2)$ at ½ and ¾ of the runtime, with a respective average final penalty value of 79153 and 83482, also outperform both baseline and adapted baseline models. We list the following models as the most noteworthy models for their performance in Examination Timetabling: the time-based models that use random swapping and $box(2,2)$ swapping at ½ and ¾ of the total runtime (see models $rand\_box(2,2)\_1/2$ and

rand_box(2,2)*_3/4* in Tables 5A.5 and 5.5) and the adapted baseline model (see model *adapted baseline* in Tables 5A.1 and 5.5).

Amongst the models listed, we identify the time-based model that uses random swapping, and special swapping $box(2,2)$ after ½ of the total runtime as the best Examination Timetabling model. For this model's specific parameters, refer to the Table 5A.5. This model is designated as the best Examination Timetabling model because it achieves the lowest average penalty value within 600 seconds of runtime. As shown in Table 5.5, the final penalty value reaches 79153 in average. We also note that it takes around 325.35 seconds and 525.54 seconds in average out of 600 seconds to reach 95% and 99% of the final value.

Name	***Baseline***	***Adapted baseline***	***Rand_box*(2, 2)_ *1/2***	***Rand_box*(2, 2)_ *3/4***
Total runtime	600 sec	600 sec	600 sec	600 sec
Average initial value	1158318	399627	416704	409969
Highest initial value	1168029	414168	445405	446124
Lowest initial value	1148807	384905	377254	386358
Range of initial values	19222	29263	68151	59766
Average final value	475783	85178	79153	83482
Highest final value	493252	89140	79982	84434
Lowest final value	464072	81328	77871	81852
Range of final values	29180	7812	2111	2582
Average time for 95% of final value	473.42 sec	322.57 sec	325.35 sec	292.77 sec
Average time for 99% of final value	563.72 sec	516.92 sec	525.54 sec	506.11 sec

Table 5.5 Results of baseline and adapted baseline models and a time-based model that use random swapping and special swapping box(2,2)

Like in Course Timetabling, in order to test the model's robustness, we put this model through a second round of testing by executing 10 runs of 1800 seconds. For both rounds of testing with different total runtime, Table 5.6 presents the average, highest, lowest and range of the initial and final values. From Table 5.6, we can see that the average final penalty value is relatively lower on the second round of testing, i.e., respectively 64183 and 79153. We also note that the highest final penalty value with a runtime of 1800 seconds is lower than the lowest final penalty value with a runtime of 600 seconds, i.e., respectively 66880 and 77871. This indicates that, all results from a runtime of 1800 seconds are better than those from a runtime of 600 seconds. However,

considering the same parameters and data in both rounds of testing, we observe that the average initial penalty value is slightly lower in the latter one, i.e., respectively 402198 and 416704. We see that both initial and final values are lower in average on the second round of testing. Yet, when comparing the second round of testing to the first round of testing, the difference in average final penalty values, i.e., 14970, is larger than the difference in average initial penalty values, i.e., 14506. Therefore, although the second round of testing starts at a lower penalty value in average, we can still say that a longer runtime allows for better results. As for the range in values, the range of the initial values is slightly larger on the second round of testing, i.e., respectively 69973 and 68151, and similarly the range of final penalty values is also larger than the first round of testing, i.e., respectively 5267 and 2111. When represented as a percentage of the average final penalty value, the ranges correspond to respectively 8.21% and 2.67%. Therefore, we can say that, for the model we propose as the best Examination Timetabling model, a longer runtime allows for better results to be generated at the expense of a slight increase in variability. Although the variability is compromised, we judge that the variability is low in both 600 seconds and 1800 seconds of runtime, implying that a single execution of the model will unlikely return a penalty value that is very far from the average penalty value of a given total runtime.

Name	***Rand_box*(2, 2)_*1/2***	***Rand_box*(2, 2)_*1/2***
Total runtime	600 sec	1800 sec
Average initial value	416704	402198
Highest initial value	445405	427047
Lowest initial value	377254	357074
Range of initial values	68151	69973
Average final value	79153	64183
Highest final value	79982	66880
Lowest final value	77871	61613
Range of final values	2111	5267

Table 5.6 Comparison of the best Course Timetabling model's penalty values between 600 seconds and 1800 seconds

We move on to the constraint satisfaction obtained through a single execution of the best Examination Timetabling model for a total runtime of 600 seconds. The weight set for the examination constraints can also be found in **Section 5.1**, listed in Table 5.2. We use Table 5.6 to present, for each type of constraint in Examination Timetabling, the total number of constraints, as well as the number of unsatisfied constraints and the resulting total penalty value. In this execution, the penalty value progresses from 559047 to 165310 within 600 seconds. The resulting

timetable of the model proposed fully satisfies the constraints of timeslot prohibition, timeslot exclusivity grouping and one-day exclusivity grouping. The building preferences are also satisfied, but it is important to note that only less than 1% of sub-sections have these requests simulated. Although layout necessities are also few, considering 114 rooms with the concerned layouts, only 9 time-rooms are penalized. With 910 sub-sections involved in the building grouping constraint, 182 out of 910 (20.0%) are affected a penalty. As for room capacity, 122 out of 995 sub-sections (12.26%) are unsatisfied. Only 1 out of 910 (0.11%) were penalized for the similarity grouping. Contrastingly, for the 78 sub-sections with timeslot preferences, we note that 67.95% of them are satisfied in this aspect, meaning that they have at least one timeslot preference satisfied. Finally, the two-day exclusivity constraint is almost completely respected with 3 out of 90 sub-sections (0.03%) penalized.

Constraint	***Total number of constraints***	***Number of unsatisfied constraints***	***Total penalty value***
Timeslot preference	78	70	5880
Timeslot prohibition	618	0	0
Similarity grouping	910	1	900
Timeslot exclusivity grouping	90	0	0
One-day exclusivity grouping	90	0	0
Two-day exclusivity grouping	90	18	310
Capacity	995	122	61000
Layout necessity	1 + 114	9	6300
Building	2	0	0
Building grouping	910	182	91000

Table 5.7 Constraint satisfaction of a resulting examination timetable

6 Conclusion

The educational field shows strong interest in timetabling problems. This is partially because the process of timetabling is recurrent, and as an institution expands, the process becomes increasingly resource demanding. In fact, educational institutions, such as HEC Montréal, go through a lengthy manual process that can realistically extend to a few months of operation. This is due to the many interventions necessary in such a process. In this thesis, we proposed an algorithm to automate the process of timetabling. In the case of HEC Montréal, each model proposed for Course Timetabling and Examination Timetabling was able to return a timetable that satisfies most constraints after only 600 seconds. Through rigorous testing, we proved that the models are considerably robust, in the sense that they are capable of returning consistent results even in presence of randomness. Furthermore, we showed that a runtime of 1800 seconds can be very beneficial, as the models consistently returned even better results. We strongly believe that the implementation of this algorithm can immensely improve the timetabling process within the institution. This automation is deemed to reduce the time for active resolution in the process from several days down to a few hours. Additionally, such automation allows for a degree of versatility that is inconceivable in manual timetabling.

Through the literature review in **Chapter 2**, we learned that, despite the numerous existing research work conducted on the topic of timetabling, no efficient solving method has yet been found for these problems categorized as NP-complete. However, the most notable solving techniques for these problems of high complexity are metaheuristics, due to their ability to cover a wide variety of problems and provide high quality solutions. In **Chapter 3,** we solidified the problem instance and the goal of the algorithm, which is to produce conflict-free timetables that mostly satisfy students and lecturers. With this in consideration, in **Chapter 4**, we proposed a heuristic algorithm based on local search to solve HEC Montréal's Course Timetabling and Examination Timetabling problems. This heuristic searches for improving solutions by exploring complete and feasible solutions. By imposing a single hard constraint (being that all courses must be assigned to a time-room), we position for a more random search in a search space that is less constrained. The key elements of this algorithm are 1) the partial evaluation; 2) the distinction in time-specific and room-specific constraints; 3) the clustering of similar rooms into room types.

First, partial evaluation, in contrast to complete evaluation, leads to a significant economy in evaluation time. By structuring the time-room and course data into lists, we can follow the changes of the solution generated by a swap, and directly evaluate the changes in the affected time-rooms and courses. This framework ultimately eliminates repeated calculations. Second, knowing that timetabling has a dimension of time and space, we can identify scenarios where swaps only affect one dimension. By recognizing the distinction between time-specific and room-specific constraints, we can avoid calculations of unaffected constraints. Third, clustering rooms with similar characteristics to create room types allows to avoid unimpactful swaps, contributing to a reduction of the problem's combinatorial symmetry. Numerous models were then presented and evaluated on their performance and convergence in **Chapter 5**. All models presented for Course Timetabling and Examination Timetabling were able to easily return timetables with the majority of the constraints satisfied within a reasonable timeframe. From the experimental results, we proposed two models, one for Course Timetabling and another for Examination Timetabling. The experimental results of this algorithm show high convictions to the purpose of automating the timetabling process of HEC Montréal.

In addition to its optimization ability, the algorithm we proposed shows great simplicity and flexibility, which is essential for its capability in covering similar scheduling problems. We strongly believe that the three key elements of this algorithm (partial evaluation, distinction in time-specific and room-specific constraints, and clustering of similar rooms) can be beneficial for other optimization problems in the scheduling sphere. Concerning future works, the application of the algorithm on a variety of scheduling problems can provide an avenue of extension to prove the versatility of this heuristic algorithm. Through this extension, it would be interesting to develop on the challenges in adapting the algorithm proposed to a different problem instance. An emphasis on the difficulty of the instance and its likeness to the Course Timetabling or Examination Timetabling of HEC Montréal could lead the way to more interesting discussions. Moreover, since we are unable to prove the optimality of the solution with the heuristic proposed, the suggested extension could provide more depth to the heuristic's optimization capacities. This thesis allowed numerous ideas to be studied in the search of a well-rounded algorithm. To maintain a natural flow and to keep the thesis within its scope, a few ideas had to be omitted upon their inconclusive results. For instance, the types of swaps discussed in **Section 4.6.1** could lead to interesting models, but

none were found within the extend of our model testing. Also, a model inspired by VNS was created, but was ultimately disregarded. The recent success of VNS in metaheuristics reported in the literature review from **Chapter 2** led to the idea to incorporate a shaking procedure to the algorithm. This procedure aims to avoid local optima, and we approached this by periodically accepting non-improving swaps in a probabilistic manner. In such a model, a shaking phase is initiated when no improving swap has been found in a certain amount of time. While these two ideas were not pursued due to inconclusive test results, they can nonetheless create ample opportunities for further research. Perhaps, these suggestions for future works could shed light on the complexity theory problem of P vs. NP mentioned in **Chapter 2**, and hopefully lead the way to an efficient algorithm to solve a NP-complete problem.

Bibliography

Abdullah, S., Burke, E. K., & McCollum, B. (2007, September). A hybrid evolutionary approach to the university course timetabling problem. In *2007 IEEE congress on evolutionary computation* (pp. 1764-1768). IEEE.

Abdullah, S., Turabieh, H., McCollum, B., & McMullan, P. (2012). A hybrid metaheuristic approach to the university course timetabling problem. *Journal of Heuristics*, *18*(1), 1-23.

Alsmadi, O. M. K., Za'er, S., Abu-Al-Nadi, D. I., & Algsoon, A. (2011, May). A novel genetic algorithm technique for solving university course timetabling problems. In *International Workshop on Systems, Signal Processing and their Applications, WOSSPA* (pp. 195-198). IEEE.

Alvarez-Valdes, R., Crespo, E., & Tamarit, J. M. (2002). Design and implementation of a course scheduling system using tabu search. *European Journal of Operational Research*, *137*(3), 512-523.

Asham, G. M., Soliman, M. M., & Ramadan, A. R. (2011). Trans genetic coloring approach for timetabling problem. Artificial Intelligence Techniques Novel Approaches & Practical Applications, IJCA, 17-25.

Asmuni, H., Burke, E. K., & Garibaldi, J. M. (2005, September). Fuzzy multiple heuristic ordering for course timetabling. In *The Proceedings of the 5th United Kingdom Workshop on Computational Intelligence (UKCI05), London, UK* (pp. 302-309).

Asmuni, H., Burke, E. K., Garibaldi, J. M., McCollum, B., & Parkes, A. J. (2009). An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. *Computers & Operations Research*, *36*(4), 981-1001.

Aubin, J., & Ferland, J. A. (1989). A large scale timetabling problem. Computers & Operations Research, 16(1), 67-77.

Ayob, M., Burke, E. K., & Kendall, G. (2006). An iterative re-start variable neighbourhood search for the examination timetabling problem. *Practice and Theory of Automated Timetabling. LNCS, 1153*, 336-344.

Babaei, H., Karimpour, J., & Hadidi, A. (2015). A survey of approaches for university course timetabling problem. *Computers & Industrial Engineering*, *86*, 43-59.

Bakir, M. A., & Aksop, C. (2008). A 0-1 integer programming approach to a university timetabling problem. Hacettepe Journal of Mathematics and Statistics, 37(1), 41-55.

Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A., & Urli, T. (2016). Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research*, *65*, 83-92.

Borchani, R., Elloumi, A., & Masmoudi, M. (2017). Variable neighborhood descent search based algorithms for course timetabling problem: Application to a Tunisian University. *Electronic Notes in Discrete Mathematics*, *58*, 119-126.

Burke, E. K., Elliman, D., & Weare, R. (1994, September). A genetic algorithm based university timetabling system. In *Proceedings of the 2nd east-west international conference on computer technologies in education* (Vol. 1, pp. 35-40).

Burke, E., Bykov, Y., Newall, J., & Petrovic, S. (2004). A time-predefined local search approach to exam timetabling problems. *Iie Transactions*, *36*(6), 509-528.

Burke, E. K., Eckersley, A. J., McCollum, B., Petrovic, S., & Qu, R. (2010). Hybrid variable neighbourhood approaches to university exam timetabling. *European Journal of Operational Research*, *206*(1), 46-53.

Chaudhuri, A., & Kajal, D. (2010). Fuzzy genetic heuristic for university course timetable problem. *Int. J. Advance. Soft Comput. Appl*, *2*(1), 100-121.

Chen, M. C., Goh, S. L., Sabar, N. R., & Kendall, G. (2021). A survey of university course timetabling problem: perspectives, trends and opportunities. *IEEE Access*, *9*, 106515-106529.

Cooper, T. B., & Kingston, J. H. (1995, August). The complexity of timetable construction problems. In International conference on the practice and theory of automated timetabling (pp. 281-295). Springer, Berlin, Heidelberg.

Dandashi, A., & Al-Mouhamed, M. (2010, May). Graph coloring for class scheduling. In ACS/IEEE International Conference on Computer Systems and Applications-AICCSA 2010 (pp. 1-4). IEEE.

Daskalaki, S., Birbas, T., & Housos, E. (2004). An integer programming formulation for a case study in university timetabling. *European journal of operational research*, *153*(1), 117-135.

Daskalaki, S., & Birbas, T. (2005). Efficient solutions for a university timetabling problem through integer programming. *European journal of operational research*, *160*(1), 106-120.

Di Gaspero, L., Mizzaro, S., & Schaerf, A. (2004, August). A multiagent architecture for distributed course timetabling. In *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2004)* (pp. 471-474).

Di Gaspero, L., & Schaerf, A. (2000, August). Tabu search techniques for examination timetabling. In *International Conference on the Practice and Theory of Automated Timetabling* (pp. 104-117). Springer, Berlin, Heidelberg.

Erben, W., & Keppler, J. (1995, August). A genetic algorithm solving a weekly course-timetabling problem. In *International Conference on the Practice and Theory of Automated Timetabling* (pp. 198-211). Springer, Berlin, Heidelberg.

Golabpour, A., Shirazi, H. M., Farahi, A., Kootiani, A. Z. M., & Beigi, H. (2008, December). A fuzzy solution based on Memetic algorithms for timetabling. In *2008 International Conference on MultiMedia and Information Technology* (pp. 108-110). IEEE.

Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2017). Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, *5*(3), 423-454

Hertz, A. (1991). Tabu search for large scale timetabling problems. *European journal of operational research*, *54*(1), 39-47.

Hoos, H. H., & Stützle, T. (2004). Stochastic local search: Foundations and applications. Elsevier.

Kohshori, M. S., & Abadeh, M. S. (2012). Hybrid genetic algorithms for university course timetabling. *International Journal of Computer Science Issues (IJCSI)*, *9*(2), 446.

Lü, Z., & Hao, J. K. (2010). Adaptive tabu search for course timetabling. *European journal of operational research*, *200*(1), 235-244.

McCollum, Barry. (2006). University timetabling: Bridging the gap between research and practice. *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling.*

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, *24*(11), 1097-1100.

Muklason, A., Irianti, R. G., & Marom, A. (2019). Automated course timetabling optimization using tabu-variable neighborhood search based hyper-heuristic algorithm. *Procedia Computer Science*, *161*, 656-664.

Müller, T. (2005). Constraint-based timetabling.

Novák, V., Perfilieva, I., & Mockor, J. (2012). *Mathematical principles of fuzzy logic* (Vol. 517). Springer Science & Business Media.

Obit, J. H., Ouelhadj, D., Landa-Silva, D., Vun, T. K., & Alfred, R. (2011, December). Designing a multi-agent approach system for distributed course timetabling. In *2011 11th International Conference on Hybrid Intelligent Systems (HIS)* (pp. 103-108). IEEE.

Oude Vrielink, R. A., Jansen, E. A., Hans, E. W., & van Hillegersberg, J. (2019). Practices in timetabling in higher education institutions: a systematic review. *Annals of operations research*, *275*(1), 145-160.

Razak, H. A., Ibrahim, Z., & Hussin, N. M. (2010, March). Bipartite graph edge coloring approach to course timetabling. In 2010 International Conference on Information Retrieval & Knowledge Management (CAMP) (pp. 229-234). IEEE.

Redl, T. A. (2004). A study of university timetabling that blends graph coloring with the satisfaction of various essential and preferential conditions. Rice University.

Schaerf, A. (1999). A survey of automated timetabling. *Artificial intelligence review*, *13*(2), 87-127.

Thompson, J. M., & Dowsland, K. A. (1996). Variants of simulated annealing for the examination timetabling problem. *Annals of Operations research*, *63*(1), 105-128.

Vianna, D. S., Martins, C. B., Lima, T. J., Vianna, M. D. F. D., & Meza, E. B. M. (2020). Hybrid VNS-TS heuristics for university course timetabling problem. *Brazilian Journal of Operations & Production Management*, *17*(2), 1-20.

Welsh, D. J., & Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, *10*(1), 85-86.

Wren, A. (1995, August). Scheduling, timetabling and rostering—a special relationship?. In *International conference on the practice and theory of automated timetabling* (pp. 46-75). Springer, Berlin, Heidelberg.

Zhang, L., & Lau, S. (2005, November). Constructing university timetable using constraint satisfaction programming approach. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)* (Vol. 2, pp. 55-60). IEEE.