Universidad Autónoma de San Luis Potosí
Facultad de Ingeniería
Centro de Investigación y Estudios de Posgrado

Modelo de Selección de Algoritmos por Instancia para Resolver Problemas de Horarios Educativos

T E S I S

Que para obtener el grado de:

Doctorado en Ciencias de la Computación

Presenta:
Felipe de la Rosa Rivera

Asesor:
Dr. José Ignacio Núñez Varela

San Luis Potosí, S. L. P. Noviembre de 2020

Universidad Autónoma de San Luis Potosí
Facultad de Ingeniería
Centro de Investigación y Estudios de Posgrado

Per-instance Algorithm Selection Model for Solving Educational Timetabling Problems

T E S I S

Que para obtener el grado de:

Doctorado en Ciencias de la Computación

Presenta:
Felipe de la Rosa Rivera

Asesor:
Dr. José Ignacio Núñez Varela

San Luis Potosí, S. L. P. Noviembre de 2020

17 de septiembre de 2020

M.I. FELIPE DE LA ROSA RIVERA
P R E S E N T E.

En atención a su solicitud de Temario, presentada por el **Dr. José Ignacio Núñez Varela** Asesor de la Tesis que desarrollará Usted, con el objeto de obtener el Grado de **Doctor en Ciencias de la Computación**, me es grato comunicarle que en la Sesión del H. Consejo Técnico Consultivo celebrada el día 17 de septiembre del presente año, fue aprobado el Temario propuesto:

TEMARIO:

"Modelo de selección de algoritmos por instancia para resolver problemas de horarios educativos"

Introducción.
1. Problemas de horarios educativos.
2. Selección de algoritmos.
3. Generador de instancias.
4. Selección de características.
5. Portafolio de metaheurísticas.
6. Modelo de selección de algoritmos por instancia.

Conclusiones
Anexos
Referencias

"MODOS ET CUNCTARUM RERUM MENSURAS AUDEBO"

A T E N T A M E N T E

[illegible]

DR. EMILIO JORGE GONZÁLEZ GALVÁN
DIRECTOR.

UNIVERSIDAD AUTONOMA DE SAN LUIS POTOSI
FACULTAD DE INGENIERIA
DIRECCION

Copia. Archivo.
*etn.

"1945-2020: 75 años de formación de profesionales en la Facultad de Ingeniería"

UASLP
Universidad Autónoma de San Luis Potosí

FACULTAD DE INGENIERÍA

CENTRO DE INVESTIGACIÓN Y ESTUDIOS DE POSGRADO

UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ
FACULTAD DE INGENIERÍA

AUTORIZACIÓN DE IMPRESIÓN

29/	**OCT. /**	**2020**
DÍA	MES	AÑO

Secretaría General de la Facultad de Ingeniería
PRESENTE.

Nos permitimos hacer de su conocimiento que el (la) Sr(ita):

DE LA ROSA	**RIVERA**	**FELIPE**
APELLIDO PATERNO	*APELLIDO MATERNO*	*NOMBRE (S)*

ha efectuado a satisfacción las correcciones que se le indicaron durante la revisión conjunta a su trabajo de titulación, por lo cual estamos autorizando con esta forma la impresión del mismo.

Sin otro particular por el momento, protestamos las seguridades de nuestra atenta y distinguida consideración.

ATENTAMENTE

Dr. José Ignacio Núñez Varela	José Núñez V.
NOMBRE	FIRMA
Dr. José Carlos Ortiz Bayliss	[illegible]
NOMBRE	FIRMA
Dr. Juan Carlos Cuevas Tello	[illegible]
NOMBRE	FIRMA
Dr. Cesar Augusto Puente Montejano	[illegible]
NOMBRE	FIRMA
Dra. Sandra Edith Nava Muñoz	[illegible]
NOMBRE	FIRMA
NOMBRE	FIRMA

Resumen

El problema de horarios educativos basados en currículos (CB-CTT, por sus siglas en inglés) es un problema periódico en las instituciones de educación superior de todo el mundo, el cual ha demostrado ser del tipo NP-difícil (en inglés, NP-hard) en la práctica. Debido a su relevancia, este problema ha sido estudiado por comunidades de investigación que han propuesto una amplia variedad de métodos de solución dentro de diferentes contextos educativos. Sin embargo, ningún de ellos ha demostrado ser superior a los demás en todas las instancias del espacio del problema. En la última década, se han propuesto diversas estrategias como la hibridación y las hiper-heurísticas para integrar métodos de solución en diferentes problemas combinatorios. En contraste, esta tesis propone un enfoque alternativo para mejorar la calidad de la soluciones para instancias del problema CB-CTT: seleccionar, de un conjunto de meta-heurísticas, el método de solución con el mejor desempeño esperado para resolver una instancia dada. Para la construcción del modelo de selección, se analizan formalmente cuatro elementos del problema CB-CTT (i. e., instancias, características, algoritmos y medidas de desempeño) de acuerdo con el marco de meta-aprendizaje (en inglés, meta-learning). Como resultado, se realizan cuatro contribuciones relevantes al estado del arte: i) el diseño de un generador de instancias del problema CB-CTT, ii) la formulación de métricas para caracterizar instancias y soluciones, iii) la evaluación del desempeño de los algoritmos y metaheurísticas, iv) el diseño de un modelo de selección de algoritmos por instancia. Los resultados experimentales muestran que la implementación del modelo de selección propuesto produce mejores soluciones que las obtenidas con el algoritmo de mejor desempeño. Por tanto, el modelo propuesto puede aplicarse para integrar métodos de solución en el área de los problemas de horarios.

Abstract

The Curriculum-Based Course Timetabling (CB-CTT) is a real-world problem periodically solved in higher education institutions worldwide, which has proved to be NP-hard in practice. Because of its practical relevance, it has been studied by research communities that have proposed a wide variety of solution methods within different educational contexts. Still, none of them has proved to outperform the others across all instances of the overall problem space. In the last decade, various strategies have been proposed to integrate the strengths of existing solution methods in combinatorial problems such as hybridization and hyper-heuristics. Instead, to improve the quality of the solution for CB-CTT instances, this thesis proposes an alternate approach to automatically select, from a meta-heuristics portfolio, the solution method with the best-expected performance to solve a given CB-CTT problem instance. For the construction of the selection model, four elements of the CB-CTT problem (i.e., instances, features, algorithms, and performance measures) are formally analyzed following the meta-learning framework. As a result, four relevant contributions are made to the current state of the art: i) the design of a CB-CTT instance generator, ii) the formulation of metrics to describe both instances and their solutions, iii) the performance evaluation of algorithms and meta-heuristics, iv) the design of a per-instance algorithm selection model. The experimental results show that implementing the proposed selection model increases the quality of the solutions compared to a single-best solution method. Hence, it can be applied to integrate the strengths of solution methods in the timetabling domain in a useful manner.

A mí Señor y Salvador, Jesucristo. Toda la gloria es para ti.

Agradecimientos

Hace cuatro años inicié este proyecto de investigación; sin embargo, mi inquietud por el tema de programación de horarios surgió dieciocho años atrás, mientras cursaba la asignatura de investigación de operaciones. Durante este tiempo, muchas personas han contribuido al desarrollo de esta tesis en diferentes formas.

Agradezco al Dr. José Ignacio Núñez Varela por apoyarme en el proceso de ingreso al doctorado, y por arriesgarse a dirigir este proyecto de investigación. Gracias por tus preguntas punzantes y por tus interminables listas de correciones en todos mis textos.

Agradezco al grupo de investigación del área de ciencias de la computación, en especial al Dr. César Augusto Puente Montejano y a la Dra. Sandra Edith Nava Muñoz, miembros de mi comité de tesis. Sus evaluaciones y puntos de vista fueron sumamente enriquecedores durante todo este proceso.

Gracias al Dr. Juan Carlos Cuevas Tello, por introducirme al fascinante campo del aprendizaje automático y al interesante mundo del basquetbol. También al Dr. José Carlos Ortiz Bayliss, por darle a mi investigación ese impulso final que necesitaba.

A mis compañeros de doctorado, muchas gracias por compartir su tiempo conmigo. Cuatro años parecían mucho tiempo, pero transcurrieron muy rápido. Gracias por mantener la cafetera y la tetera del LCD siempre encendida.

Familia, las palabras no son suficientes para agradecerles. Lo que he logrado y lograré es debido a su amor incondicional. Siempre están presentes en todo lo que hago, es un honor tenerlos en mi vida.

Finalmente, a todos mis amigos en este país y en el extranjero, gracias por celebrar mis triunfos y llorar mis fracasos. Agradezco a Dios por hacer coincidir nuestros caminos.

Índice General

Contents

List of Figures

List of Tables

Introduction

Human resources have a significant role in service-oriented organizations. Therefore, managers and supervisors have developed diverse methodologies to optimize the scheduling of tasks assigned to employees [3] —a planning problem known as timetabling. The elements considered for timetabling varies across different fields of application. Timetabling in educational institutions differs from other related problems as it considers not only the scheduling of human resources (i.e., professors) but also the scheduling of customers (i.e., students). Hence, it is concerned about finding a plan to ensure that both professors and students can attend to a set of assigned academic events without clashes (i.e., overlapping times).

Educational timetabling is a combinatorial optimization problem that institutions solve frequently. At its basic concept, it involves the assignment of resources and time slots to a defined set of academic events (i.e., classes), according to a series of hard (mandatory) and soft (optional) constraints. However, because of the diverse conditions that defines educational timetabling, it has been divided into three related problems [4]:

Examination Timetabling (Ex-TT): Schedules exams sessions of a given duration into a number of periods while satisfying a number of hard constraints.

Post Enrollment Course Timetabling (PE-CTT): Generates timetables in such a way that all students can attend to all the classes on which they have previously enrolled.

Curriculum-Based Course Timetabling (CB-CTT): Schedules predefined sets of classes in which groups of students are to be enrolled.

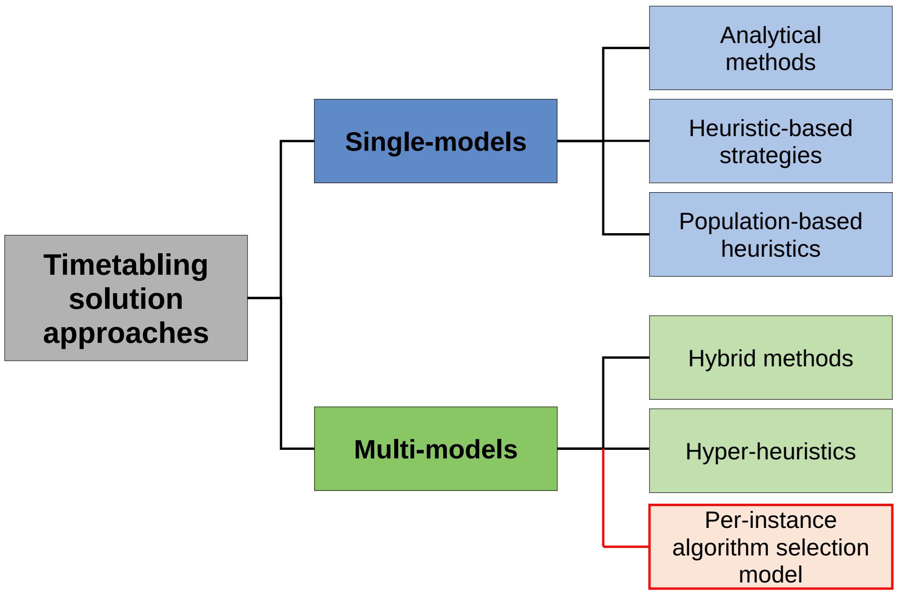

Timetabling solution approaches
Single-models
Multi-models
Analytical methods
Heuristic-based strategies
Population-based heuristics
Hybrid methods
Hyper-heuristics
Per-instance algorithm selection model

Figure 1: Timetabling solution approaches in the current state of the art.

Because of their educational models, many universities must solve the CB-CTT problem to produce feasible timetables that maximize both professor/student satisfaction and resource usage. However, as this problem has proved to be NP-hard in practice [5], finding an optimal solution is difficult to accomplish.

Since its first formal definition [6], a wide range of algorithms and heuristics have been proposed to find practical solutions for instances (i.e., concrete formulations) of educational timetabling problems. According to the number of methods applied in the solution process, they can be grouped into two broad approaches: *single-models* and *multi-models* (as shown in Figure 1):

On the one hand, the single-model approach refers to *analytical methods* (e.g., linear programming), *heuristic-based strategies* (e.g., tabu search), and *population-based heuristics* (e.g., genetic algorithms), that rely on the implementation of a single strategy to solve entire timetabling instances. On the other hand, the multi-model approach refers to strategies that solve timetabling instances by different combinations of solution methods. In the current state of the art, two prevalent multi-model strategies are: *i) hybrid-methods*, that divide instances into sub-problems sequentially solved by at least two different single-models, and *ii) hyper-heuristics*, that automatically combine a set of low-order heuristics to solve distinct stages of timetabling instances.

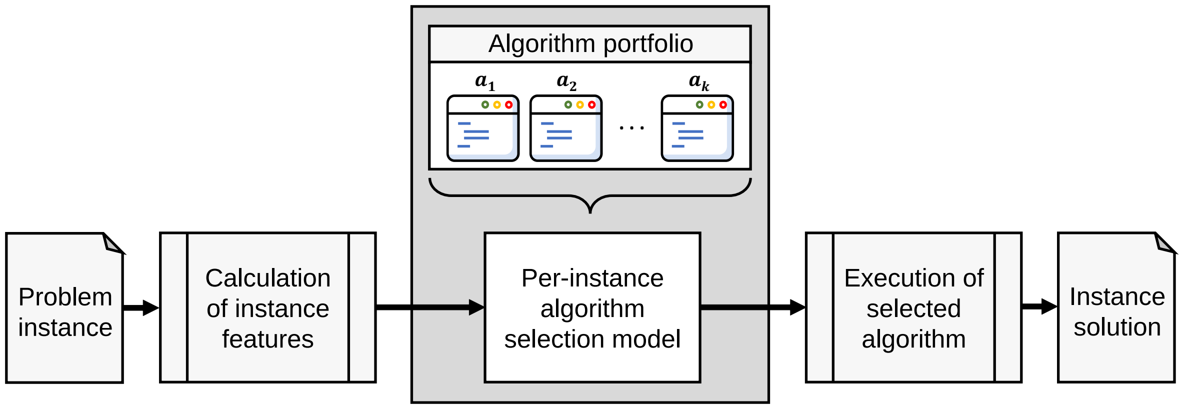

Algorithm portfolio
a_1
a_2
a_k
Problem instance
Calculation of instance features
Per-instance algorithm selection model
Execution of selected algorithm
Instance solution

Figure 2: General process used to solve instances applying per-instance algorithm selection models.

A third multi-model strategy that has not been applied for the solution of CB-CTT instances is the construction of a *Per-instance Algorithm Selection Model*[1] (per-instance ASM). Per-instance ASMs differ from other multi-model approaches in that they do not combine solution methods to solve sub-problems of instances but select on a per-instance basis the apparent best method to solve entire instances.

Figure 2 presents the general process used to solve problem instances applying per-instance ASMs. As shown, the process starts with a problem instance that is described by the calculation of relevant features. Then, these features are employed by the per-instance ASM to predict the algorithm that is likely to perform the best to solve the instance. Finally, the algorithm selected from the portfolio is executed to get the solution.

Due to the success of algorithm portfolios in international competitions, per-instance algorithm selection has become a relevant solution approach. Starting with SATzilla [7], constructed to solve instances of the propositional satisfiability problem (SAT), this multi-model approach has proved to be effective on diverse combinatorial problems, such as multi-mode resource-constrained project scheduling [8] and maximum satisfiability [9]. Selection models have proved to be robust solution approaches, able to integrate the complementary strengths of algorithms for the solution of hard problems.

[1]In the current state of the art, there is no clear consensus about the term to refer to the computational tools for algorithm selection. In this thesis, we employ the word model, commonly used in the publications of the international research group on configuration and selection of algorithms (COSEAL).

However, to be useful, they require to be suited to the specific conditions that characterize different problem domains.

Because of its proven effectiveness in related combinatorial fields, this thesis focuses on the development of a per-instance algorithm selection model for the solution of Curriculum-Based Course Timetabling (CB-CTT) instances. Therefore, it is mainly involved with the following research fields: combinatorics, optimization, and machine learning.

The rest of this introduction proceeds as follows. First, a brief review of the findings that motivated this thesis are discussed. Second, the research goals are summarized. Third, the challenges and research questions are defined. Fourth, the main contributions of the thesis are listed. Finally, the global structure of the thesis is outlined.

Motivation

Due to its NP-hard nature, many heuristic-based approaches have been proposed to get good solutions for CB-CTT problem instances in practical applications. Recent surveys [10–12] show an increment in the number and diversity of the proposed solution methods. However, as most of them are designed to suit the constraints of particular educational institutions, they can be considered competent only within small CB-CTT problem sub-spaces.

At this moment, the available methods (i.e., algorithms and heuristics) in optimization and combinatorics are diverse enough to produce competent solutions in a wide range of real applications [13]. However, as the problem subspaces in which these methods are competent are often unknown, selecting the best solution approach for a particular instance is a complex computational problem, known as *per-instance algorithm selection*.

As stated by the *No Free Lunch Theorem* [14], there is no single algorithm able to produce the best solution for all the instances of a problem domain. Therefore, selecting the best algorithm to solve a particular instance (or sub-space of instances) has become a crucial factor to improve the quality of the solutions.

Per-instance algorithm selection is a complex task that requires both: *i)* professional experience in the problem domain, and *ii)* knowledge about machine learning approaches [15] . Therefore, despite its relevance in the solution process of combinatorial problems, in practice, it is often performed based on empirical approaches rather than with formal analysis.

In the context of educational timetabling, per-instance algorithm selection is a problem not formally addressed in the current state of the art. Therefore, to analyze its suitability for the solution of CB-CTT instances, this thesis describes the construction of a per-instance algorithm selection model following the *meta-learning framework* proposed by Brazdil et al [2]. Within the context of algorithm selection, the goal of meta-learning is to apply machine learning approaches to generate a selection mapping between the relevant features of a problem domain and the performance of a set of solution methods. Due to its flexibility, which allows the implementation of varying machine learning approaches, this framework has proved to be useful across different problem domains in the construction of accurate selections models, without adding significant computational effort to the solution process of the instances [16].

It is important to notice that two problems will be analyzed throughout this thesis: *i)* the CB-CTT problem (properties and solution methods), and the *ii)* per-instance algorithm selection problem (data analysis and machine-learning methods).

Research Questions

This research is based on the meta-learning framework, which have proved to be competitive on solving optimization problems in different application fields [17, 18]. However, the fact that most of the current meta-learning research focuses on describing its implementation within forecasting areas and continuous-domain problems, leads to two important research questions addressed in this thesis:

- How can machine learning approaches be used to accurately relate the relevant features of CB-CTT instances to the performance of algorithms?

- How can a per-instance algorithm selection model be generated to apply the best algorithm to solve a given CB-CTT instance?

Research Goals

General

To construct a *per-instance algorithm selection model* based on the meta-learning framework to select from a portfolio of algorithms the one that generates the best solution for a particular CB-CTT instance.

Specific

1. To select the proper *data format* to represent CB-CTT instances.
2. To generate a representative *dataset* of CB-CTT instances.
3. To formulate a set of *relevant features* to characterize the different CB-CTT problem sub-spaces.
4. To select a set of algorithms to build the *algorithm portfolio.*
5. To define a set of *performance measures* to describe the solving effectiveness of the algorithms over a representative number of CB-CTT instances.
6. To build a *machine learning*-based model to perform the *per-instance algorithm selection* task.

Thesis Contributions

According to the research goals described above, the following is the list of contributions from this thesis.

- The design of a parameterized CB-CTT *instance generator* to increase the size of benchmarking datasets which can be used to analyze the performance of future solving approaches.
- The formulation of a set of *complexity metrics* able to distinguish CB-CTT instance sub-spaces that share similar solving difficulty.
- The *performance evaluation* of a set solution methods proposed to solve CB-CTT instances.
- A *per-instance algorithm selection model* (based on the meta-learning framework), constructed to predict from a portfolio of algorithms, the one with the best-expected performance to solve a given CB-CTT instance.

Research Methodology

To organize all activities related to this research project, we followed the methodology shown in Figure 3. As observed, it consists of three main phases summarized next.

Research definition: It consists of the activities for delimiting the scope and goals of our research. It is involved with: i) *problem understanding*, a comprehensive analysis of the elements associated with educational timetabling and algorithm selection; ii) *literature review*, a broad revision of the current state of the art; and iii) *research opportunities*, a formal definition of the research gaps to be addressed in our work.

Model construction: It focuses on performing the activities required to construct our per-instance algorithm selection model. It includes: *scope of the model*, defining the goal and general operation of the model; ii) *components development*, creating the elements required for the model; and iii) *components integration*, linking the elements to construct the model.

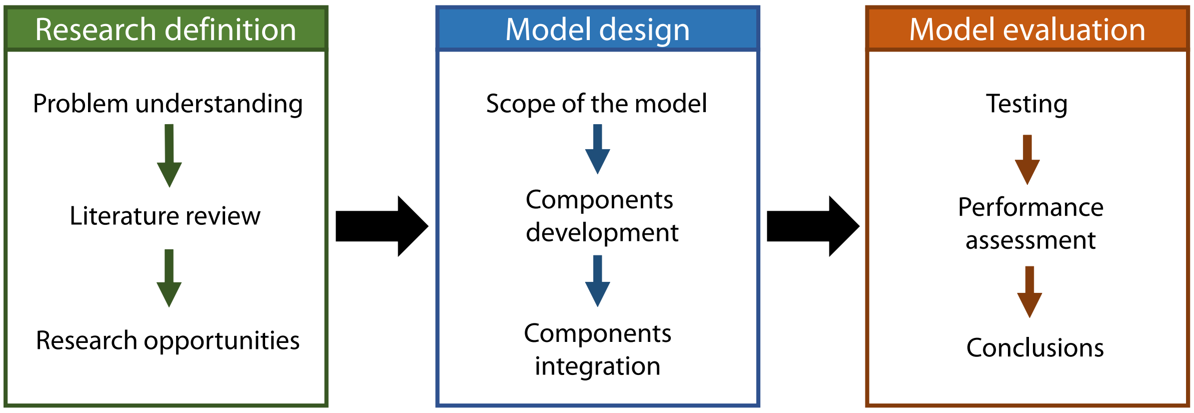


Figure 3: Research methodology.

Model evaluation: It concentrates on evaluating the performance of the per-instance algorithm selection model. It comprises: i) *testing*, defining computational experiments to evaluate the model; ii) *performance assessment*, selecting and calculating performance measures; and iii) *conclusions*, interpreting the performance of the model to identify its strengths and weaknesses.

Thesis Outline

The rest of this thesis is structured according to the construction process of the per-instance algorithm selection model. Next, a brief summary of the subsequent chapters are presented to the reader:

Chapter 1: This chapter presents a general description of the CB-CTT problem (formulations and data formats). The set of solution methods available in the current state of the art are also described.

Chapter 2: This chapter describes the conceptual foundations of algorithm selection. It presents a survey of recent approaches that have been applied to build algorithm selection models for combinatorial optimization problems and introduces the meta-learning framework employed in this research.

Chapter 3: This chapter presents the design of the instance generator employed to create the dataset for the per-instance algorithm selection model. It describes the structure of the instances, the parameters to be defined, and introduces the concept of *empirical hardness* (difficulty of solution).

Chapter 4: This chapter presents the sets of features defined to characterize the generated instances and describes the feature selection process applied to evaluate their relevance. As a result, it also provides an experimental interpretation of the empirical hardness of CB-CTT instances.

Chapter 5: This chapter describes the set of solution methods selected to conform the algorithm portfolio. It presents their performance and discusses their variation across the defined instance space.

Chapter 6: This chapter contrasts the machine learning-based approaches commonly applied to solve algorithm selection problems. Then, it presents the per-instance algorithm selection model, its experimental setup, and the results related to its performance.

Conclusions: This chapter summarizes the main findings obtained from the construction of the per-instance algorithm selection model. These findings are interpreted in the context of previous researches. The theoretical, and practical relevant implications of the findings are also considered. Finally, the chapter addresses the strengths and limitations of the study and proposes areas for future research.

Chapter 1

Curriculum-Based Course Timetabling

In this section, the mathematical foundations of the Curriculum-Based Course Timetabling problem are introduced. Next, the standardized formats proposed to represent instances are discussed. Finally, a literature review of the solution methods proposed to find good solutions for educational timetabling instances is presented.

1.1 Mathematical Formulation

Curriculum-Based Course Timetabling (CB-CTT) is a relevant problem within the educational timetabling field, which has been formulated and represented in different ways to suit the particular needs of diverse educational contexts. However, to encourage research collaboration, in 2007 was defined in a standardized manner for the Second International Timetabling Competition (ITC-2007).

As defined for the ITC-2007, the CB-CTT problem can be described as the weekly scheduling of lectures for several classes (i.e., academic events) given a number of teachers, rooms, and time periods, in which the conflicts between classes are set according to the curricula set by the university [19]. The concept of *curriculum* is particularly relevant because it differentiates the CB-CTT problem from two related timetabling problems: Examination Timetabling (Ex-TT) and Post Enrolment Course Timetabling (PE-CTT).

In this context, *curriculum* refers to a set of classes defined to be taken together by a group of students.

According to its standard definition, the elements commonly required to define a CB-CTT instance (i.e., a concrete formulation of the CB-CTT problem) are:

- A set of teaching days D (where a day $d \in D$); and a set of teaching intervals per day I_d, (where an interval $i_d \in I_d$).
- A set of time slots TS composed of a day and a teaching interval $\langle d, i_d \rangle$.
- A set of teachers T, where each teacher $t \in T$ has a limited weekly workload $w_t \in \mathbb{N}$.
- A set of rooms R, where each room $r \in R$ has a limited weekly availability $a_r \in \mathbb{N}$.
- A set of classes C, where each class $c \in C$ has a total weekly duration $d_c \in \mathbb{N}$, which can be split in different lectures (i.e., meetings). A class requires at least three resources: a teacher, a room, and a group of students.
- A set of curricula Q, where a curriculum $q \in Q$ is a group of classes that shares a group of students. Therefore, this group of classes cannot be scheduled at the same times.

Thus, in general terms, the solution to a CB-CTT instance consists on assigning the required time slots and resources to all lectures, according to a set of constraints. There are two types of constraints: *i)* hard constraints, which are mandatory (i.e., they must be fulfilled in order to produce a feasible solution); ii) soft constraints, which are optional (i.e., their fulfillment only increases the quality of a feasible solution.)

To represent the elements just described, consider the example illustrated in Figure 1.1. The given example presents the timetable of a *curriculum* in the field of social sciences. This *curriculum* includes two *classes* from different *courses*, "Politics", and "Ethics". The *class*, "Politics", with a *weekly duration* of three *teaching intervals*, has

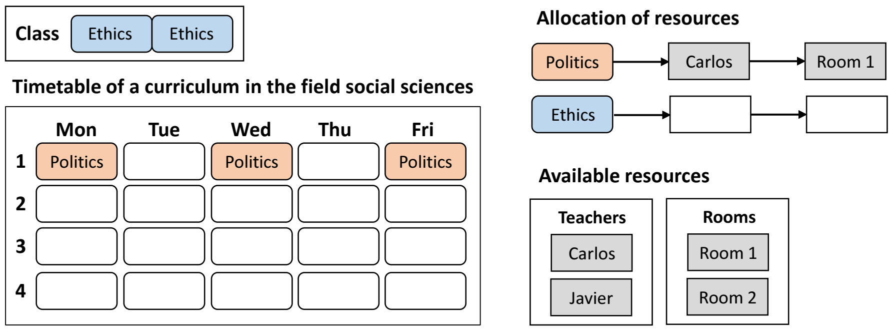


Figure 1.1: Illustration of the elements commonly involved in the mathematical formulation of a CB-CTT instance.

already been split into three *lectures* and scheduled [1] in a timetable of twenty *time slots* (four *teaching intervals* per day). Besides, their required resources (a *teacher* and a *room*) have been allocated to its lectures. The *class* "Ethics", with a *weekly duration* of two *teaching intervals*, has not assigned times slots or resources. Depending on the constraints defined, it can be scheduled in the timetable as a single lecture (with a duration of two *teaching intervals*) or as two lectures (with a duration of one *teaching interval*). As shown in the figure, two teachers and two rooms can be allocated to its lecture(s), taking into account the resources already allocated to the class "Politics" to avoid *clashes of resources*. Thus, if "Room 2" is allocated to the *class* "Ethics", it cannot be scheduled in the *time slots*: *Mon-1*, *Wed-1*, and *Fri-1*, as doing so implies a violation of a constraint considered as a *hard* in timetabling instances.

As explained by Bettinelli et al. [20], the CB-CTT is a combinatorial problem inherently related to the graph coloring problem, which is a well-known NP-hard computational problem. If we represent the CB-CTT problem using a graph in which each vertex represents a lecture and each edge a pair of lectures that cannot be simultaneously scheduled because they share a resource (i.e., a teacher, a room, or a curriculum). Then, suppose we consider a different color for each time slot. In that case, the core problem of CB-CTT is assigning one color to each vertex so that adjacent vertices are assigned different colors. The complexity of this problem is relevant because NP-hard problems are not solvable by deterministic algorithms in polynomial time; therefore, they require

[1] Within the current literature, there are not standard terms to distinguish between the assignment of time slots and resources to classes. In this thesis, the term *schedule* refers to the assignment of time slots, and allocation, to the assignment of resources.

```
Courses: <CourseID> <Teacher> <# Lectures> <MinWorkingDays> <# Students> <Double Lectures>
Rooms: <RoomID> <Capacity> <Site>
Curricula: <Curriculum Id> <# Courses> <CourseID> ... <CourseID>
Unavailability_Constraints: <CourseID> <Day> <Day_period>
Room_Constraints: <CourseID> <RoomID>
```

Figure 1.2: Structure of the Extended Course Timetabling format *ECTT*.

the formulation of alternative solution approaches to find relative good solutions.

1.2 Timetabling Data Formats

The solution of CB-CTT instances requires a proper mathematical formulation of variables and constraints, but also an appropriate representation of these elements in a format that can be used by a computer.

In the literature, at least two data formats have been used to represent CB-CTT problems: the Extended Course Timetabling format (*ECTT*), and the XML High School Timetabling format (*XHSTT*).

1.2.1 Extended Course Timetabling Format

Bonutti et al. [21] proposed the ECTT format to extend the representation capacity of the *course timetabling data format* (*CTT*), developed to handle the instances of the ITC 2007 [19]. The ECTT format represents the data of timetabling instances according to the following five sections (see Figure 1.2):

- **Courses**: Classes to be scheduled, that are labeled with an ID and defined by the following five fields: teacher, number of lectures, minimum number of days in which lectures must be given, maximum number of students, and specification of double lectures.
- **Rooms**: Set of available classrooms, labeled with an ID and described in terms of their capacity and location.
- **Curricula**: Events shared by a group of students, labeled with an ID and described by the number and ID of the courses, which belong to the curricula.

```
<HighSchoolTimeTableArchive Id= >
    <Instances>
        <Instance Id= >
            <Metadata>
                <Name></Name>
                <Contributor></Contributor>
                <Date></Date>
                <Country></Country>
            </Metadata>
            <Times>
                <TimeGroups>
                        ...
                </TimeGroups>
                <Time Id= >
                    ...
                <\Time>
            </Times>
            <Resources>
                <ResourceTypes>
                        ...
                </ResourceTypes>
                <ResourceGroups>
                        ...
                </ResourceGroups>
                <Resource Id= >
                        ...
                </Resource>
            </Resources>
            <Events>
                <EventGroups>
                        ...
                </EventGroups>
                <Event Id= >
                    ...
                </Event>
            </Events>
            <Constraints>
                    ...
            </Constraints>
        <\Instance>
    <\Instances>
</HighSchoolTimeTableArchive>
```

Figure 1.3: General structure of the XML schema that defines the High School Timetabling format *XHSTT*.

- **Unavailability constraints**: Set of time slots in which courses cannot be allocated.

- **Room constraints**: Classrooms in which courses must be allocated.

1.2.2 XML High School Timetabling Format

The XHSTT data format was selected by the Euro Working Group on Automated Timetabling (Euro WATT) as the standard to be used on the ITC 2011 [22]. The structure of this format is defined by an XML schema (illustrated in Figure 1.3), composed of four entities [23]:

- **Times**: Set of possible time slots in which events can be scheduled. These time slots are often grouped into *time groups* (e.g., days or weeks).

- **Resources**: Set of available resources that can be assigned to the events. Each resource belongs to a specific resource type (e.g. teacher, rooms, etc.), and can be grouped into *resource groups* for administration purposes.

- **Events**: Set of classes to be scheduled. Each event has a duration, which represents the amount of time slots that must be scheduled, and a demand of a set of resources. Events can also be grouped into *event groups*.

- **Constraints**: Set of constraints that must be fulfilled during the scheduling of events. Each constraint can be defined as hard or soft, and it is evaluated according to two parameters: *i)* a cost that indicates the *penalty value* of a single violation and *ii)* a *cost function* that defines how the penalty values are added to the objective function to be minimized. Table 1.1 summarizes the 16 types of constraints available in this format.

1.2.3 Comparison of CB-CTT Data Formats

The two data formats described above differ in their structure and in their representation capability. On the one hand, the ECTT format operates based on a text file which encodes instances according to a fixed data order. In addition to the basic resource allocation and task scheduling constraints, it allows the representation of CB-CTT problems that are limited by only two types of constraints: unavailable times of courses and allocation of rooms. On the other hand, the XHSTT format structures data according to an XML schema, which relates four entities to define scheduling problems based on a set of sixteen types of constraints that can be applied to represent a diverse set of real-life conditions.

For this thesis, the XHSTT format was selected for two main reasons: i) its organized structure that allows syntactical validation of the instances, and ii) its broader capacity to represent real-life conditions that cannot be encoded using the ECTT format (such

Table 1.1: Description of the 16 types of constraints available in the XHSTT format.

	Name	Acronym	Description
1	Assign Resource Constraint	ARC	Requires that all the resources required by the classes be assigned
2	Assign Time Constraint	ATC	Requires the assignment of time slots to classes
3	Split Events Constraint	SEC	Limits the number of lectures that can be derived from a given class and their duration
4	Distribute Split Events Constraint	DSEC	Limits the number of lectures of a particular duration that can be derived from a given class
5	Prefer Resources Constraint	PRC	Specifies that some resources are preferred to be assigned to some classes
6	Prefer Times Constraint	PTC	Specifies that some time slots are preferred to be assigned to some classes
7	Avoid Split Assignments Constraint	ASAC	Specifies that the resources assigned to all non-consecutive lectures derived from a given class must not vary
8	Spread Events Constraint	SPEC	Specifies how non-consecutive lectures should be spread out in different time slots
9	Link Events Constraint	LEC	Specifies that certain set of classes must be assigned the same time slots
10	Order Events Constraint	OEC	Specifies that the time slots of two classes must be assigned in such a way that the first class ends before the second begins
11	Avoid Clashes Constraint	ACC	Specifies that certain resources must have no clashes; that is, they should not be assigned to two or more classes simultaneously
12	Avoid Unavailable Times Constraint	AUTC	Specifies that some resources are unavailable to be assigned to any class at certain times
13	Limit Idle Times Constraint	LITC	Limits the number of time slots that resources are idle within a *time group* (e.g., limit the daily idle time slots of teachers)
14	Cluster Busy Times Constraint	CBTC	Limits the number of *time groups* a resource is busy (e.g., limit the number of days a teacher gives lectures per week)
15	Limit Busy Times Constraint	LBTC	Limits the number of time slots within a *time group* that a resource is busy (e.g., limit the daily number of time slots a teacher gives lectures)
16	Limit Workload Constraint	LWC	Limits the total workload assigned to a resource

as, allocating rooms to courses and defining working shifts for teachers). An additional benefit of using this format is its markup language, which employs human-readable tags to define the elements of an instance; thus, it is easier to interpret than plain-text formats.

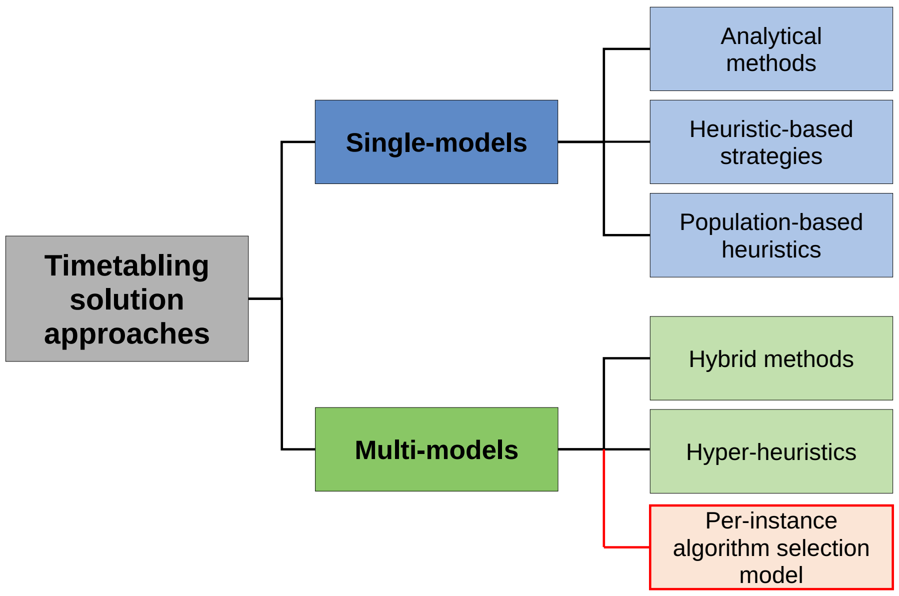

Timetabling solution approaches
Single-models
Multi-models
Analytical methods
Heuristic-based strategies
Population-based heuristics
Hybrid methods
Hyper-heuristics
Per-instance algorithm selection model

Figure 1.4: Timetabling solution approaches in the current state of the art.

1.3 Timetabling Solution Methods

The diversity of constraints defined by educational institutions worldwide has led to a broad range of formulations and solution methods for educational timetabling, none of which —given the NP-hard nature of the problem— guarantee finding an optimal solution for an instance. Within the current state of the art, the set of algorithms and heuristics is extensive, making the selection of solution approaches a problem on its own.

The first formal formulation of school timetabling, proposed by Gotlieb in 1963 [24], considered only three sets of related variables: teachers, classrooms, and time slots. However, because of the number of practical applications reported in the literature, periodical surveys are performed to keep researchers up-to-date in this problem domain. Next, a brief review of common methods applied for the solution of timetabling instances is presented according to the categories shown in Figure 1.4[2]

1.3.1 Single-Model Approaches

Single-model approaches encompass solution methods of different nature applied to solve *entire* timetabling instances. According to their mathematical foundation, they can be

[2]Same as Figure 1 presented in the Introduction, but reproduced here for convenience.

categorized into: *analytical methods*, *heuristic-based strategies*, and *population based-heuristics*.

1.3.1.1 Analytical Methods

Analytical methods are formulations of timetabling instances based on mathematical methods that iteratively search the optimal value of objective functions, defined to calculate the number and cost of constraint violations. Some common methods of this type are: Graph coloring, Integer programming, and Constraint satisfaction programming.

Graph Coloring

A few years after Gotlieb proposed the first timetabling formulation, Welsh & Powell [25] modeled timetabling instances as graph coloring problems obtaining good results. However, due to its limited modeling capabilities (which allows the representation of a short set of constraints), its application has been limited to solve basic instances that only require that events and resources not be overlapped.

Graph coloring uses a sequential approach to assign activities, one by one, starting with those more difficult to be assigned. However, it becomes impractical as the number of nodes to solve increases.

Linear Programming

Because of its common usage among researchers, linear programming was used in the first studies related to the automatic generation of timetables in 1969 [26].

In its basic form, the linear programming model considers four basic entities: times, events, resources, and constraints. Each event requires allocating a set of resources that must meet a specific set of requirements of time and functionality. These requirements link them with a set of constraints that must be satisfied to minimize a cost function that adds up the penalty values of hard and soft constraint violations.

The main advantage of modeling timetabling instances as linear programming problems is that they can be solved using conventional software applications that do not require advanced computational knowledge. However, it still requires a deep understanding of principles for mathematical modeling.

Constraint Satisfaction Programming

Constraint satisfaction programming is a type of mathematical modeling that considers constraints as boundaries, which restricts the values considered as feasible to a set of decision variables. According to Hentenryck & Saraswat [27], by following this reasoning, a problem can be modeled as a function of three elements $CSP = (X, D, C)$ where: X is a finite set of variables $X = x_1, x_2, ..., x_n$; D is a finite set of domain values, $D = d_1 \times d_2 \times, ..., \times d_n$ that the variables can take; and, C is a finite set of constraints, $C = c_1, c_2, ..., c_m$ coming from logical relationships between the subsets of variables. The final solution is obtained by assigning feasible values to each variable while satisfying the entire set of defined constraints. An example of the implementation of this method can be found in the timetabling study case solved by Zhang & Lau [28].

1.3.1.2 Heuristic-Based Strategies

Heuristic-based strategies work by first generating an initial solution for a problem instance. Then, iteratively modifying such an initial solution until certain criteria are met. Some common strategies that follow this solution approach are: Tabu search, Simulated annealing, Hill climbing, and Variable neighborhood search.

Tabu Search

Tabu search was proposed by Glover in 1986 [29], and it emerges to provide local search algorithms with some type of "intelligence", to explore the solution space. The method starts with an initial solution that is improved trough iterative movements to neighbor points with better solution values. Each movement in the solution space is included in a *tabu list*, a record of all the previously considered neighbors, which is updated to avoid analyzing the same solutions many times. Searching movements in the solution space conclude when a termination condition is met. The design and implementation of a software package based on this solution method can be found in the research work of Alvarez et al. [30].

Simulated Annealing

Simulated annealing is a method proposed by Kirpatrick et al. [31] to find the global minimum of a cost function with several local minima. Its basic idea is to simulate the heating process and the temperature reduction of a material during the tempering process. Each iteration selects a random point according to a probability distribution. The algorithm analyzes new points that increases the value of the objective function, but also (with a certain probability) points that diminish its value, thus avoiding being trapped in local minima.

The process starts with the generation of a random initial solution, which is iteratively replaced based on a random probabilistic function, guided by the temperature progression formula: $T_{i+1} = T_i \times \beta$. It is important to point out that the temperature change β and the initial temperature value T are relevant parameters usually defined based on human experience. A representative example of this method can be found in the work of Aycan & Ayan [32], in which they compare and combine different searching strategies using simulated annealing.

Hill Climbing

Hill climbing method relies on the analogy of considering an objective function as the highest peak of a mountain that is to be climbed. Thus, at each *step* (iteration), a "climber" must choose the move that "leads him uphill" in the least possible time. At each iteration of the search, a *step* is given only if it improves the value of the objective function, or it reduces the distance to feasibility.

One of the most basic implementations of this method is the *Steepest Hill Climbing* (SHC). SHC selects from the defined neighborhood of the current solution, the *step* that better optimizes the objective function, and accepts the *step* only if it is an improving one. Its stopping criteria is given by a maximum number of *steps* that must be performed. The effectiveness of hill climbing and other local search methods is analyzed in the work of Schaerf & Di Gaspero [33].

Variable Neighborhood Search

The variable neighborhood search (VNS) was proposed by Mladenović and Hansen [34] as a way to improve the performance of local search methods by proceeding to a systematic change of neighborhoods within a defined solution space. According to its authors, the main advantage of the VNS method is that "*it does not follow a trajectory, but explores increasingly distant neighborhoods of the current incumbent solution, and jumps from there to a new one if and only if an improvement was made.*"

The general operation of the VNS consists of two phases: *i)* a perturbation phase, to explore as much as possible dissimilar solutions in different neighborhoods, and *ii)* a descent phase, to find the local optimum in each explored neighborhood. An example of the operation of this solution method can be found in the research work of Abdullah et al. [35].

1.3.1.3 Population-based heuristics

Population-based heuristics work from a *set* of initial solutions (initial population), that after each iteration is analyzed by a selection mechanism to determine the best current set of solutions (best individuals). Subsequently, the overall population is manipulated (depending on the type of methodology used) to replace the current individuals with better ones, until, with each new cycle, the desired solution is reached. Some common heuristics are: Genetic algorithms, Ant colony optimization, and Memetic algorithms.

Genetic Algorithms

Genetic algorithms, proposed by Holland [36], simulate the processes of evolution of the species, through the mechanism of natural selection, to optimize the value of mathematical models, expressed by a *fitness* (objective) function. According to Babaei et al. [37], the common basic steps that must be considered when implementing a genetic algorithm are the following.

- **Selection**: Evaluates the fitness of individuals to determine which solutions deserve to be preserved to be reproduced and which must be discarded.
- **Regeneration**: Applies reproduction operations, crossover and mutation, on parents to produce children (new individuals).
- **Replacement**: Replaces the least-fit individuals of the current population with new individuals.

These steps are repeated until a termination condition (e.g. a number of generations, or an objective value) is reached. As an example of the implementation of genetic algorithms in the timetabling field we refer to the reader to the research work of Alsmadi et al. [38].

Ant Colony Optimization

Dorigo and his team were the first to introduce the first optimization algorithm based on ant colonies in the early 1990s [39]. By mimicking the natural ability of ants for searching food, based on its social communication via pheromones (known as stigmergy), they were able to solve optimization problems.

In its most basic form, ant colony optimization methods generates artificial ants which look for the shortest path between the nest and its source of food. The process they follow implies marking its trail with pheromones that evaporate as time passes. After some iterations, it is possible to identify the routes with a higher concentration of artificial pheromones, that is, the ones which were most frequently selected by the ants. Using this approach allows to define routes with different level of attractiveness for the next ants, thus, making evident the optimal solutions that must be selected. Nothegger et al. [40] applied this solution method for solving timetabling instances of the ITC-2007 obtaining good results.

Memetic Algorithms

The concept of Memetic algorithms was introduced in the late 1980s [41] to denote a family of heuristics that are focused on the hybridization of *heuristic scanning* and *local search* methods. The *heuristic scanning* works by looking for good potential regions within the solution space, and the *local search* methods explore those regions by iteratively proving different combinations of values for the problem variables.

The main difference between memetic and genetic algorithms is the concept of *meme*, which is the basic element used for evolving solutions. A *meme* differs from a gene because when transmitted to future generations, each individual adapts it, while the gene is transmitted without any changes. Therefore, memetic algorithms can be considered as a type of cultural algorithm, which uses a process of dual inheritance to micro-evolutionary and macro-evolutionary level. At a micro-evolutionary level, cultural algorithms consider the transmission of behaviors or traits between individuals in a population, and at a macro-evolutionary level, the formation of generalized beliefs based upon individual experiences. An implementation of this type of algorithms for the solving timetabling instances in universities can be found in the research work of Jat & Jang [42].

1.3.2 Multi-model

So far, a brief review of single-model approaches applied to solve entire educational timetabling instances has been presented; now, the strategies employed by multi-model approaches are described (see Figure 1.4). Multi-model approaches include knowledge areas that seek to exploit the main advantages of the analytical and heuristic methods (and others, like combinatorial optimization and fuzzy logic) by automatically combining them to intervene at different stages of the timetabling solution process. Depending on their nature, they can be categorized into hybrid methods and hyper-heuristics.

1.3.2.1 Hybrid Methods

According to Jourdan et al. [43], when facing NP-hard problems, a category to which most of the combinatorial optimization problems belong, researchers usually select one of two approaches, depending on the size of the problem at hand. For small instances, they implement exact methods known to be time expensive. However, when instances are too large, they often solve them by using stochastic methods (i.e., heuristics), which usually entails a higher variation in the quality of the solutions. An alternative approach, frequently applied to overcome the limitations of both solution approaches, is hybridization.

Hybridization refers to any combination of methods (algorithms and heuristics) that are sequentially applied to solve a problem instance —striving to reduce the negative effect of their limitations. For example, Kotusch [44] employs a hybrid method that generates initial timetables using a sequential heuristic; then, this initial timetable is later improved using a simulated-annealing heuristic.

Despite its popularity for solving combinatorial optimization problems, three relevant factors could limit the effectiveness of hybrid approaches. The first factor is the large possible number of combinations between the single-model approaches to be tested to find the best hybrid model to solve a particular instance space (i.e., a similar group of instances). The second factor is the potentially hard process required to link (transfer information) algorithms of different nature, which might require employing distinct mathematical formulations at each stage of the solution process. The third factor is the *static* nature of the hybrid approach; once the sequence of the solution process has been set, it cannot be easily adjusted to solve other instance spaces.

1.3.2.2 Hyper-heuristics

The concept of hyper-heuristics was first introduced by Cowling et al. [45] to refer to strategies designed to select heuristics to solve combinatorial problems. Now, the term is used to refer to any approach designed to select, combine or adapt low-level heuristics to construct solution methods —known as high-level heuristics— for solving problem instances.

The main goal of hyper-heuristics is to find the sequence of low-level heuristics that optimizes the solution of a given problem instance. Therefore, unlike the previously described approaches, it is not *static* (i.e., defined *before* the solution process), but *dynamic*, (i.e., defined *during* the solution process).

Despite their adaptive nature, a significant drawback of timetabling hyper-heuristics is their computational cost [46]. For each instance to be solved, a different high-level heuristic is constructed, involving computational tasks related to the iterative selection and testing of low-level heuristics. However, as with the other described solution methods, there is no guarantee to find an optimal solution.

1.3.3 Discussion of solution methods

In the presented literature review, two types of solution approaches for timetabling instances have been described: *single-models* and *multi-models*. As described, single-model approaches (applied to solve entire instances) include *i)* analytical methods, which based on exact formulations have proved to be useful to find optimal solutions for small-size instances, and *ii)* stochastic heuristics (both solution and population-based), which explores the solution spaces using varying strategies to avoid being stuck in local optima. An alternative approach is the application of *multi-models*, designed to combine the strengths of different solution methods, either statically —using hybrid methods— or dynamically —using hyper-heuristics.

As presented in Figure 1.4, a third multi-model strategy, not formally applied within the context of the CB-CTT problem, is per-instance algorithm selection. This strategy differs from hybrid methods in that it can integrate algorithms without information transference, thus reducing the computational effort required to solve instances. Besides, though being an adaptive approach —dynamically suited to the features of each instance to be solved— it is not concerned with the construction of high-level heuristics (as hyper-heuristics are), but with the selection of algorithms and heuristics, to provide more time to the solution process *per se*.

1.4 Summary

This chapter presented the theoretical foundation for the solution of CB-CTT instances, organized into three parts. The first part explained the mathematical formulation of the problem and illustrated the main elements required to define an instance. This formulation is relevant for a clear understanding of the combinatorial nature of the problem and its related solving complexity. The second part described two data formats defined as international standards to represent timetabling instances and compared their structures and modeling capabilities. As a result, the reasons for selecting the XHSTT data format were pointed out, providing a general overview of the set of constraints considered for this thesis. Finally, the third part provided a brief review of solution approaches already applied in the educational timetabling field, indicating relevant differences with the per-instance algorithm selection model approach employed in this thesis. In the next chapter, this approach is described in detail.

Chapter 2

Per-Instance Algorithm Selection

The previous chapter presented the mathematical formulation of the Curriculum-Based Course Timetabling (CB-CTT) problem and a literature review of the available data formats and solution approaches. This chapter focuses on a multi-model solution approach still not applied within the context of educational timetabling: per-instance algorithm selection.

Unlike other solution methods, discussed in the previous chapter, per-instance algorithm selection aims to integrate the strengths of algorithms without requiring complex processes to link algorithms —as hybrid methods— or complex ensembling methods to build algorithms —as hyper-heuristics. Hence, it can be regarded as an adaptive method, which aims to integrate the strengths of algorithms and heuristics in a computationally efficient manner.

Per-instance algorithm selection has achieved significant improvements in the solutions of well-known combinatorial optimization problems, where the goal is to find the best solution for a given objective function or to determine if there exists a solution able to satisfy certain constraints. Some of these problems include propositional satisfiability (SAT) [7], the quadratic assignment problem (QAP) [47], and the traveling salesman problem (TSP) [48]. Therefore, given the proven effectiveness in combinatorial optimization problems and its goal to combine the strengths of different solution methods (i.e., algorithms and heuristics), in this thesis, we defined the per-instance algorithm selection approach to improve the solution of CB-CTT instances.

As an introduction to this multi-model solution approach, in this chapter, we describe three aspects required to formalize the construction of any per-instance algorithm selection model (per-instance ASM): *i)* formulation of per-instance ASM, *ii)* design of per-instance ASM, *iii)* methodology of per-instance ASM.

2.1 Formulation of Per-Instance ASM

Research communities of well-studied problem domains have long observed that different algorithms perform best over different instance spaces. Thus, no single algorithm dominates the others across entire problem spaces [49]. This phenomenon, known as *performance complementarity*, gives rise to a relevant computational problem: *how to integrate the strengths of different algorithms to obtain better solutions?*

Current research works have proposed diverse alternatives to answer this question [17]. Most of these works make use of a general strategy: trying to predict the *performance* of a set of algorithms as a function of relevant *instance features.* This strategy has been suited to the particular needs of different problem domains using various methods that constitute the state of the art of *algorithm selection.*

According to Kerschke [49], to avoid confusion of related terms, it is crucial to distinguish between three approaches related to algorithm selection: *i*) algorithm configuration, *ii*) per-instance algorithm selection, and *iii*) algorithm schedules.

- Algorithm configuration aims to choose the best values for a set of parameters in order to optimize the execution of an algorithm.

- Per-instance algorithm selection aims to select, from a set of algorithms, the one with the best expected performance for a given instance.

- Algorithm schedules aims to define the sequence and running time for a set of algorithms.

Decades ago, Rice [6] was the first to formally define the per-instance selection approach using the term *algorithm selection problem*. However, within the current literature, this is now a broad term used to also refer to other related but conceptually different approaches. To avoid confusion, this thesis employs the original formulation proposed by Rice but it makes use of the more precise term *per-instance algorithm selection problem* to refer to it.

The formal definition of the per-instance algorithm selection problem requires the four basic elements described next.

Problem space ($\mathcal{P}$): A representative set of *instances* of the problem under study.

Feature space ($\mathcal{F}$): A set of *features* describing the characteristic properties of the problem space.

Algorithm space ($\mathcal{A}$): A set of *algorithms* able to solve the problem.

Performance space ($\mathcal{Y}$): A set of *performance metrics* of the algorithms (e.g., running time), or its solutions (e.g., value of the objective function), across the problem space.

From these elements, the per-instance algorithm selection problem is stated as follows. Given a problem *instance* $x \in \mathcal{P}$, described by a set of relevant *features* $f(x) \in \mathcal{F}$, and solved by a set of *algorithms* $\mathcal{A}$, find the *selection mapping* $S(f(x), \mathcal{A})$ of features to algorithms, in order to select the algorithm $\alpha \in \mathcal{A}$ that optimizes the *performance metric* $y(\alpha(x)) \in \mathcal{Y}$.

Despite its apparent simple formulation, per-instance algorithm selection is a robust computational approach involving multiple data-processing tasks. Diverse machine learning approaches have been proposed to analyze the large amount of data required for the construction of selection models in different problem domains. In the combinatorial optimization field, two research works have proved useful for the design and construction of algorithm selection models in real applications. The first one, useful for the design of per-instance ASM, is the survey performed by Kotthoff [1], which describes the modeling decisions to be considered for the overall configuration of this type of model. The

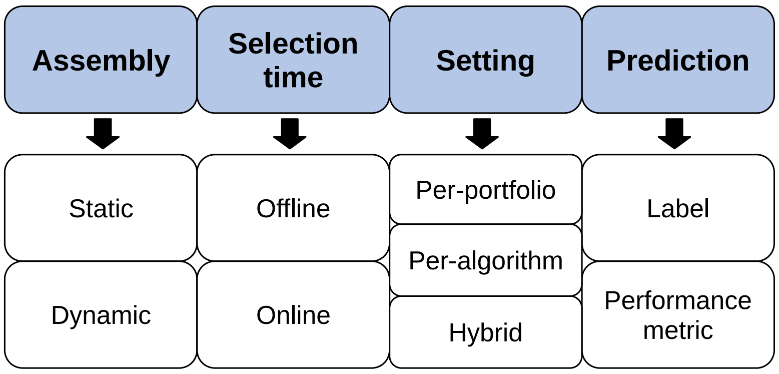


Figure 2.1: Modeling decisions to be addressed for the design of algorithm selection models in the combinatorial field, according the survey performed by Kotthoff [1].

second one, defined as a general methodology, is the meta-learning framework proposed by Brazdil et al. [2], which describes the elements and sequence of tasks to construct this type of model. The next sections discuss these research works and explain their relevance to this thesis.

2.2 Design of Per-Instance ASM

The construction of an per-instance algorithm selection model is a complex process that involves more than putting together a set of algorithms. It comprises diverse statistical analyses and prediction tasks that need to be addressed in a structured manner to define the overall configuration of the selection models.

According to the survey about algorithm selection for combinatorial problems made by Kotthoff [1], there are four modeling decisions that must be addressed to define the overall structure of any per-instance algorithm selection model in the combinatorial field. These modeling decisions, presented in Figure 2.1, are briefly explained next.

Assembly: The first major decision to be made regarding the design of a per-instance algorithm selection model is defining the assembly approach for the algorithm portfolio. Two approaches can be used: assembly the algorithm portfolio *before* any instance is solved, or *while* the solution process is running. The first of these approaches is said to be *static* since the composition of the portfolio remains unchanged during the solving process, that is, unaffected by the instance being solved. The second approach is said to be *dynamic*, since it can change either the composition or configuration of the constituent algorithms, according to instances properties.

Selection time: Similarly to the assembly of an algorithm portfolio, the selection of an algorithm can be performed in two different times: *before* the instance is solved (*offline*), or *while* it is being solved (*online*). The online approach is more flexible since it monitors the execution of the chosen algorithm to confirm that it is performing as expected. However, this selection flexibility entails a higher computational effort.

Setting: The computational effort required to select an algorithm depends mainly on the settings defined to structure the model. According to their complexity, they can be grouped into three types. *Per-portfolio* settings are designed to predict the performance of entire portfolios (as a group), without a clear understanding of the individual performance of the algorithms. *Per-algorithm* settings are designed to predict the individual performance of the algorithms across a problem space, and by comparing these predictions, selecting the best algorithm to be applied for a particular instance. *Hybrid* settings combine per-portfolio and per-algorithm models in sequential or hierarchical order to get more accurate predictions, at the cost of more computationally expensive models.

Prediction: The final output used to select the algorithm depends on the setting defined for the model. On the one hand, *per-portfolio* settings select algorithms in a direct manner, representing algorithms as labels that are predicted with classification models. On the other hand, *per-algorithm* settings select algorithms in an indirect manner, predicting the performance of the algorithms (in a numeric scale) and then choosing the one with the best expected performance.

The presented modeling decisions define the overall configuration of per-instance selection models and, as a consequence, both their usefulness and computational complexity. Complex models usually make more accurate algorithm selections. However, if they become computationally more expensive than actually solving the instances, they become unpractical. Ultimately, there is no point in performing a per-instance algorithm selection that requires more resources than solving the instances.

2.3 Methodology of Per-Instance ASM

Besides defining the overall configuration to construct a per-instance algorithm selection model, another major task to be addressed is choosing the right methodology to conduct the data-processing flow required to build the selection mapping $S(f(x), \mathcal{A})$, of instance features to the performance of algorithms.

Across different problem domains, multiple methods have been used to create such mapping S [1, 50]. As observed in international algorithm selection competitions [16], some examples include: regression models, that predict the performance of individual algorithms; unsupervised clustering, that partitions instances into clusters and assigns an algorithm to each cluster; pairwise classification, that compares the performance of each pair of algorithms and selects the one with most "victories" in the overall comparisons; stacking, that combine the predictions of machine learning models of different nature to make the final selection. However, one way or another, most of them have been built using the process that Brazdil et al. [2] formally defined as meta-learning.

As shown in Figure 2.2, the *meta-learning framework* starts with a repository of instances representative of a problem domain. Each instance is used as a training example to perform the features-to-algorithm mapping. Thus, such instances are used to generate the required *meta-data* that stores both, relevant features of the problem (*meta-features*), and information about the performance of the algorithms applied to solve the instances (*algorithms' performance*). This meta-data is used to train a machine learning method (*meta-learner*) to build the *algorithm selection model* for a given problem.

The following is a brief review of the implementation strategies available in the current state of the art for each stage of the meta-learning process.

Data repository: Because of their practical applications, many benchmarking datasets

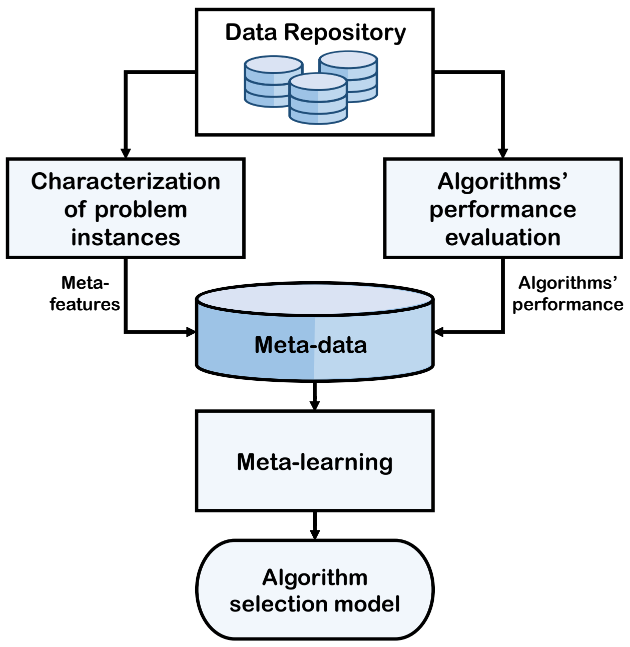

Data Repository
Characterization of problem instances
Algorithms' performance evaluation
Meta-features
Algorithms' performance
Meta-data
Meta-learning
Algorithm selection model

Figure 2.2: Meta-learning framework for algorithm selection models (adapted from [2]).

have been proposed to compare the effectiveness of solution methods for different combinatorial optimization problems, such as the traveling salesman problem [51] and graph coloring [52]. However, if these datasets are not available, researchers usually require to gather a significant amount of instances about a specific problem [53, 54], or artificially generate enough instances to represent a particular problem domain [55].

Characterization of problem instances: The performance of meta-learning-based models directly depends on selecting significant features (meta-features) in order to represent relevant properties of problem instances. Most meta-features are obtained by extracting morphological characteristics from the data. According to Reif et al. [56] these characteristics can be categorized, as: simple (directly observed), statistical (e.g., mean, variance), information-theoretic (e.g., Shannon entropy), model-based (extracted from complex learning algorithms), and landmarking (obtained from light-learning algorithms).

Algorithms performance evaluation: Because of its generalization capability, meta-learning has been applied to algorithm selection in a wide variety of fields related to prediction, classification, and optimization tasks. Depending on its purpose, the performance of algorithms has been evaluated in different ways such as: prediction errors [54, 57, 58], intra-cluster measures [59], classification accuracy [60], value of objective functions [51], and solution times [61].

Meta-learning: The selection mapping $S(f(x), \mathcal{A})$ is usually built using supervised learning. The current literature contains examples of three main approaches to build meta-learners: prediction, classification, and ranking. In *prediction*, a regression model is used to select the algorithm with the highest expected performance [8]. In *classification*, a classifier is built to predict the label that indicates the best algorithm for a given instance [62]. Finally, when *ranking* is used, either regression models or classifiers are applied to predict not only the best algorithm to solve a problem, but also the relative order in which the algorithms could be applied [63, 64].

Algorithm selection model: Once the meta-learning process is complete, the algorithm selection model designed for a problem can be evaluated. According to the type of meta-learner, two types of metrics, commonly applied to evaluate the performance of the selection models, are used: accuracy and concordance. On the one hand, *accuracy* measures summarize the proportion of selection errors for prediction [8] and classification [65] approaches. On the other hand, *concordance* measures, such as Spearman's coefficient [59], evaluate the relationship between the predicted and the ideal ranking of algorithms.

As noticed, the implementation of the meta-learning process covers a wide range of conceptual and technical decisions that requires both, expertise in machine learning methods and professional experience in the problem domain. Hence, a successful implementation demands not only a systematic application of computational tools but mainly of a well-designed data processing flow.

2.4 Summary

As presented in this chapter, the decisions required to construct accurate per-instance algorithm selection models involve diverse aspects related to *i)* the definition of four related spaces (i.e., problem, feature, algorithm, and performance), and *ii)* the configuration of the computational models employed to build the selection mapping $S(f(x), \mathcal{A})$.

In the current literature, different strategies have been proposed to analyze these aspects in diverse problem domains. However, due to their flexibility and usefulness in the combinatorial optimization field, in this thesis, we employed the described *modeling decisions* and *meta-learning framework* to formalize the construction process of the CB-CTT per-instance algorithm selection model.

In the overall process, the *modeling decisions* were taken into account to not add more computational effort to the solution of CB-CTT instances, and the *meta-learning framework* to guide the tasks related to the data analyses required by the performance mapping.

The following chapters describe the overall meta-learning methodology for the construction of a per-instance algorithm selection model, able to integrate a portfolio of algorithms for the solution of CB-CTT instances. As this is a multi-model approach not yet applied for solving educational timetabling problems, each chapter points out the main contributions of this thesis at each stage of the construction process. Chapter 3 describes the generation process of the repository of problem instances; Chapter 4 presents the meta-features formulated to characterize the instances; Chapter 5 describes the algorithm portfolio and evaluates its performance; Chapter 6 describes the per-instance algorithm selection model.

Chapter 3

Instance Generator

The previous chapter presented a general overview of elementary aspects to construct a per-instance algorithm selection model. As explained, the construction process involves diverse data-processing tasks related to four related spaces: *Problem space* ($\mathcal{P}$), *Feature space* ($\mathcal{F}$), *Algorithm space* ($\mathcal{A}$), and *Performance space* ($\mathcal{Y}$). To perform the required data-processing tasks in a logical manner, for this thesis, we selected the meta-learning framework. As illustrated in Figure 3.1, this framework defines a sequence of activities that considers all the essential elements of per-instance algorithm selection models. This chapter describes the first of these elements, a representative data repository of instances for the Curriculum-Based Course Timetabling (CB-CTT) problem.

In Section 1.2, two formats were evaluated to represent the CB-CTT instances required for this thesis in a standard manner, namely the ECTT and the XHSTT data formats. As explained, the XHSTT format was selected because of two factors: its hierarchical tag-based structure, and its broader modeling capacity. The XHSTT data format represents timetabling instances based on an XML schema (illustrated in Figure 1.3) that requires four entities, sequentially defined as follows: *times*, *resources*, *events*, and *constraints*. Explaining in detail the syntax of the format is out of the scope of this thesis. However, we refer to the reader to two useful sources: *i)* the official webpage of the XHSTT format[1], containing a full specification of its syntax, and *ii)* the Appendix B at the end of this thesis, illustrating the modeling process of a real timetabling problem

[1]http://jeffreykingston.id.au/cgi-bin/hseval.cgi?op=spec

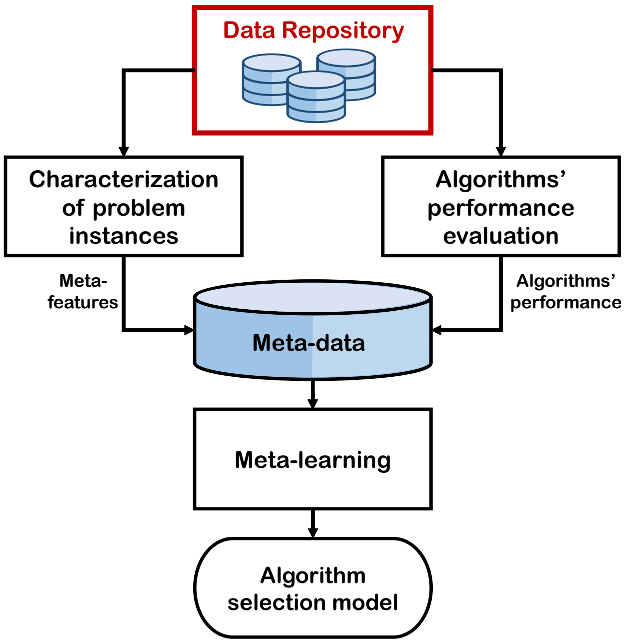

Data Repository
Characterization of problem instances
Algorithms' performance evaluation
Meta-features
Algorithms' performance
Meta-data
Meta-learning
Algorithm selection model

Figure 3.1: Meta-learning framework for the construction of the CB-CTT per-instance algorithm selection model (adapted from [2]). For reference purposes, the first element, *Data repository*, related to the *Problem space* ($\mathcal{P}$), is highlighted in red.

into an XHSTT instance.

Within educational timetabling, at present, there is only one available standard benchmarking dataset modeled according to the XHSTT format, namely the XHSTT-2014 dataset hosted by the Centre of Telematics and Information Technology (CTIT) of the University of Twente [66]. The dataset consists of a collection of 25 real-world instances that are encoded according to the XML schema proposed by Post et al. [23]. However, because of its size, such a dataset contains a rather small number of instances, which may limit the findings of this thesis.

To overcome the current lack of CB-CTT instances, as one of the contributions of this thesis, we designed a customizable generator using the ElementTree library of the Python programming language. This generator was further applied to generate multiple XHSTT instances with different combinatorial properties, representing as much as possible the diversity of conditions found in real problems. For explanation purposes, in this chapter, the design and setting of the instance generator are explained first, then the generation process of the instances is described.

3.1 Design of the CB-CTT Instance Generator

The general strategy defined for the design of the instance generator was structured in two stages, according to the essential information required to define a educational timetabling instance: a *curricular plan* (Stage 1), and a set of *particular conditions* (Stage 2). On the one hand, *curricular plans* are general descriptions of academic programs, and in a broad sense define: *i)* the set of obligatory and optional courses, *ii)* the sequence of these courses, and *iii)* their duration and required resources. On the other hand, *Particular conditions* are institutional regulations that must be followed for the *scheduling* of times and *allocation* of resources within a determined educational context.

3.1.1 Stage 1 - Curricular Plan

The first element created in the generation process is the curricular plan, which describes the set of courses to be considered to formulate a timetabling instance. The generator creates a different curricular plan for each instance, defined by a *curricular grid* and a set of *courses' requirements.* At a macro level, the curricular grid is set by specifying the number of terms (e.g., semester, trimesters, quarters) required to study a bachelor's degree and the number of courses on each term. At the micro level, the courses' requirements are set by specifying the duration and resources required by each course in the curricular grid.

Once the curricular map is set, the generator selects a set of courses from the curricular grid to create the classes that will be included in the timetabling instance. The number of classes to be scheduled are defined based on two parameters: *active terms*, and *number of groups.* The first parameter refers to a subset of terms, from the curricular plan, planned to be available for student enrollment. The second parameter refers to the number of classes to be generated for each course of those *active terms.*

Figure 3.2 presents a graphical example of a five-term curricular plan. Each square represents a course, defined in terms of three parameters randomly generated: *weekly duration*, *type of professor required*, and *type of room required.* For example, the re-

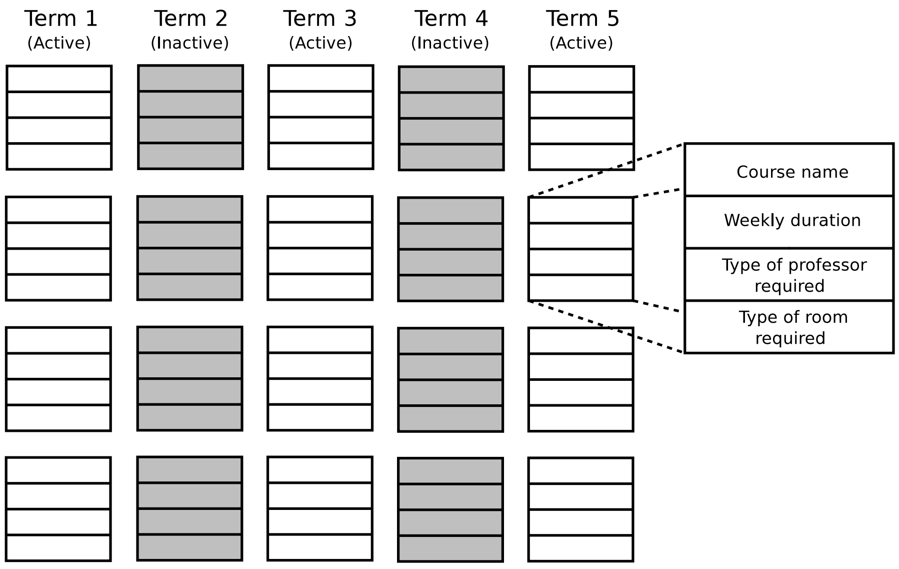


Figure 3.2: Graphical representation of a five-term curricular plan, where terms 1, 3 and 5 are available for student enrollment, and each square represents a course defined by its name, weekly duration, type of professor and type of room required.

quirements of a course called "Matemathics", could be defined as follows: <*Course name*><Mathematics>; <*Weekly duration*><5>; <*Type of professor required*><Science teacher>; <*Type of room required*><Classroom>. As observed in the figure, three terms colored in white (first, third and fifth) are defined as active terms, hence if the parameter number of groups is set as <number of groups><2>, the generator will include 24 classes in the instance; two classes (with the same requirements) for each one of the courses represented as white squares.

3.1.2 Stage 2 - Particular Conditions

After all the classes (created from the courses in the curricular plan) are included in the instance, the generator defines the particular conditions for the assignment of times and resources to those classes. These conditions are related to three entities of the XHSTT format: *times*, *resources*, and *constraints*.

Times

The instance generator represent *times* as a list of non-overlapping time intervals, called *time slots*, that are grouped into different sets of *days* and *working shifts*. The number of days and time slots define the dimensions of the *time grid* in which the classes will be scheduled; this time grid is then equally divided in different working shifts. For example, a time grid consisting of four morning time slots and four evening time slots, from Monday to Friday, would be modeled with the following parameters, <*time slots per day*><8>; <*days*><5>; <*shifts*><2>.

Each class has a weekly duration that indicates the number of time slots that it must be scheduled. According to their duration, classes can be split into *lectures* and then scheduled using different configurations. For example, a class called "Ethics" with a weekly duration of two time slots, could be scheduled using the following *lecture configurations*: a single two-times slots lecture (2), or two one-time slot lectures (1-1). The larger the duration of a class, the more the possible lecture configurations.

Resources

Resources are elements that have to be present during all the lectures of a class. Our instance generator considers three types of resources:

- **Curricula**: In timetabling, curriculum refers to a set of classes that are shared by a group of students. This type of resource is used by education planners to ensure that students can enroll in their classes without clashes. For example, to ensure that fifth-semester students can attend to all their common classes. Our instance generator defines the curricula of an instance based on the parameters *active terms* and *number of groups* explained above. For example, if two groups are considered for the courses in the active terms shown in Figure 3.2, the instance generator will define six curricula: Term 1 Group A, Term 1 Group B, Term 3 Group A, Term 3 Group B, Term 5 Group A, and Term 5 Group B. Using this strategy, the classes assigned to each curriculum are required to be scheduled at a different time.

- **Teachers**: Professors are grouped based on three attributes: *preferred working shift* (e.g. morning, evening), *knowledge area* (e.g. math, science, management, etc.) and, *type of contract* (full-time or part-time teacher).

- **Rooms**: The physical spaces in which lectures take place are categorized according to their equipment as classrooms or laboratories.

Constraints

Constraints are mathematical expressions that limit the assignment of times and resources to the lectures. In our generator, the set of available constraints that are applied to model real conditions are defined using three general parameters:

- **Status**: Specifies if the constraint will be "turned on" or "turned off", that is, whether or not it will be included in the instance.
- **Penalty type**: Specifies if the constraint will be considered as *hard* (mandatory) or *soft* (optional) in the cost function.
- **Weight**: Specifies the penalty value which will be added to the cost function if the constraint is violated.

Based on the values defined for these parameters, a total of 27 real conditions can be modeled by the constraints included in our instance generator. Some examples of these real conditions are:

- **Prefer room**: Require the assignment of a specific type of laboratory for a class.
- **Weekly workload of full-time teachers**: Limit the weekly working hours of a full-time teacher.
- **Working shift**: Require that the lectures of a teacher be scheduled only on one of two possible shifts: morning or afternoon.
- **Single lecture**: Require that all lectures of a class be scheduled in different days.

The full set of real conditions modeled by the instance generator is described in Appendix A.

3.1.3 Pre-assignment

As explained in Section 1.1, solving a CB-CTT instance requires the assignment of times and resources to all the lectures, according to a set of constraints. However, in real-life situations, the times or resources of certain lectures might be known in advance, thus being *pre-assigned* before the solution process. Lectures with pre-assigned times and resources reduce the size of the search space to solve an instance, an important aspect also considered in the design of our generator.

To analyze the effect of pre-assignment in the solution process of an instance both time of pre-assignments are performed by the generator as follows. *Time pre-assignment* is performed by defining a proportion of events to be randomly assigned to a set of consecutive time slots. *Resource pre-assignment* is performed by defining a proportion of events to which either a teacher or a room is randomly assigned, according to the requirements of each course.

3.2 Setting up the Instance Generator

Setting up the parameters of the instance generator is, in itself, a time-consuming task. It involves selecting from a domain of possible values the ones that guarantee logical relationships between the elements of an instance. A wrong selection of these parameters can lead to the generation of defective instances, without feasible solution spaces. To prevent a wrong setting of parameters, the generator was designed to automatically avoid two types of errors: i) insufficiency of resources, and ii) over-constraining.

3.2.1 Insufficiency of Resources

Insufficiency of resources is an error that occurs when the demand for resources required by the lectures surpasses the available supply. For example, requiring eight rooms to allocate the lectures planned for an instance but having only six rooms.

To ensure that it is possible to allocate the required resources to all the lectures, the generator calculates the total demand for each type of teacher and room. Then, according to the weekly workload defined for each type, it generates the minimum number of teachers and rooms necessary to balance the supply and demand of resources.

3.2.2 Over-constraining

Over-constraining is an error that occurs when the fulfillment of a constraint implies the violation of another one. For example, requiring that a laboratory be used only four days a week, but assigning it to classes defined to be scheduled daily (five days a week). The generator avoids over-constraining errors using a hierarchical approach to define the particular conditions for the assignment of times and resources; first, the basic conditions, then the additional.

On the one hand, basic conditions are those always defined as hard (mandatory) constraints in the instances. These conditions, numbered according to their order in the Appendix A, are the following: *Assign teachers* (1), *Assign rooms* (2) and *Assign times* (3) to all lectures; *Avoid clashes* (4) of resources ; *Prefer teachers* (7) and *Prefer rooms* (8) to be allocated according the courses requirements; ensure *Teachers stability* (10), *Rooms stability* (11) and *Courses stability* (12); avoid assigning teachers and curricula out of their respective *Working shifts* (16) and *Study shifts* (17). On the other hand, additional conditions correspond to the rest of the ones listed in the the Appendix A. Additional conditions are included (randomly as hard or soft constraints), one by one, only if their inclusion does not cause conflict with the basic ones.

3.3 CB-CTT Instance Dataset

The described CB-CTT instance generator was designed to create a diverse set of instances, both in size and constraint density. As explained, it works in two stages: *Stage 1* defines the curricular map (i.e., courses and their requirements); while *Stage 2* the particular conditions for the assignment of times and resources (i.e., times, resources, and constraints). As a mean to define the experimental space without incurring in logical errors, in this thesis the parameters of the CB-CTT instance generator were set according the five-steps suggested by Barr et al. [67] to perform computational experiments. These steps are: *i)* defining relevant parameters, *ii)* setting boundaries for the parameter values, *iii)* defining modeling scenarios, *iv)* generating instances, *v)* verifying consistency and diversity of the instances.

3.3.1 Defining Relevant Parameters

The instance generator provides a set of 152 parameters that can be tuned to define a CB-CTT instance. From these parameters, 26 are related to lectures, times, and resources; and 126 are related to constraints. In order to generate a diverse CB-CTT dataset, all of the 152 parameters were considered relevant and required to be set in the instance generation process[2]

3.3.2 Setting Boundaries for the Parameters Values

The values of the parameters were limited to represent conditions that are commonly found in real university timetabling problems, thus specifying the domain of the random values to be generated. For example, the domain of values for the parameter <*Terms of the curricular plan*>, which defines the number of terms of a curricular plan, was set to {8, 9, 10, 11}. Therefore, the generator created curricular plans ranging from a duration of 8 to 11 terms (i.e., semesters, trimesters or quarters).

3.3.3 Defining Modeling Scenarios

A structured manner to analyze the effects that different combinations of parameter values can have over the complexity of an instance is defining modeling scenarios. Modeling scenarios work by partitioning the domain of each parameter into ranges of values with an expected similar complexity. For example, it is known that including a more diverse set of resources on an instance increases the number of constraints to be applied. Thus,

[2]This thesis focuses on presenting a general description of the performed tasks to construct a per-instance algorithm selection model for solving CB-CTT instances. We plan to include all details related to the instance generator, CB-CTT dataset, and instance features in a set of technical reports. Meanwhile, we have created the following Github repository to update the resources related to this research project `https://github.com/Felipedlr/CB-CTT-dataset`.

to analyze the effect of the diversity of classrooms in the complexity of an instance, the following three scenarios were defined: <*Types of classrooms*>: <1>, <2>, or <3>. In a similar way, different modeling scenarios were defined for each parameter required by the generator.

3.3.4 Generating Instances

After defining the domains and modeling scenarios, the values of the parameters were produced from a uniform probability distribution using pseudo-random number generators. The designed generator was run on a desktop PC with an Intel Core i7-6700 processor at 3.4 GHz and 16 GB of RAM, running Microsoft Windows 10. A dataset of 10,000 instances were generated in a total running time of 21,315 seconds; that is, 2.13 seconds per instance.

3.3.5 Verifying Consistency and Diversity of the Instances

As described in Section 1.2, the XHSTT data format was proposed in 2011 as an international standard to represent timetabling instances from different countries. However, at the moment, not many solvers have been proposed to support this format.

In the current state of the art, two types of XHSTT instance solvers can be distinguished: i) solvers proposed to *generate* initial solutions and ii) solvers proposed to *improve* initial solutions. To estimate the empirical hardness (i.e., apparent solving complexity) of the generated instances, the *KHE timetabling engine* [68] —a solver that has proved to be effective to generate initial solution in current research works [69–73]— was employed to accomplish two goals: i) validate the structural consistency of the generated instances, and ii) measure their empirical hardness.

If the KHE timetable engine is able to generate an initial solution for an instance, then this validates the structural consistency of such an instance. As part of the solution, the KHE timetable engine returns a report describing both the total cost of violations of hard constraints (*infeasibility value*) and, the total cost of violations of soft constraints (*objective value*). In this thesis, we combined these values into a single cost called penalty value, which —as defined for many instances in international timetabling competitions— assigns a cost value of 1,000 to each violation of a hard constraint, and a cost value of 1, to each violation of a soft constraint.

The penalty value was used to measure the empirical hardness of the instances, and to evaluate the diversity of the dataset. Based on an empirical statistical analysis of histogram plots, instances were grouped into five categories according to their penalty values:

- **Very Easy**: Instances with a penalty value between [0 - 1,000).
- **Easy**: Instances with a penalty value between [1,000 - 10,000).
- **Medium**: Instances with a penalty value between [10,000 - 50,000).
- **Hard**: Instances with a penalty value between [50,000 - 100,000).
- **Very Hard**: Instances with a penalty equal or greater than 100,000.

The generated dataset of 10,000 instances was classified according to these ranges of penalty values, revealing an uneven distribution of instances among the five defined categories. In this distribution, the categories *Very Easy* and *Easy* exhibited a significantly higher number of instances than the rest. Hence, to avoid dealing with problems related to unbalanced datasets in subsequent stages of the meta-learning process, 1200 instances of each category were selected for the final CB-CTT instance dataset (i.e., a total of 6,000 instances).

3.4 Summary

The first requirement to create a useful algorithm selection model is having a diverse collection of instances properly resembling the conditions found in real-life situations. In the current state of the art, different collections of timetabling instances have been proposed as benchmarking datasets for researchers. Still, only one of them (consisting of 25 instances) employs the XHSTT data format used for this thesis. To address the current lack of XHSTT instances in the timetabling domain, in this thesis, we created a hierarchical instance generator able to model a set of 27 common conditions.

In this chapter, the design of our instance generator was presented first; then, the generation process of the required CB-CTT instances was explained in detail. It is important to point out that because of the large amount of parameters (152) to be set in the generator, we were able to create a highly-diverse dataset of 6,000 instances (ranging from *very easy* to *very hard*). This dataset —validated using a well-known XHSTT instance solver— was defined as the *problem space* for the construction of our per-instance algorithm selection model and described according to the set of features presented in the next chapter.

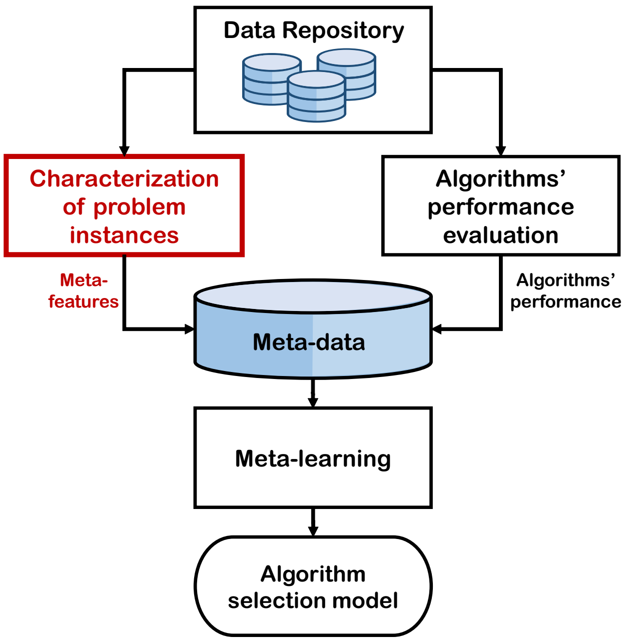
Chapter 4

Feature Selection

In the previous chapter, the generation process of the instance dataset for the Curriculum-Based Course Timetabling (CB-CTT) problem was described. As explained, it is a dataset balanced according five different instance types that indicate the initial solving complexity of the instances, ranging from *Very Easy* to *Very Hard.* The dataset includes 6,000 instances, 1,200 of each type.

According to the meta-learning framework shown in Figure 4.1, the next step for the construction of our per-instance algorithm selection model is the characterization of the generated instance dataset in terms of a set of *meta-features*, that is, measurable properties able to describe the main characteristics of the CB-CTT problem. To achieve this goal we propose a collection of features to describe the generated instances (modeled according to the XHSTT format described in Section 1.2.2) into numerical values. The relevance of these features depends on their effectiveness to predict the performance of algorithms across the CB-CTT problem domain. Hence, to evaluate the proposed collection of features —called CB-CTT metrics— we also present an assessment of its predictive power based on the concepts of *empirical hardness model* and *feature selection.*

As defined by Leyton et al. [74], *empirical hardness models* are statistical predictors built to estimate the performance that a given algorithm will achieve when solving a given problem instance. Due to its statistical nature, these predictors are often formulated as regression models, which can be analyzed to develop new solving approaches


Data Repository
Characterization of problem instances
Algorithms' performance evaluation
Meta-features
Algorithms' performance
Meta-data
Meta-learning
Algorithm selection model
Figure 4.1: Meta-learning framework for the construction of the CB-CTT per-instance algorithm selection model (adapted from [2]). For reference purposes, the elements *Characterization of problem instances* and *Meta-features*, related to the *Feature space* ($\mathcal{F}$), are highlighted in red.

and improve the theoretical understanding of a problem domain.

As a contribution to the understanding of the CB-CTT problem, in this chapter, besides the formulation of a collection of features (CB-CTT metrics) to characterize instances in the XHSTT format, we present the analysis of empirical hardness models following the *feature selection* methodology shown in Figure 4.2. The goal of this methodology is selecting and interpreting the set of features (Relevant CB-CTT metrics) that better predicts the solving difficulty of the instances. The applied methodology includes the four processes described next:

1. **Instance description**: It involves describing the CB-CTT instance dataset in terms of representative numerical values calculated from the formulated CB-CTT metrics (collection of features).

2. **Performance prediction**: It involves creating empirical hardness models to predict the solving difficulty of the CB-CTT instances (measured in terms of the *penalty value*[1]).

[1]As explained in Section 3.3.5, the penalty value is the sum of the cost of hard and soft constraint violations incurred in the solution of an instance

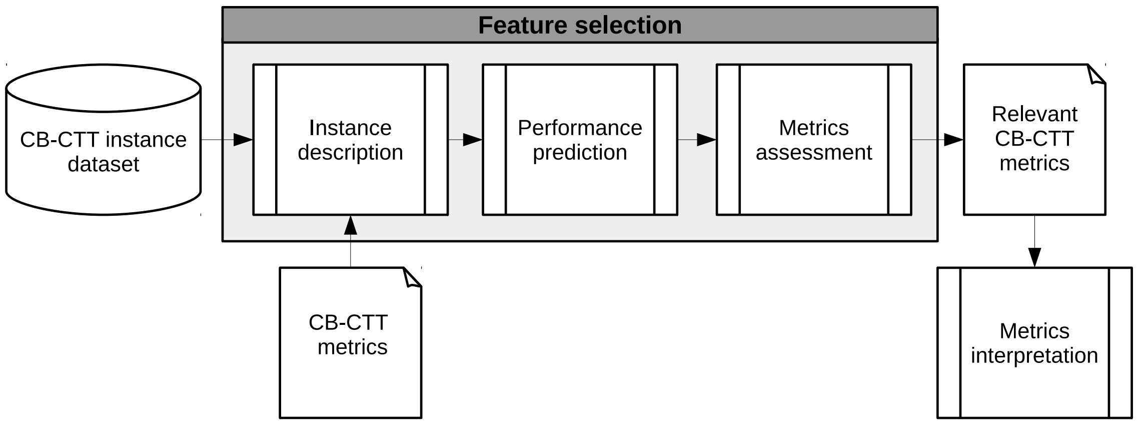


Figure 4.2: Feature selection methodology used to identify the metrics that better describe the empirical hardness of CB-CTT instances.

3. **Metrics assessment**: It involves analyzing the composition of the created empirical hardness models to determine the relevance of each metric (or feature) in the prediction task.

4. **Metrics interpretation**: It involves explaining the practical implications of the most relevant CB-CTT metrics in the solution process of the instances.

The remaining of this chapter is organized based on the presented feature selection methodology. All elements are described in detail.

4.1 CB-CTT Metrics

Formulating features to characterize instances on a problem domain is a laborious task which requires a combination of general and domain-specific knowledge. However, once a descriptive set of features has been formulated, it can be used by the research community as a starting point to propose new solving methods and drive research forward [1].

Features within the combinatorial optimization field can be exploited in different ways, for example: to choose the best algorithm to solve a given instance [8, 51]; to guide the selection of the best operator to be applied at each iteration of an hyper-heuristic approach [46,75]; to improve the searching strategies of meta-heuristics [10,76]. However, their effectiveness to perform these tasks cannot be *a priori* guaranteed, since it depends on their descriptive capability over the problem space.

Currently, one set of features that has been proposed to characterize instances within the university timetabling domain is the set of 32 features formulated by Smith-Miles & Lopes [77]. The set considers four types of features: 3 size-related features, 2 landmarking features, 21 graph-coloring features, and 6 timetabling features. However, as it is mainly based on the properties of conflict graphs $G(V, E)$ (where V is a set of vertices corresponding to *events* that need to be timetabled, and E is a set of edges connecting any two vertices when the events cannot occur at the same time), the set is not suitable to describe the full set of constraints considered by our CB-CTT instance generator (explained in Chapter 3). To overcome this limitation, this thesis proposes four types of features to characterize timetabling instances within the CB-CTT domain, and as each type measures different properties of the problem (e.g., solution space, conflicts of resources), they are called *metrics*.

The proposed CB-CTT metrics consists of: i) 74 *simple metrics* that provide a basic notion of the size of the elements that compose an instance; ii) 44 *solution space metrics* that, based on counting functions, measure the size of possible combinations for task scheduling and resource allocation; iii) 17 *feature ratios* that calculate numerical relationships between some *simple* and *solution space* metrics; and iv) 14 *constraint density indexes*[2] defined by Kingston [68]. Therefore, the set consists of a total of 149 CB-CTT metrics or features that are described next.

4.1.1 Simple Metrics

We define *simple metrics* as observable data that provides a basic notion of the size of an instance. As instances are modeled using the XHSTT format, simple metrics are calculated by parsing XML trees to perform counting processes regarding the basic elements that compose an instance. For example, total number of classes, total number of teachers, total number of rooms, total number of constraints, etc.

In order to explain the set of 74 simple metrics, we make use of two tables (Table

[2]These 14 constraint indexes are the only ones not formulated by us, but as part of the KHE Timetabling Engine (the solver applied to obtain the initial solutions for our instance dataset), they were included to complement our metrics.

Table 4.1: Simple constraint-related metrics used to describe each one of the 14 types of constraints used by the CB-CTT generator.

	Metric	Description
1	Constraints	Number of constraints of a certain type that are applied to an instance
2	Points of application	Number of elements that needs to be evaluated to verify the fulfillment of a type constraint
3	Soft penalty value	Hard penalty value that is going to be added to the cost function if all points of application fail incurs at least in one constraint violation.
4	Hard penalty value	Hard penalty value that is going to be added to the cost function if all points of application fail incurs at least in one constraint violation.

4.1 and 4.2). Table 4.1 presents four metrics that are used to describe each one of the 14 types of XHSTT constraints considered by our instance generator[3] —a total of 56 metrics. For example, if two Limit Workload Constraints (LWC) defined as hard (mandatory) are included on an instance, one to limit to 20 the working hours of a set of 8 part-time teachers, the other to limit to 15 the working hours of a set of 10 full-time teachers, then, the simple constraint-related metrics for such type of constraint (LWC) are as follows, *Constraints*: 2, *Points of application*: 18, *Soft penalty value*: 0, *Hard penalty value*: 18,000.

On the other hand, Table 4.2 presents 18 metrics related to the global structure of a CB-CTT instance. For example, number of days, number of classes to be scheduled, number of available rooms, and number of teachers included on an instance.

4.1.2 Solution Space Metrics

At its basic formulation, the CB-CTT problem is a combinatorial problem which requires performing two types of assignments: i) assigning lectures to time slots (task scheduling) and, ii) assigning resources to lectures (resource allocation). Although these assignment tasks are performed simultaneously, and must not be considered independent stages of the solution process, they work at different solution spaces, the first one related with *time slots* and, the second one, with *resources*. Therefore, they are referred to, by

[3]From the available 16 types of XHSTT constraints described in Table 1.1, Distribute Split Events Constraint and Order Events Constraint are not used by our instance generator.

Table 4.2: Simple structure-related metrics used to describe the size of the basic elements of an instance.

	Metric	Description
1	Time slots	Available number of time slots that are considered to solve an instance
2	Days	Number of days in which time slots are grouped
3	Shifts	Number of shifts in which time slots are grouped
4	Knowledge areas	Number of categories that defines the expertise areas of the teachers (e.g. math, engineering, management) required by the classes
5	Types of rooms	Number of room types required by the classes
6	Teachers	Available number of teachers that can be allocated
7	Full-time teachers	Available number of full-time teachers that can be allocated
8	Part-time teachers	Available number of part-time teachers that can be allocated
9	Rooms	Available number of rooms in which classes can be allocated
10	Curricula	Number of curricula in which classes are grouped
11	Curriculum times	Maximum number of weekly time slots that a curriculum can be allocated
12	Events	Total number of classes to be scheduled
13	Total events duration	Total duration of the classes to be scheduled
14	Preassigned times	Total number of time pre-assignments
15	Total resource requests	Total number of resources requests required by the classes
16	Preassigned resources	Total number of resource pre-assignments
17	Total points of application	Total points of application to be analyzed to evaluate the fulfillment of all constraints.
18	Total penalty value	Sum of the penalty values of all points of application

two shortened terms, as: *scheduling* (for task scheduling) and, *allocation* (for resource allocation).

According to Ochiai et al. [78], the optimization process performed by any algorithm or heuristic is a combination of searching tasks within feasible solution spaces. Therefore, describing the features of such spaces could lead to the development of better solution methods. To measure the dimensions of both solution spaces, a set of 44 metrics (16 scheduling space metrics and 28 allocation space metrics), based on the concept of counting functions, were formulated.

4.1.2.1 Scheduling Space Metrics

Within the CB-CTT context, *scheduling* consists on splitting the weekly duration of a class into a set of lectures to be appointed according to a set of time-related constraints.

The number of ways in which this process can be performed defines the *scheduling space* of an instance. For example, consider a three-hour literature class to be scheduled on a weekly basis. There are three possible configurations for scheduling its lectures: a single three-hour lecture (*3*), a combination of two-hour and one-hour lectures (*2-1*), and a set of three one-hour lectures (*1-1-1*). If eight available time slots per day, from Monday to Friday, are available to schedule this class, then, the number of possible ways (without repetition) to perform the scheduling of each configuration is, (*3*): 30; (*2-1*): 1,270; (*1-1-1*): 4,290. Hence, if no time constraints are further applied, the *non-constrained scheduling space* of this literature class is calculated by the sum of *non-constrained counting functions* that define the possible scheduling combinations for these configurations. For this example, a total of 5,590 possible combinations.

Non-constrained Scheduling Space

As described by Stanley [79], the goal of enumerative combinatorics is counting the number of elements of a finite set by formulating a *counting function* (S). Therefore, to count the number of possible ways in which the lectures of a class can be scheduled without constraints, we performed two tasks: *i)* identify the variables that define the *non-constrained counting functions* and, *ii)* model all the possible lecture-configurations for scheduling a class.

The variables that describe *non-constrained counting functions* are three. On the one hand, time slots per day (m) and class days per week (d) set the size of the *time grid* to schedule the lectures. On the other hand, lectures durations l_i (where i is the ith lecture) defines the lecture configuration being analyzed. For example, a one-lecture configuration with a duration of three time slots, must be defined with only one lecture duration: $l_1 = 3$; while a (2-1) two-lecture configuration, must be defined in terms of two lectures durations: $l_1 = 2$, $l_2 = 1$. Equation 4.1 defines the *non-constrained counting function* for one-lecture configurations, which are the simplest of all.

$$S = d(m - l_1 + 1) \tag{4.1}$$

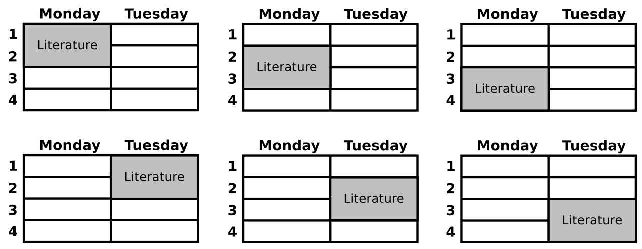


Figure 4.3: Graphical representation of the six possible ways in which a two-time slot lecture can be scheduled within a time grid of two days by four time slots.

Figure 4.3 illustrates an example to which this counting function can be applied. The diagram represents the six possible ways in which a two-time slot literature lecture can be scheduled within a set of four time slots, from Monday to Tuesday.

Equation 6.1 defines the *non-constrained counting function* for two-lecture configurations. As noticed, two additional variables are included: an index j, used as a counter to perform the summation, and a variable (u), which counts the number of lectures that has the same duration, as lectures of equal duration generate schedule arrangements with repetition that must be also evaluated.

$$S = d\left[(d-1)\prod_{i=1}^{2}(m-l_i+1) + \left(2\sum_{j=1}^{m-l_1-l_2} j\right)\right] \Big/ u! \tag{4.2}$$

Figure 4.4 illustrates an example to which this counting function can be applied. The diagram represents the twenty two possible ways in which a *(1-1)* lecture configuration can be scheduled within the same eight-time slots time grid shown in Figure 4.3.

As observed in these two equations, if the number of lectures to be considered in a lecture configuration increases, the calculation process of its *non-constrained counting function* becomes more complicated, since it requires a larger set of possible scheduling combinations. Although the equations for three, four and five lecture configurations are not presented in this section, they were deduced and applied in a similar manner.

Constrained Scheduling Space

Whenever a time-related constraint is applied to an instance, its scheduling space is reduced depending on the particular conditions modeled by the constraint. For example,

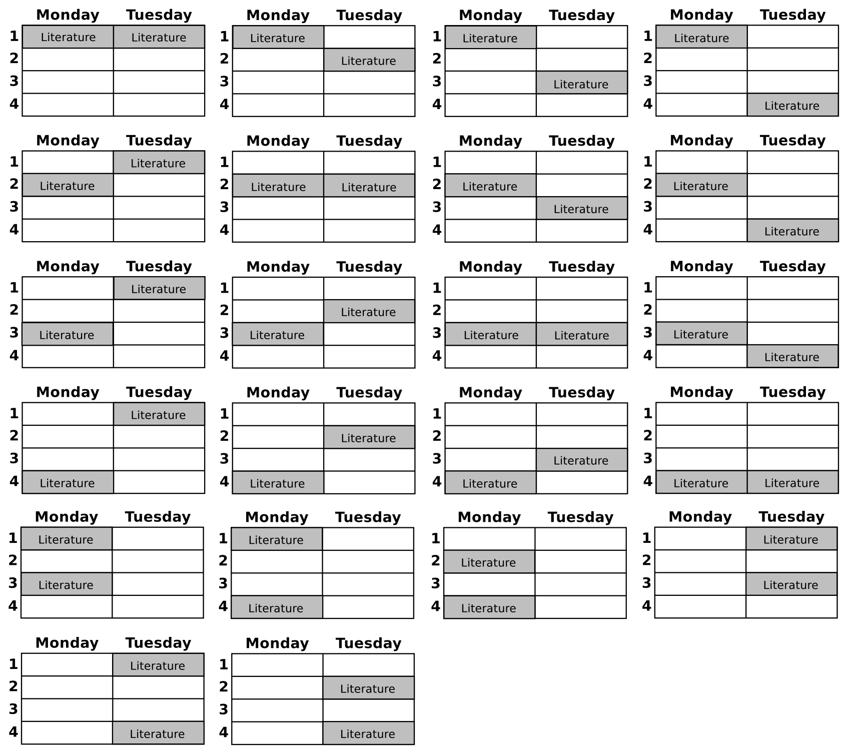


Figure 4.4: Graphical representation of the twenty two possible ways in which two one-time slot lectures can be scheduled within a time grid of two days by four time slots.

if a constraint is applied to the *non-constrained scheduling space* illustrated in Figure 4.4 to require that only one lecture be scheduled per day, then the size of the *scheduling space* would be reduce from 22 to 16, because the last six lecture-configurations of the figure would be considered violations of this constraint.

In this thesis, the reduction of each constraint over the *non-constrained scheduling space* of a CB-CTT instance is quantified in three different ways:

- **Space**: Counts the remaining size of the scheduling space after applying *only* a certain type of constraint.
- **Removed**: Counts the number of combinations that are removed from the scheduling space after applying *only* a certain type of constraint.
- **Percentage**: Calculates the percentage in which the scheduling space is reduced after applying *only* a certain type of constraint.

In summary, the set of 16 scheduling space metrics consist of: 1 *non-constrained scheduling space* metric that aggregates the initial *scheduling spaces* of the classes of an instance without limiting the number of ways in which the scheduling task can be performed; 12 scheduling reduction metrics that measure the shrinkage of the scheduling space due to each one of the applied XHSTT constraints (described in Table 1.1) related to the scheduling space (i.e., Split Events Constraint, Prefer Times Constraint, Spread Events Constraint, and Link Events Constraint); and 3 scheduling constrained metrics that measure the overall net effect of the constraints on reducing the initial scheduling space (i.e., *constrained scheduling space, constrained scheduling removed* and *constrained scheduling percentage*). Out of the 44 solution space metrics, 16 are scheduling space metrics.

4.1.2.2 Allocation Space Metrics

Within the CB-CTT context, allocation is the assignment of the available resources to the lectures that require them. The number of ways (defined by a set of resource-related constraints) in which this assignment can be performed is what defines the *allocation space.*

To illustrate the calculation of the *non-constrained allocation space*, consider a set of three classes to be scheduled four time slots a week (weekly duration, w), each one requiring a full-time teacher. If there are three possible teachers (resources, r) to be assigned to such classes, and the allocation time grid consists of twenty time slots (ts) (equally distributed in a five-day week). Then, the *non-constrained counting function* for the possible allocation of resources is calculated using Equation 4.3.

$$S = (w)(r)(ts) \tag{4.3}$$

Figure 4.5 shows the described example. Each class can be assigned to the twenty available time slots of each teacher; therefore, its number of possible allocations is calculated as follows: $S = (4)(3)(20) = 240$. As the number of classes to be allocated is three, the total *non-constrained allocation space* for this example is calculated as: $240 + 240 + 240 = 720$.

Similarly to the *scheduling space*, the *unconstrained allocation space* is reduced if any

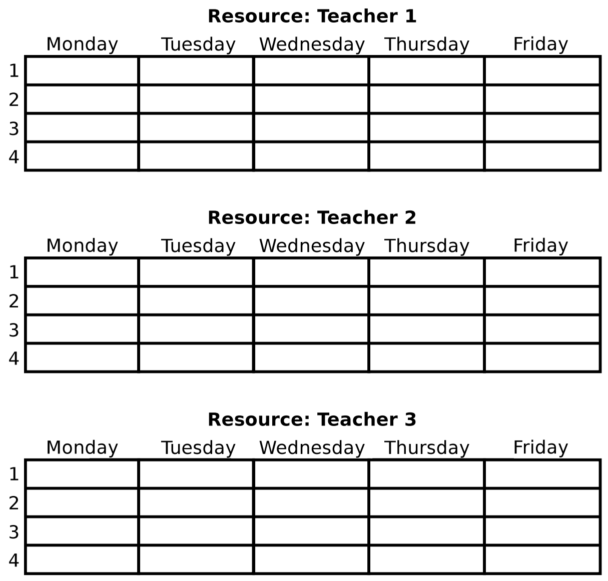


Figure 4.5: Graphical representation of the timetable of three teachers considering a time grid of five days by four time slots.

of the resource-related constraints, described in Table 1.1, are applied to an instance. These are: Prefer Resources Constraint, Avoid Split Assignments Constraint, Avoid Clashes Constraint, Avoid Unavailable Times Constraint, Limit Idle Times Constraint, Cluster Busy Times Constraint, Limit Busy Times Constraint, and Limit Workload Constraint. For example, if the Limit Workload Constraint is applied to limit to 15 the number of hours that teachers can work per week, the allocation space would be reduced from 240 to 180.

In summary, the set of 28 allocation space metrics consists of: 1 *non-constrained allocation space* that aggregates all the possible allocations of resources to classes, 24 allocation reduction metrics, and 3 allocation constrained metrics (i.e., *constrained allocation space*, *constrained allocation removed* and *constrained allocation percentage*). Out of the 44 solution space metrics, 28 are allocation space metrics.

4.1.3 Feature Ratios

Ratios are simple mathematical (statistical) calculations used to express relationships and perform meaningful comparisons between numerical data. In the collection of CB-CTT metrics, we included 17 feature ratios formulated with two purposes: *i)* describe potential significant relationships between the sets of *simple* and *solution space* metrics,

Table 4.3: Description of the 17 feature ratios included in the set of CB-CTT metrics.

	Metric	Description
1	Total rooms conflicts	Sum of the ratios of required time slots per type of room to available time slots per type of room
2	Total teachers conflicts	Sum of the ratios of required time slots per type of teacher to available time slots per type of teacher
3	Total curricula conflicts	Sum of the ratios of required time slots per curriculum to available time slots per curriculum
4	Total resources conflicts	Sum of total room conflicts, total teacher conflicts and total curricula conflicts
5	Average rooms conflicts	Average ratio of required time slots per type of room to available time slots per type of room
6	Average teachers conflicts	Average ratio of required time slots per type of teacher to available time slots per type of teacher
7	Average curricula conflicts	Average ratio of required time slots per curriculum to available time slots per curriculum
8	Average resources conflicts	Total resources conflicts divided by the total number of resources
9	Rooms weighted conflicts	Weighted average ratio of required time slots per room type to available time slots per type of room
10	Teachers weighted conflicts	Weighted average ratio of required time slots per teacher type to available time slots per type of teacher
11	Percentage of assigned resources	*Preassigned resources* divided by the *Total resource requests*
12	Average event duration	*Total events duration* divided by the number of *Events*
13	Scheduling space shrinkage	*Constrained scheduling space* divided by the *non-constrained scheduling space* (*ATC space*)
14	Allocation space shrinkage	*Constrained allocation space* divided by the *non-constrained allocation space* (*ARC space*)
15	Average constraint penalty	*Total penalty value* of the instance divided by the *total points of application* defined by its constraints
16	Average allocation space per resource	*Constrained allocation space* divided by the number of *resources*
17	Average scheduling space per event	*Constrained scheduling space* divided by the number of *events*

described above, and *ii)* analyze conflicts of resources by comparing the requirement and availability of resources. The 17 feature ratios are described in Table 4.3.

4.1.4 Constraint Density

In addition to the three sets of metrics that were formulated for this thesis in the previous subsections, a set of 14 constraint density indexes defined by Kingston [68], as part of the *KHE timetabling engine*[4], was included into the collection of CB-CTT metrics.

As defined by Kingston, the *density* of each type of XHSTT constraint is calculated

[4]As explained in Section 3.3.5, the KHE Timetabling Engine was the solver applied to generate the initial solutions for the generated CB-CTT instance dataset.

dividing the number of elements to which a constraint is applied by the number of elements to which this constraint could be applied. For example, if the constraint *Assign Time Constraint* (ATC) is set to require that 80 of a total of 100 lectures defined in an instance be scheduled, then the density of this constraint is calculated as: Total lectures to which ATC is applied divided by the possible lectures to which ATC could be applied. In this case, the ATC density = 80/100.

The density of each constraint is calculated based on different elements, according to the condition it models. These elements could be times, resources or events of an instance.

4.2 Feature Selection

The collection of CB-CTT metrics described in the previous section calculates different properties that are expected to be good predictors of the empirical hardness of CB-CTT instances. To verify the relevance of those metrics, we now describe the four processes involved in the feature selection methodology presented in Figure 4.2.

The main goal of feature selection is reducing data dimensionality without losing much of the original information. In theory, having more features should produce better learning models. But as stated by Yu & Liu [80], in practice, excessive features might lead to higher learning times and over-fitted models. Therefore, finding the optimal subset of features to enhance learning efficiency and prediction accuracy is a critical task.

To obtain the optimal subset of relevant of CB-CTT metrics to characterize the empirical hardness of the instance dataset (described in Section 3.3), three sequential processes were applied: *instance description*, *performance prediction*, and *metrics assessment* (see Figure 4.2). Then, these relevant CB-CTT metrics, were interpreted to provide a better understanding of the conditions that make CB-CTT instances hard to solve.

Table 4.4: Attribute-value format to describe the generated CB-CTT instance dataset.

Instances	**Metrics**			**Performance**
	$\mathbf{M_1}$	$\dots$	$\mathbf{M_{125}}$	
I_1	$m_{1,1}$	$\dots$	$m_{1,125}$	p_1
I_2	$m_{2,1}$	$\dots$	$m_{2,125}$	p_2
I_3	$m_{3,1}$	$\dots$	$m_{3,125}$	p_3
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
I_{6000}	$m_{6000,1}$	$\dots$	$m_{6000,125}$	p_{6000}

4.2.1 Instance Description

As the first process of the feature selection methodology, the 6,000 CB-CTT generated instances (where each instance is an XML tree), were described in terms of the CB-CTT metrics presented in Section 4.1. All values were normalized to have the same magnitude range, and as a preprocessing step, two types of features were removed: i) uni-valued features (that have the same value in all instances), and ii) perfectly correlated features (that provide the same information). After this elimination, the number of metrics was reduced from 149 to 125.

The generated instances were described using the attribute-value format shown in Table 4.4. In this table, each row represents a CB-CTT instance I_i (where $1 \leq i \leq 6,000$) and each column M_j represents a metric or feature (where $1 \leq j \leq 125$) that has value m_{ij} for each instance. The value *Performance*, indicates the empirical hardness of an instance, measured as the *penalty value* of the initial solution generated with the KHE timetabling engine (as explained in Section 3.3.5).

4.2.2 Performance Prediction

The analysis of the the data summarized in the attribute-value format (see Table 4.4) was performed based on a regression approach that employs supervised learning methods to assess the relevance of the collection of CB-CTT metrics to predict the penalty value of the instances.

For the regression task, eight regression algorithms from the scikit-learn library of

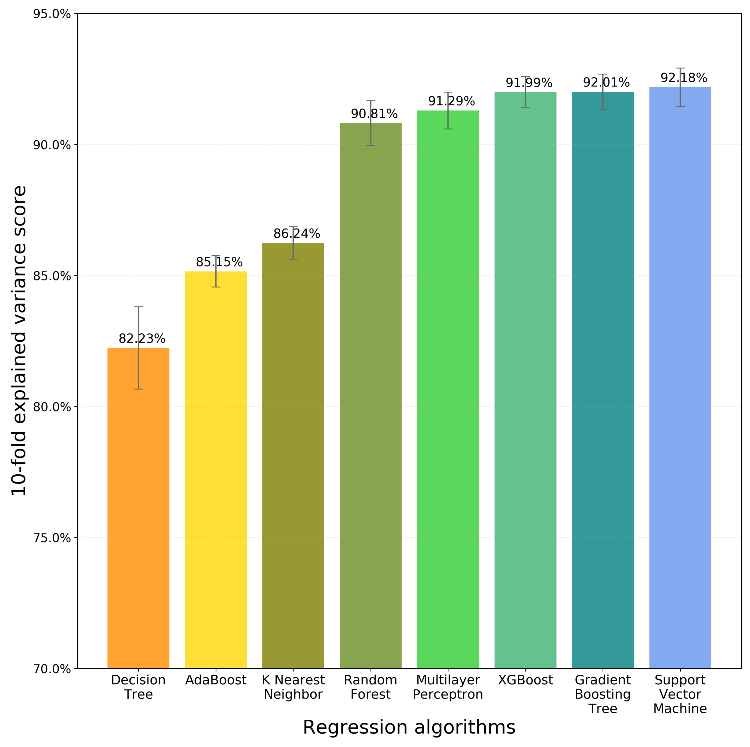


Figure 4.6: Explained variance score of machine learning regressors applied to predict the penalty value of the generated XHSTT dataset using a 10-fold cross-validation.

Python [81] were tested[5]: Decision Tree, AdaBoost, K-Nearest Neighbor, Random Forest, Multilayer Perceptron, XGBoost, Gradient Boosting Tree, and Support Vector Machine. To avoid the risk of drawing wrong conclusions from over-fitted models, their performance was measured in terms of the *explained variance score* of a 10-fold cross-validation process. The explained variance score is a measure employed to compare the predictions ($\hat{y}$) made by a regression model and the values of an output variable (y) to evaluate the *prediction error* ($y - \hat{y}$). The explained variance score is calculated as:

$$explained\ variance\ score = 1 - \frac{Variance[y - \hat{y}]}{Variance[y]}$$

Figure 4.6 shows the explained variance score (from lower to higher) of the eight learning algorithms. As observed, they exhibit a significant disparity on their cross-validated explained variance score —measured in the 10 testing folds— that ranges from 82.23% (obtained by Decision Tree), to 92.18% (obtained by Support Vector Machine).

[5]The description of the regression models and their default parameters can be found in the official documentation of scikit-learn at `https://scikit-learn.org/stable/supervised_learning.html`

The best five of these algorithms (Random Forest, Neural Network, XGBoost, Gradient Boosting Tree, and Support Vector Machine), although different in the mathematical methods in which they are based on, achieved a similar performance. This indicates that the explained variance score for the regression models that can be built from this CB-CTT metrics is around 91%. Having statistical evidence about the accuracy of the empirical models built from the proposed CB-CTT metrics, the models with the highest predictive performance were selected to evaluate the relevance of the metrics.

4.2.3 Metrics Assessment

As shown in Figure 4.2, the output of the feature selection processes is a subset of *Relevant CB-CTT metrics* able to characterize the generated instances with a predictive power similar to that of the full set of *CB-CTT metrics.* To determine this subset, the contribution of each metric to the prediction accuracy of the best empirical models was assessed.

As pointed out in recent research works [82, 83], even though learning algorithms have been adopted to build prediction models in a wide range of applications, most of the time they are implemented as black boxes. They do not provide an understanding of the problems being modeled. Model interpretation is a problem in itself, whose difficulty depends on the method used in its construction. Some methods are easily explainable since their results can be explicitly tracked to their components, while some others, such as neural networks, require the construction of auxiliary explanation techniques [83].

From the five prediction algorithms, shown in Figure 4.6, three are based on decision trees (i.e., Random Forest, XGBoost, Gradient Boosting Tree), and one is difficult to explain (i.e., Multilayer Perceptron). Therefore, for the assessment of the metrics, two regression algorithms were selected: *i)* Support Vector Machine (SVM) based on linear kernels, the one with the highest cross-validated explained variance score, and *ii)* Gradient Boosting (GB) Tree built from 100 regression trees, the best of the decision tree-based algorithms.

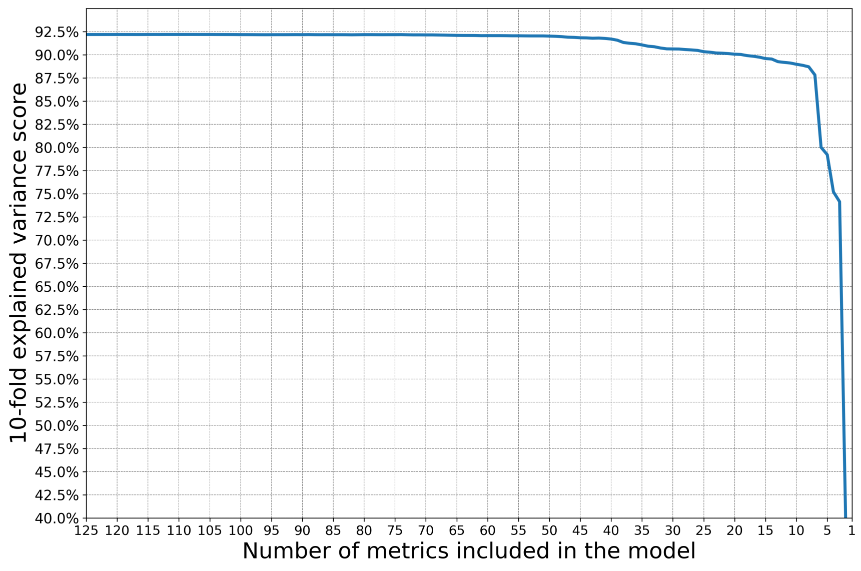


Figure 4.7: 10-fold cross-validated explained variance score of SVM-based models built from a different number of features (CB-CTT metrics).

4.2.3.1 Support Vector Machines

The metrics, or features, that could be considered as relevant from SVM-based models were obtained by applying a *Recursive Feature Elimination* method (RFE) [84]. This method follows the *Sequential Backward Generation* approach (SBG) to iteratively remove, based on a cross-validated evaluation, the least important of a set of features.

Figure 4.7 shows the cross-validated explained variance score of the SVM-based models built at each iteration of the RFE method. As observed, the explained variance score decreased significantly when only a few metrics were included in the models. Therefore, to statistically determine the minimum number of required metrics, we performed a time series analysis based on a moving average of 20-metric windows with a convergence threshold of 0.5% for the root mean square deviation. According to these calculations, we determined that at least 31 metrics were required to build SVM-based models as accurate as the ones that uses the full set of metrics. Therefore, this number of metrics was defined to perform the coefficient feature analysis described next.

At its most basic concept, an SVM defines a set of hyperplanes in an n-feature dimensional space to maximize the margin distance between vectors that belong to different classes. However, to predict continuous-valued outputs in regression tasks, this

Table 4.5: Set of 31 relevant CB-CTT metrics selected from the analysis of SVM-based models, organized according to the type of metric. The number indicates how relevant the metric is (one been the most relevant).

Simple	Solution Space	Feature Ratios
(1) Timeslots	(11) SPEC space	(17) Total rooms conflicts
(2) Shifts	(12) Constrained scheduling space	(18) Total curricula conflicts
(3) Teachers	(13) ASAC space	(19) Total resources conflicts
(4) Rooms	(14) ACC space	(20) Average curricula conflicts
(5) Total resources requests	(15) AUTC space	(25) Average allocation space per resource
(6) Total events duration	(16) Constrained allocation space	
(7) Preassigned resources	(23) PRC removed	
(8) ARC hard penalty value	(24) ASAC removed	
(9) ASAC hard penalty value	(27) AUTC removed	
(10) AUTC hard penalty value	(31) SPEC percentage	
(21) Full-time teachers		
(22) Curricula times		
(26) Types of rooms		
(28) LWC hard penalty value		
(29) ACC hard penalty value		
(30) LWC points of application		

margin distance is often optimized with Vakpnik's ε-insensitive loss functions.

As stated by Guyon et al. [84], a straightforward method to determine the importance of its related features is by comparing the relative size of the n-coefficients in the hyperplanes. By performing this comparison at each one of randomly-generated 10 training-testing folds, we selected the set of metrics that better predicted the penalty values of the instances.

Table 4.5 presents the set of 31 relevant CB-CTT metrics, organized according to the type of metric, as defined in Section 4.1. Each metric is numbered in ascending order, from most to least relevant, according to the coefficients analysis of the support vector machines.

4.2.3.2 Gradient Boosting Trees

As with SVM-based models, the minimum number of metrics required to build GB trees with a similar accuracy as using all metrics, was determined by the implementation of an RFE method that follows an SBG approach. By applying the same convergence threshold to the cross-validated accuracy of the GB trees (plotted in Figure 4.8), it was defined 16 as the minimum number of required metrics.

As stated by Hastie et al. [85], the motivation for *boosting* is combining the outputs of many "weak" classifiers (whose error rate is only slightly better that a random

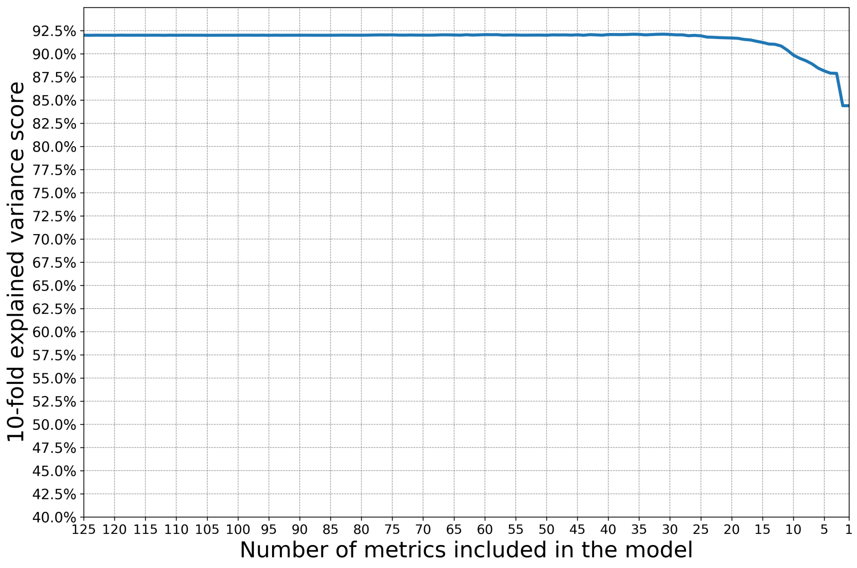


Figure 4.8: 10-fold cross validation explained variance score of GB trees constructed at each step of the RFE method.

guessing) to produce a powerful "committee". That is, a sequence of weak classifiers $G_m(x), m = 1, 2, ..., M$, that combined through a weighted voting function produce the final prediction.

When applied to regression tasks, boosting approaches combine predictors sequentially to minimize the loss function defined to quantify the prediction errors. At each iteration, error residuals are analyzed and minimized by adding a new predictor to the "committee" until error residuals become constant.

In the applied GB tree implementation, predictors consist of individual decision trees that split according to the mean squared error defined by Friedman [86], and are fitted to the data based on purity scores to minimize the sum of the prediction squared errors.

To evaluate the relevance of the metrics in the GB trees, a hierarchical approach was adopted. First, the relevance of the metrics at the individual trees was calculated by determining the weighted reduction in node purity from the split at each node. Next, the relevance of the metrics in all trees were averaged.

Table 4.6 presents the set of 16 relevant metrics, organized according to the type of metric as defined in Section 4.1. Each metric is numbered in ascending order, from most to least relevant, according to the node purity analysis of the GB trees.

Table 4.6: Set of 16 relevant metrics selected from the analysis of GB trees, organized according to the type of metric. The number indicates how relevant the metric is (one been the most relevant).

Simple	Solution Space	Feature Ratios	Constraint Density
(1) Total events duration (2) Preassigned resources (3) AUTC hard penalty value (4) LITC hard penalty value (5) LWC hard penalty value (15) LEC points of applications	(10) PRC removed (13) AUTC space (14) LEC space	(7) Total curricula conflicts (8) Total resources conflicts (9) Average event duration (11) Percentage of assigned resources (12) Average curricula conflicts (16) Rooms weighted conflicts	(6) AUTC density

Table 4.7: Set of 9 common metrics, according to their mean observed relevance in the previous two sets.

Simple	Solution Space	Feature Ratios
(1) Total events duration (2) Preassigned resources (3) AUTC hard penalty value (8) LWC hard penalty value	(6) AUTC space (9) PRC removed	(4) Total curricula conflicts (5) Total resources conflicts (7) Average curricula conflicts

4.3 Relevant CB-CTT Metrics

From the metric assessment process, two sets of relevant CB-CTT metrics were defined: a set of 31 metrics based on the analysis of SVM and a set of 16 metrics based on the analysis of GB trees. To analyze the similarity of these sets, we defined a third set including their common metrics. As a result, we considered three sets of relevant metrics to be further evaluated and interpreted:

- **SVM metrics:** Set of 31 relevant metrics selected from SVM-based models.
- **GBtree metrics:** Set of 16 relevant metrics selected from GB trees.
- **Common metrics:** Set of 9 metrics determined by the intersection of *SVM metrics* and *GBtree metrics*. This set of metrics is presented in Table 4.7 and was numbered, from most to least relevant, according to their mean observed relevance in the previous two sets. As observed, this set does not include any of the constraint density indexes proposed by Kingston (described in Section 4.1.4), as none of them proved to be consistently relevant in the previous two sets of metrics.

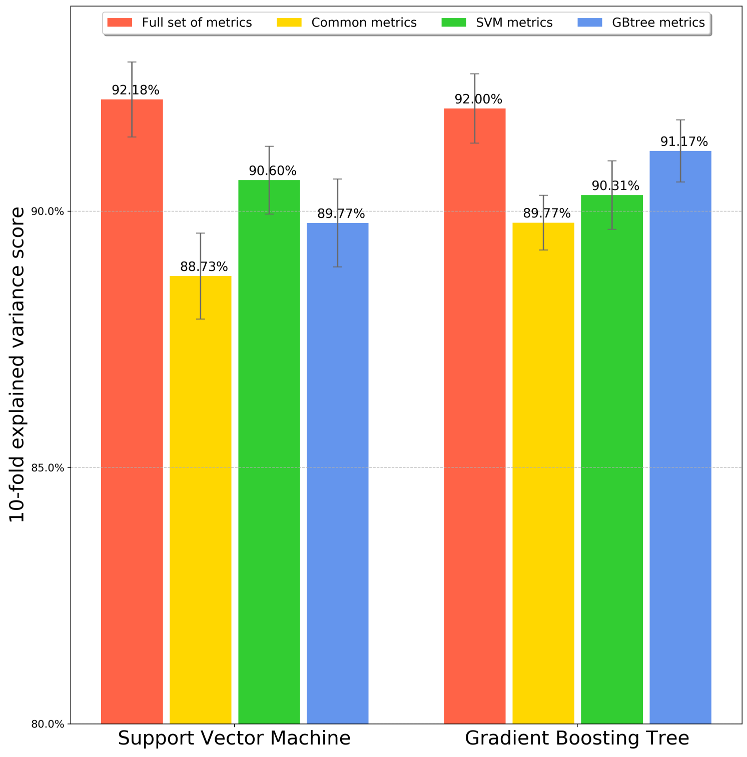


Figure 4.9: 10-fold explained variance score of SVM and GB trees regressors applied to predict the penalty value of the CB-CTT instance dataset using the three defines sets of relevant metrics (the full set of 149 metric is shown for comparison purposes).

Each set of relevant metrics include a different fraction of the original set of 149 metrics. Hence, to evaluate the information loss due to the dimensionality reduction, we compared the prediction effectiveness of the three sets to that obtained with the full set of 149 metrics.

Figure 4.9 shows the explained variance score of the penalty value predicted with each set of relevant metrics. The predictions were made using SVM and GB tree regression models and evaluated with a 10-fold cross-validation approach. As observed in the figure, on the one hand, the sets of Common metrics and GB tree metrics exhibited a higher prediction performance in GB tree models than in SVM models. However, they also showed a higher performance deviation between both regression approaches. On the other hand, SVM metrics achieved a similar performance in both types of regression approaches, suggesting more stable prediction capabilities when used to build prediction models based on a different set of learning algorithms.

In the presented comparison, the lower prediction effectiveness of the set of Common metrics is evident, but the difference between the sets of SVM metrics and GBTree metrics is not clear. For this reason, both SVM metrics and GB metrics will be considered for the creation of or per-instance algorithm model, and further interpreted in the next section to provide a better understanding about the difficulty of solving CB-CTT instances.

4.4 Metrics Interpretation

Despite the importance to map instance properties of academic timetabling problems with the performance of available solving methods [87], few research works have proposed features to describe the empirical hardness of timetabling instances [88–90]. To fill this gap, in this chapter, we defined three sets of *relevant CB-CTT metrics* as potential predictors of timetabling empirical hardness, but, based on a comparison of their information loss, we selected two sets —SVM metrics and GBtree metrics— to be considered for the construction of our per-instance algorithm selection model.

As a contribution to the current state-of-the-art, this section presents the final step of the feature selection methodology shown in Figure 4.2, metrics interpretation. Our goal is to discuss the relationships between the selected relevant CB-CTT metrics and the empirical hardness of the generated CB-CTT instances.

Table 4.8 shows the union of the set of SVM metrics and GBtree metrics, a total of 38 metrics, grouped according to three aspects: i) size, ii) scheduling (time assignment), and iii) allocation (resource assignment). In this table, the nine metrics common to both sets are marked with an asterisk (*). Based on this data, next, we present an interpretation of the main factors that make CB-CTT instances hard to solve.

4.4.1 Size-related Hardness

The first column of Table 4.8 shows the 12 metrics related to the size of CB-CTT instances. These metrics characterize the structure of the instances as regards to its basic dimensions (e.g., number of time slots, number of teachers, number of rooms). In

Table 4.8: Set of 38 metrics obtained from the union of the sets of SVM metrics and GBtree metrics. For interpretation purposes, these metrics are grouped according to three aspects: size, scheduling, and allocation. The nine metrics common to both sets of relevant metrics are marked with an asterisk (*).

Size	**Allocation**	**Scheduling**
-Timeslots	-ARC hard penalty value	-LEC points of application
-Shifts	-ASAC hard penalty value	-LEC space
-Teachers	-ASAC removed	-SPEC space
-Full-time teachers	-LITC hard penalty value	-SPEC percentage
-Rooms	-LWC hard penalty value*	-Constrained scheduling space
-Types of rooms	-LWC points of application	
-Total events duration*	-ACC hard penalty value	
-Curricula times	-ACC space	
-Average event duration	-ASAC space	
-Total resources requests	-PRC removed*	
-Preassigned resources*	-AUTC hard penalty value*	
-Percentage of assigned resources	-AUTC space*	
	-AUTC removed	
	-AUTC density	
	-Constrained allocation space	
	-Average allocation space per resource	
	-Total room conflicts	
	-Rooms weighted conflicts	
	-Total curricula conflicts*	
	-Average curricula conflicts*	
	-Total resources conflicts*	

this list, two metrics are common to SVM and GBtree sets and particularly descriptive of the assignments required to solve an instance; these are: *i) Total events duration*, *ii) Preassigned resources*.

On the one hand, *Total events duration* measures the number of time slots required to schedule all the lectures of an instance. On the other hand, *Preassigned resources* measures the number of resources assigned prior to the solution process of an instance, which reduces the required amount of resource allocations. These metrics are ranked as important in both sets of metrics and are likely to be good indicators of the assignment tasks required in both solution spaces (described in Sections 4.1.2.1 and 4.1.2.2).

4.4.2 Allocation Hardness

As pointed out in recent surveys [3, 10–12], most of the constraints that are commonly defined to formulate timetabling problems are related to allocating resources to lectures. Therefore, the fact that the majority of the metrics included in the Table 4.8 are mainly related to the *allocation space* is not surprising.

The size of the *allocation space* of an instance has a crucial role in defining the hardness of an instance. This is evident because of the relevance of the metric *Constrained allocation space*, and of other *solution space metrics* that measure the reduction in the initial number of feasible <class, resource> combinations, due to the set of resource-related constraints applied to the instances.

Besides the basic *Avoid Clashes Constraint* (ACC) and *Avoid Split Assignment Constraint* (ASAC), three types of allocation-related constraints are recurrent on the sets of relevant metrics and proved to be descriptive of the allocation space:

- *Avoid Unavailable Time Constraints* (AUTC): Are applied to model two conditions: *working shifts* and *study shifts.* These constraints limit the time slots in which teachers and curricula can attend to classes. For example, when two shifts (morning and evening) are defined for an instance, the allocation space of teachers and curricula are reduced by half.

- *Limit Workload Constraints* (LWC): Are applied to model three conditions: *daily workload of full-time teachers*, *daily workload of part-time teachers*, and *daily workload of curricula.* These constraints define the maximum number of time slots a day both teachers and curricula can attend to classes.

- *Prefer Resources Constraints* (PRC): Are applied to model two conditions: *prefer teacher* and *prefer room.* These constraints specify the type of teachers and rooms required for each class (event), thus reducing the set of possible resources to be allocated.

In addition to these metrics, five metrics (defined in Section 4.1.3) associated with the concept of *slackness* are included in the second column of Table 4.8: *Total room conflicts*, *Rooms weighted conflicts*, *Total curricula conflicts*, *Average curricula conflicts*, and *Total resources conflicts.* In each one of them, the *slackness* (that measures the easiness of allocating a resource to the lecture that requires it), is calculated by dividing the demand of a resource by its available supply. For example, if 21-hour literature lectures are to be weekly scheduled, and there is only one literature teacher with a defined weekly workload of thirty hours a week, the slackness for this allocation task is calculated as 21/30. Therefore, the higher the slackness, the greater the hardness of allocating a resource.

4.4.3 Scheduling Hardness

Similarly to the allocation space, the size of the *scheduling space*, that aggregates the number of possible assignments of lectures to time slots, proved to be a crucial feature to define the hardness of the generated CB-CTT instances. This is evident because of the inclusion of the *Constrained scheduling space* in the third column of Table 4.8, which measures the reduction of the initial *scheduling solution space* mainly due to the application of the time-related constraints described next:

- *Spread Events Constraints* (SPEC): Are applied to model two conditions: *single lecture* and *daily lecture.* On the one hand, *single lecture* requires that only one lecture of each class be scheduled per day. On the other hand, *daily lecture* requires that the lectures of a set of randomly selected classes be scheduled on a daily basis. These constraints were the main factor of the reduction in the number of valid combinations of the scheduling space, up to a percentage between 66.4% and 89.7% in half of the generated instances.

- *Link Events Constraint* (LEC): Is applied to model the *link events* condition, which requires that all lectures of a set of classes be scheduled simultaneously. Unlike SPEC constraints, this constraint does not reduce the scheduling space of the instances in a high proportion. However, it causes an increase in the complexity of the *layer tree* structure, used to handle time assignment by the KHE solver [68].

4.5 Summary

In this chapter, we described the feature selection-based methodology employed to identify the set of features that better characterizes the CB-CTT instance dataset generated for this thesis. To did so, we described the generated instances using four sets of *CB-*

CTT metrics that were formulated to predict the *penalty value* of the instances through regression models constructed from different machine learning methods.

From the analysis of the most accurate regression models, three sets of relevant features were compared, and as shown in Figure 4.9, they demonstrated to be almost as precise as the full set of metrics to predict the *penalty value* of the instances. As a result, 38 metrics (listed in Table 4.8) were defined to be used for the construction of our per-instance algorithm selection model.

Finally, in the last section, the most relevant features that define the initial solution hardness of the instances were also interpreted as an answer to the question: *"What are the main factors that make CB-CTT timetabling instances hard to solve?"* This feature interpretation led us to conclude that the hardness of CB-CTT instances is highly related to the percentage of reduction of the scheduling and allocation spaces (measured based on counting functions), and the slackness of resources (measured based on feature ratios).

Chapter 5

Portfolio of Meta-heuristics

In Chapter 3, we described the instance generator, and the generation process followed to create the Curriculum-Based Course Timetabling (CB-CTT) instance dataset. All instances were validated using the KHE timetabling engine [68], a popular solver able to obtain good initial solutions for timetabling problems represented according to the XHSTT data format [22] used in this thesis. The diversity of the dataset was evaluated based on the initial solving difficulty of the instances, in five categories, ranging from *Very Easy* to *Very Hard*. The KHE engine was able to find optimal solutions for 321 of the *Very Easy* instances. Hence, the solutions of 5,679 instances remained to be improved by the application of additional solution methods.

A popular computational approach applied to improve the solutions of optimization problems is the implementation of meta-heuristics. In a general way, a meta-heuristic is any kind of solution strategy that makes use of a searching method to avoid being stuck in local optima when looking for an optimal solution. Within combinatorial optimization, meta-heuristics have proved to be effective methods to find good solutions in reasonable times. Therefore, to improve the 5,679 non-optimal solved instances in the CB-CTT dataset, we selected four meta-heuristics to build our algorithm portfolio.

Following the meta-learning framework, shown in Figure 5.1, in this chapter, we describe the meta-heuristics included in our algorithm portfolio and evaluate their performance regarding two important aspects: *i*) their effectiveness to improve the initial

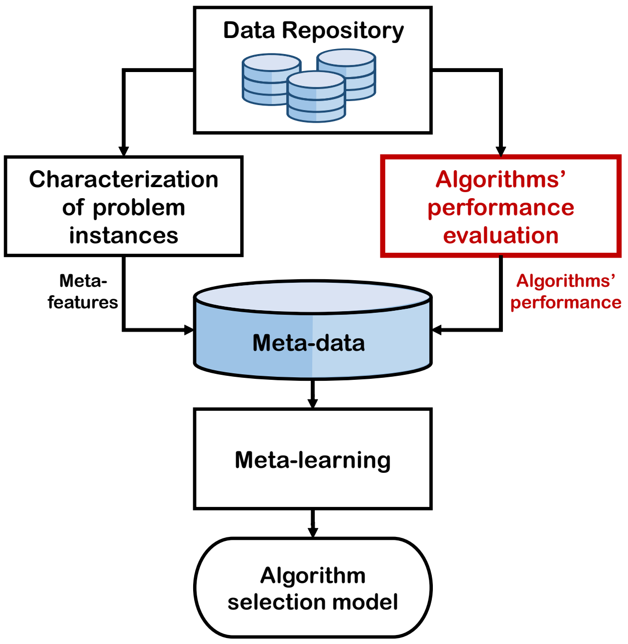

Data Repository
Characterization of problem instances
Algorithms' performance evaluation
Meta-features
Algorithms' performance
Meta-data
Meta-learning
Algorithm selection model

Figure 5.1: Meta-learning framework for the construction of the CB-CTT per-instance algorithm selection model (adapted from [2]). For reference purposes, the elements *Algorithms' performance evaluation* and *Algorithms' performance*, related to the *Algorithm space* ($\mathcal{A}$), are highlighted in red.

solutions generated with the KHE solver, and *ii*) their performance variation across the problem space.

5.1 Meta-heuristics in Combinatorial Optimization

When applied to combinatorial optimization problems, meta-heuristics explore solution spaces through the application of *low-level heuristics* that *perturb* the array of elements of an initial solution s_0 in order to improve it. Each low-level heuristic (LLH) produces a different set of solutions that shares a similar structure within a neighborhood $N_k(s)$, where k is the type of LLH applied to a given solution s.

The basic version of a meta-heuristic starts by perturbing an initial solution s_0 within a neighborhood $N_k(s)$ to produce another solution s'. If a better solution is produced, the previous solution is replaced. The search continues, in the same or in a different neighborhood, until a stop condition is met.

For illustration purposes, in Figure 5.2, the maximization search process of a meta-heuristic is exemplified. As observed, at each iteration, a new solution (s_n), with a

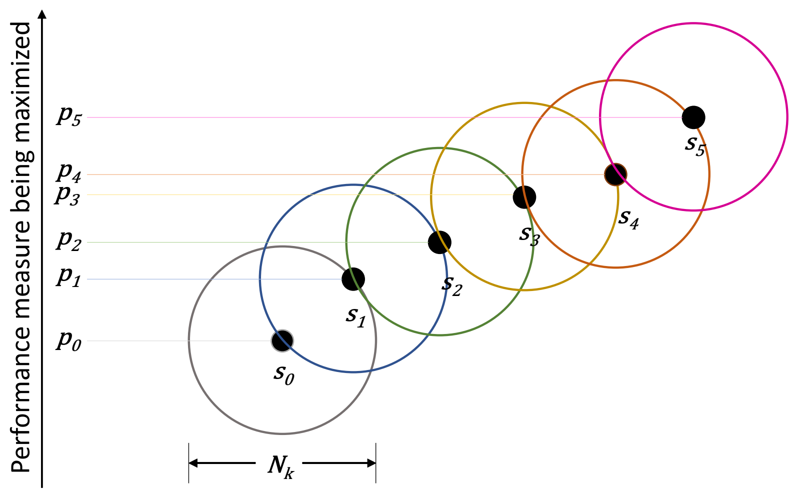


Figure 5.2: Illustration of an optimization search process employed by a meta-heuristic in the combinatorial domain. At each iteration, a new solution s_n it is generated within a neighborhood N_k to maximize a performance metric p_n.

higher performance value than the previous one (s_{n-1}), is obtained within a defined neighborhood (N_k), represented by a circle. As a result, the final solution (s_5) exhibits higher performance value than the initial solution (s_0).

As can be implied, the effectiveness of a meta-heuristic depends on solution strategies working at two different levels: *i)* a set of *low-level heuristics* (*LLHs*) able to perturb, in diverse manners (i.e., within different neighborhoods), the array of the elements of a given solution, and *ii)* an *exploration strategy* to perform these perturbations in promising solution regions without being stuck in local optima.

5.2 CB-CTT Meta-heuristics

It is important to recall that the general goal of this thesis is not to develop novel solution methods for solving CB-CTT instances but to integrate existing solution methods into a per-instance algorithm selection model able to predict the best solution method for a given CB-CTT instance.

As explained in Section 3.3.5, in the current state of the art, there are two type of solvers proposed to solve timetabling instances in the XHSTT data format: *i)* solvers that *generate* initial solutions and *ii)* solvers that *improve* initial solutions. To generate the initial solutions for our CB-CTT instance dataset, we applied the KHE timetabling

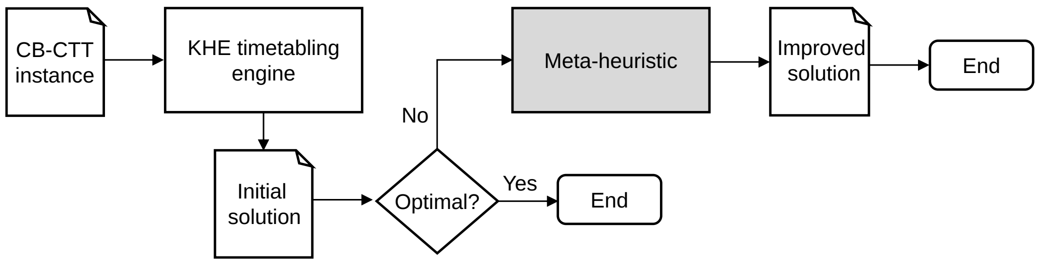


Figure 5.3: Illustration of the general process defined for solving CB-CTT instances.

engine; to improve those initial solutions, we selected the set of four meta-heuristics described in this section.

Figure 5.3 illustrates the general process defined to integrate different solvers for the solution of the CB-CTT instance dataset. As observed, each instance is first solved with the KHE timetabling engine; if the solution obtained is optimal (i.e., has a penalty value of zero), the process ends; if the solution obtained is not optimal, a meta-heuristic is applied to improve, as much as possible, the quality of the initial solution. Our per-instance algorithm selection model focus on the second step of the process (colored in gray); it aims to predict the meta-heuristic that is likely to improve the initial solution of a given instance to a greater extent.

As explained in Chapter 1 (Section 1.2.2), the XHSTT data format selected to represent the CB-CTT instance dataset for this thesis was proposed in 2011, as an international standard to model timetabling instances from different countries. However, at the moment, not many solvers are able to support this data format. Furthermore, in the current state of the art, there is a limited number of meta-heuristics designed for the solution of instances in the XHSTT format, most of them specifically customized to suit the particular constraints included in the instances of the third international timetabling competition (ITC-2011) [22].

Within the set of available meta-heuristics, the ones designed by the Group of Optimization and Algorithms (GOAL team) of the Federal University of Ouro Preto, winner of the ITC-2011, have proved to be both: *i*) suitable to address the 16 types of constraints considered by the XHSTT format, and *ii*) effective to produce nearly-optimal solutions for the benchmarking instance dataset of the ITC-2011. For these reasons, they were selected to build the algorithm portfolio required by the per-instance algorithm selection model. Next, we describe the overall structure of the selected meta-heuristics. First, the *LLHs* they apply to perturb the different elements of a solution, then the exploration strategies they use to guide their searching process in different neighborhoods.

5.2.1 Low-level Heuristics

As described by Brito et al. [70], the meta-heuristics designed by the GOAL team, perturb the solutions of timetabling instances based on six types of *LLHs*, described next:

1. **Event Swap (ES)**: Swaps the assigned time slots of two selected lectures.
2. **Event Move (EM)**: Moves a lecture from its current assigned time slot to a new unassigned time slot.
3. **Event Block Swap (EBS)**: Similarly to the ES-move, the EBS-move swaps the time slots of two selected lectures, while preserving the continuity of adjacent lectures that have different duration.
4. **Resource Swap (RS)**: Swaps the resources of a particular role (e.g., room, teacher) assigned to two selected lectures.
5. **Resource Move (RM)**: Allocates a new resource of a particular role to a lecture instead of the previously allocated one.
6. **Kempe Move (KM)**: Performs a *chain* of moves between all lectures of two selected time slots based on bipartite conflicts graphs in which vertices are lectures, and edges are links between lectures that share a resource. The goal of this move is to find a chain of permutations able to diversify the exploration of the solution space to a greater extent.

To illustrate how the described *LLHs* perturb the solutions consider the example presented in Figure 5.4. In this example, three classes (represented with different colors)

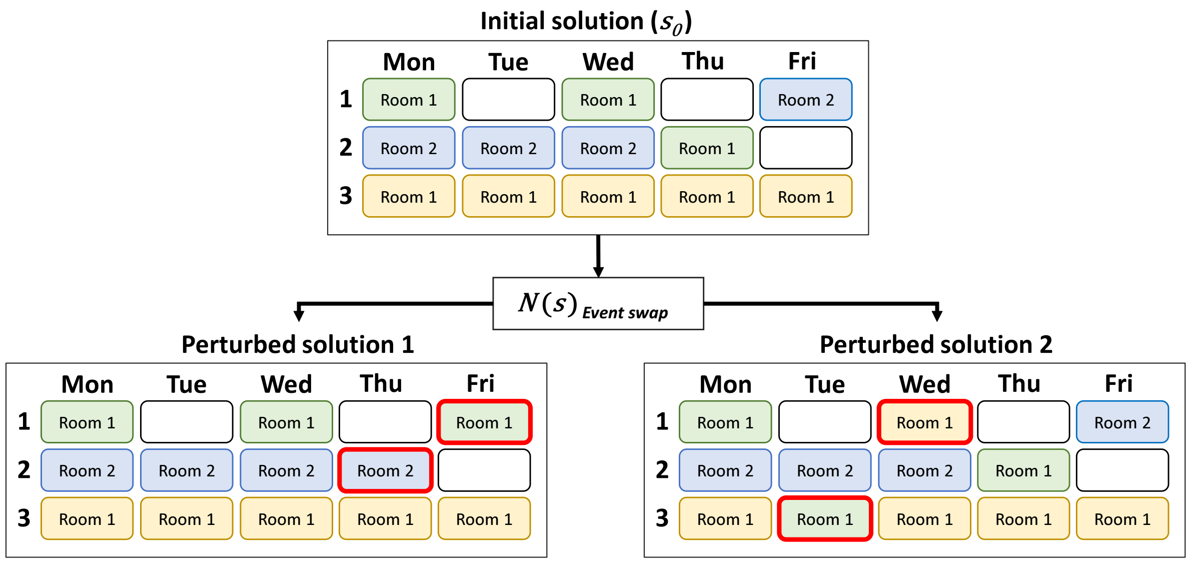


Figure 5.4: Illustration of the effect of the *Event Swap* heuristic in the perturbation of an initial solution (s_0) that defines the schedule of three classes into a five-day timetable.

are initially allocated in two rooms and scheduled in a five-day timetable. If the described *Event Swap* heuristic is applied to the initial array of elements defined in the solution s_0, it can be modified in diverse ways, as illustrated in the two perturbed solutions at the bottom of the figure. As observed, although modified with the same LLH (i.e., within the same neighborhood), both perturbed solutions produce different schedules for the classes. In each of them, different lectures (marked in red) are swapped from their initial schedule. Similarly to the illustrated *Event Swap* heuristic, all these low-level heuristics perturb different types of elements (i.e., events, time slots and resources) of CB-CTT instance solutions in different ways, resulting in the exploration of different neighborhoods of the solution space.

5.2.2 Exploration Strategies

Although based in the same set of LLHs, each meta-heuristic selected for the algorithm portfolio employs a different exploration strategy to find the optimal solution for CB-CTT instances. Hence, it differs in the frequency, the sequence, and the conditions defined to explore different solution neighborhoods. Next, we describe the exploration strategies, used by the four meta-heuristics included in the algorithm portfolio defined for this thesis:

- **ILS** [91]: A meta-heuristic based on Iterated Local Search that through the unconditional acceptance of a distant neighbor produce a new candidate solution that is improved by a descent method. The descent method runs until a number of iterations without improvement is reached, producing a solution s' that it is accepted if is better than the best found solution s^*.

- **LAHC** [92]: A meta-heuristic based on the Late Acceptance Hill Climbing that stores on a vector $\mathbf{p} = \mathbf{p_0}, \ldots\ \mathbf{p_{l-1}}$ the penalty values of a set of solutions for a given instance. At each iteration i, a candidate solution s' is evaluated and accepted if its penalty value is less than or equal to the penalty value stored on the i mod l position of p. Besides, if the accepted solution is better than the best solution s^* found so far, its penalty value is stored on the position $i\ mod\ l$ of vector $\mathbf{p}$.

- **SA** [91]: A meta-heuristic based on the Simulated Annealing that simulates the tempering process of a material. This meta-heuristic iteratively selects a random *LLH* to perturb the current solution of an XHSTT instance. The effect of each *LLH* on the penalty function is calculated as $\Delta = f(s') - f(s)$. Successful *LLHs* (those with $\Delta \leq 0$) are unconditionally accepted, but unsuccessful moves are accepted with some probability $e^{\Delta/T}$, where T is the temperature parameter that defines the probability of accepting worse solutions.

- **VNS** [71]: A meta-heuristic based on the strategy of Variable Neighborhood Search that improves the solution of XHSTT instances by performing a systematic change of neighborhoods. At each iteration, a neighborhood $N_k(s)$ is selected, then a descent method that searches for a local optima in the selected neighborhood is applied. If the solution found by the descent method is better than the current solution, the solution is updated and the neighborhood is set to the first type of *LLH*. Otherwise, the next *LLH* is set as the new neighborhood to continue the search process.

5.3 Performance of the Meta-heuristics

After defining the meta-heuristics to be included in the algorithm portfolio, the next step is evaluating their solving performance across the problem space defined by the generated CB-CTT instance dataset. As explained in Chapter 3 (Section 3.3.5), the quality of the solution of an instance is evaluated according to its *penalty value*, a cost function that assigns a value of 1,000 to each hard (mandatory) constraint violation, and a value of 1 to each soft (optional) constraint violation. The penalty value is a measure that indicates how close the solution of an instance is from its optimal solution. Therefore, we defined as the measure to evaluate the performance of the meta-heuristics.

In Chapter 3, following the solution process defined for the instances (see Figure 5.3), the KHE timetabling engine was applied to solve the generated CB-CTT dataset. According to the quality of the obtained solutions, instances were categorized as *Very Easy*, *Easy*, *Medium*, *Hard*, and *Very Hard*. The KHE timetabling engine was able to find 321 optimal solutions for *Very Easy* instances. Thus a total of 5,679 non-optimally solved instances (i.e., instances with a penalty value higher than zero) remained to be improved by the portfolio of four meta-heuristics.

To evaluate the performance of the portfolio, we executed each meta-heuristic to improve the solutions of the 5,679 non-optimally solved instances. From the obtained results, two analyses were performed regards to *i)* the global effectiveness of the algorithm portfolio and *ii)* the performance variation between the meta-heuristics across the instance dataset.

5.3.1 Global Effectiveness of the Algorithm Portfolio

The first decision to be addressed was the amount of time the meta-heuristics would be allowed to execute. It would be expected that longer running times would improve the quality of the initial solutions to a greater extent. To confirm this assumption, and define a common per-instance running time, a random sample of 600 CB-CTT instances (120 of each type) was selected.

The selected sample was solved by the four meta-heuristics using four different running times (in seconds): 100, 500, 1000, and 3,600. It was observed that a running time of 3,600 seconds produced solutions with lower penalty values than those obtained with running times of 100 and 500 seconds, however not significantly lower (on average less than 0.6%) than those obtained with a running time of 1,000 seconds. From these results, a maximum running time of 1,000 seconds per instance was set for each meta-heuristic. Additionally, for comparison purposes, all meta-heuristics started their solution process with the same initial seed.

As the first performance analysis, the effectiveness of the portfolio (as a unit) to improve the quality of the initial solutions was evaluated. The portfolio improved the initial solutions of 5,491 instances (i.e., 96.68% of the defined problem space) and found 1,098 new optimal solutions (i.e., with a resulting penalty value of zero), leading to a total reduction of 29.35% in the sum of the penalty values of the initial solutions. Because of these results, the meta-heuristics included in the portfolio were considered competitive enough to be applied to the defined CB-CTT problem space.

5.3.2 Performance Variation Between the Meta-heuristics

As the second performance analysis, the performance variation within the portfolio of meta-heuristics was evaluated. The performance variation within a portfolio is commonly overlooked in the construction process of per-instance algorithm selection models. However, it provides information about two relevant aspects, the *usefulness* and the *feasibility* of such models. *Usefulness* answers to the question: ‘*“is it worth it to create a selection model for this algorithm portfolio?”* while *feasibility* to the question: *“is the difference in the performance of the algorithms large enough to create accurate selection models?”*.

To evaluate the usefulness aspect of the portfolio, the penalty values of the solutions produced by the meta-heuristics were compared to define the *best-known solution* of each instance (i.e., the solution with the lowest penalty value). From these comparisons, the performance variation across the instance space was measured according to the following

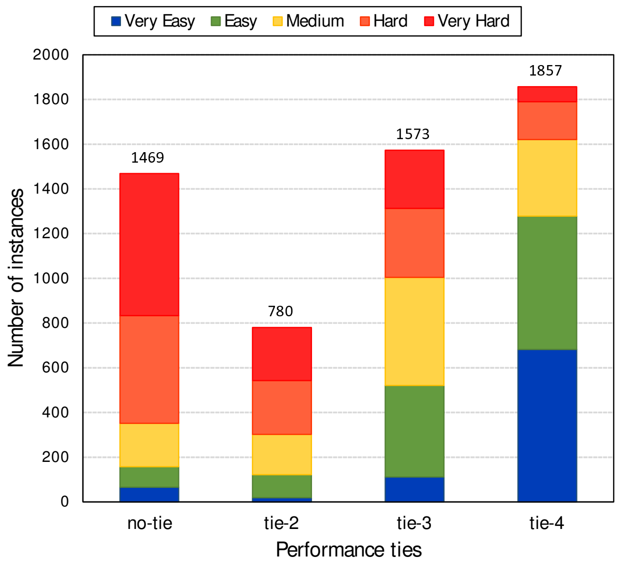


Figure 5.5: Distribution of *performance ties* according to the defined five types of instances, from *Very easy* to *Very hard*.

four types of *performance ties*: instances solved to its best-known solution by only one meta-heuristic (*no-tie*), and instances equally solved to its best-known solution by two (*tie-2*), three (*tie-3*), or four (*tie-4*) meta-heuristics.

Figure 5.5 shows the distribution of *performance ties* according to the five instance categories defined to measure the initial solving difficulty of the instances, from *Very Easy* to *Very Hard*. As observed, there are 1,857 *tie-4* instances (mainly of types *Very Easy* and *Easy*) for which the selection of a meta-heuristic is trivial, since any meta-heuristic would find a solution of the same quality. Nonetheless, there are 3,822 instances for which the implementation of a meta-heuristic selection tool would be useful. The *non-trivial algorithm selection space* corresponds to 67.30% of the instances improved with the portfolio of meta-heuristics, a proportion that makes evident the usefulness of a per-instance algorithm selection model for this portfolio, mainly for the instance categories *Medium*, *Hard* and *Very Hard*.

To evaluate the feasibility aspect of the portfolio, the performance variation within the portfolio of meta-heuristics was calculated in terms of the standard deviation. The standard deviation is a statistical measure of dispersion. Thus, when calculated to compare for the performance of the meta-heuristics, it indicates the similarity between the meta-heuristics included in the portfolio.

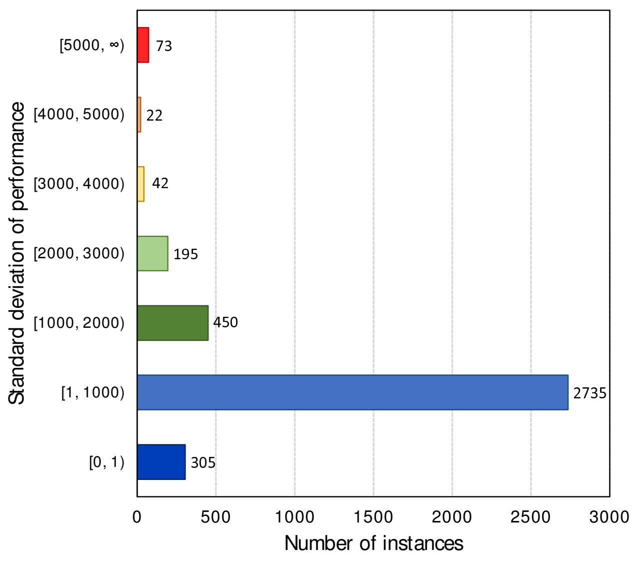


Figure 5.6: Distribution of the standard deviation of the performance of the four meta-heuristics across the non-trivial selection space.

Figure 5.6 shows the distribution of the standard deviation across the *non-trivial algorithm selection space*. As observed, for most instances, the standard deviation of the performance of the meta-heuristics is less than 1,000, a relatively small deviation concerning the penalty values assigned to constraint violations (1 for soft constraints and 1,000 for hard constraints).

The performance variation between the algorithms in a portfolio has a significant effect on the accuracy of algorithm selection models. When the performance variation is high, the predictions of simple models can lead to good selections even if they are not highly accurate. However, when the performance variation is low, even highly accurate prediction models might struggle to make correct algorithm selections. For example, if the penalty values (PV) of the solutions obtained by the portfolio of four meta-heuristics for two given instances are as follows, $instance_1$: $\{PV_{ILS} = 5$, $PV_{LAHC} = 8$, $PV_{SA} = 4$, $PV_{VNS} = 3\}$ and $instance_2$: $\{PV_{ILS} = 20$, $PV_{LAHC} = 1,150$, $PV_{SA} = 600$, $PV_{VNS} = 835\}$. Then, it is evident that because of the higher performance variation, the second instance will require less accurate statistical models to predict the algorithm with the lowest penalty value.

The relatively low standard deviation, of less than 1,000 in 79.5% of the non-trivial selection space, indicates the difficulty of constructing an accurate per-instance algo-

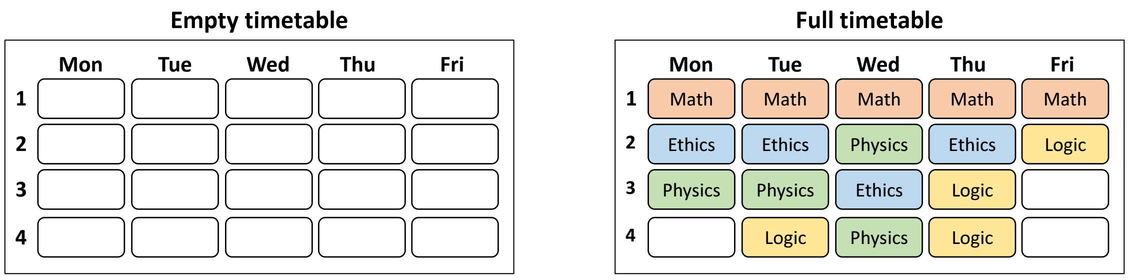


Figure 5.7: Illustration of an empty and a full timetable.

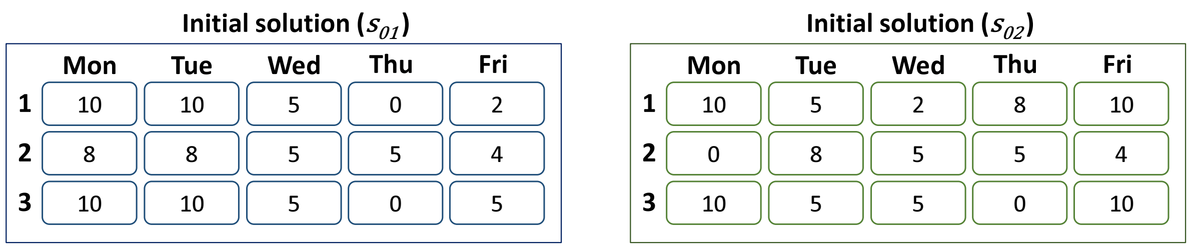
rithm selection model for the portfolio of four meta-heuristics using simple statistical models. Thus, in the next chapter, we propose a hybrid statistical approach that can be applied to useful but feasibly challenging algorithm portfolios.

5.4 Characterization of Initial Solutions

In Chapter 4, we formulated and evaluated a collection of metrics to characterize the solving process required to generate initial solutions for CB-CTT instances. This collection of metrics describe the combinatorial spaces of *empty* timetables —before the assignment of times and resources. The meta-heuristics included in the portfolio, however, start the solution process from non-optimal but already *full* timetables. Thus, require an additional set of metrics to characterize the initial arrangement of elements (i.e., times and resources) that defines the timetables to be improved.

Figure 5.7 presents an example of both types of timetables. On the one hand, the empty timetable at the left illustrates the problem solved with the KHE timetabling engine, generate an initial solution starting from a timetabling with no assignments. On the other hand, the full timetable at the right illustrates the problem solved with the selected meta-heuristics, improve the array of elements of a timetable.

Based on the LLHs employed by the meta-heuristics, next we described an additional collection of three types of metrics formulated to characterize the main properties of full timetables. These are: *i*) event dispersion, *ii*) resource similarity, and *iii*) resource usage.



Initial solution (s_{01})

	Mon	Tue	Wed	Thu	Fri
1	10	10	5	0	2
2	8	8	5	5	4
3	10	10	5	0	5

Initial solution (s_{02})

	Mon	Tue	Wed	Thu	Fri
1	10	5	2	8	10
2	0	8	5	5	4
3	10	5	5	0	10

Figure 5.8: Distribution of the events scheduled on two initial solutions. The numbers indicate the events scheduled on each time slot. For example, in the solution s_{01}, four events are scheduled on the time slot *Fri-2*.

5.4.1 Event Dispersion

Event dispersion is a type of metric that defines how evenly a set of scheduled events are distributed in a time grid of n_t times by n_d days. Therefore, it is defined as the combined standard deviation of the number of scheduled events per time (x_t), and the number of scheduled events per day (x_d), as shown in Equation 5.1.

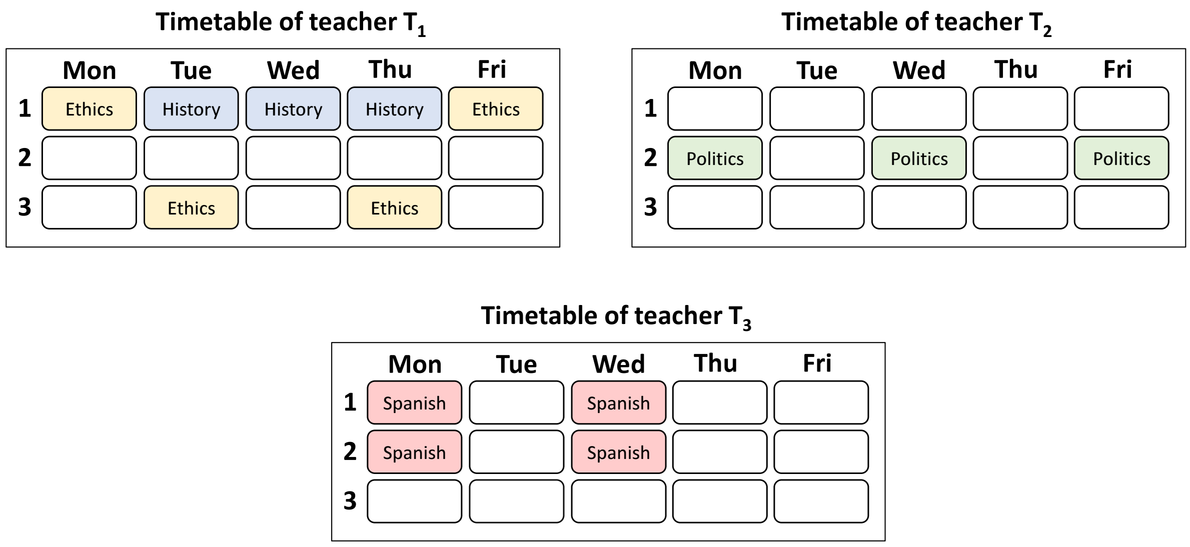
$$dispersion = \sqrt{\frac{n_t s_t + n_d s_d}{n_t + n_d}} \tag{5.1}$$

where, s_t is the standard deviation of the average number of scheduled events per time, and s_d is the standard deviation of the average number of scheduled events per day.

Consider the two time grids presented in Figure 5.8 that shows the number of events (a total of 87) scheduled on each time slot of two different initial solutions (s_{01} and s_{02}). If Equation 5.1 is applied to both time grids, the value of *dispersion* for solution s_{01} is 2.43, and 1.34 for solution s_{02}. These results indicate a more balanced distribution of events in solution s_{02}, because the higher the dispersion on a time grid is, the more unevenly the distribution of the scheduled events becomes.

5.4.2 Resource Similarity

Resource similarity is a type of metric that indicates how similar the timetables of a set of resources are (e.g., teachers and rooms). Therefore, it is defined as the average number of time slots on which a given set of resources are simultaneously scheduled.


Timetable of teacher T_1

	Mon	Tue	Wed	Thu	Fri
1	Ethics	History	History	History	Ethics
2					
3		Ethics		Ethics	

Timetable of teacher T_2

	Mon	Tue	Wed	Thu	Fri
1					
2	Politics		Politics		Politics
3					

Timetable of teacher T_3

	Mon	Tue	Wed	Thu	Fri
1	Spanish		Spanish		
2	Spanish		Spanish		
3					

Figure 5.9: Illustration of the timetables defined for three teachers from the field of social sciences.

Consider the three timetables shown in Figure 5.9, that defines the events allocated to three different teachers from the field of social sciences. To compare how similar the distribution of events is for this set of teachers, we need to count the number of time slots simultaneously allocated for each pair of teachers, as shown: $\{T_1 - T_2 : 0\}$, $\{T_1 - T_3 : 2\}$, and $\{T_2 - T_3 : 2\}$. From these counting, we calculate the *similarity* of a social sciences teacher dividing the total number of simultaneously-allocated time slots by the number of teachers; that is $(4/3) = 1.33$.

Resource similarity was calculated for the three types of resources considered in the CB-CTT instances (i.e., rooms, teachers, and curricula, and considered an important type of metric that indicates how easily the events assigned to resources of the same type can be exchanged.

5.4.3 Resource Usage

Resource usage is defined as the percentage of the total capacity of a resource already allocated in the initial solution. Therefore, it is a type of metric that describes how busy a resource is.

To illustrate the calculation of the resource usage, consider the same set of teachers from the field of social sciences, shown in Figure 5.9. As the maximum workload is 15 time slots, the *usage* for each teacher is calculated as follows: $T_1 = 7/15$, $T_1 = 3/15$, and $T_1 = 4/15$. Furthermore, as all these teachers belong to the same study field, it is possible to calculate also the he average usage of a social science teacher as $(\frac{7}{15} + \frac{3}{15} + \frac{4}{15})/3 = 0.311$. This result imply that, in this initial solution, on average, a social science teacher is busy 31.11% of its available time. As with the *similarity*, the *usage* was calculated for all the types of resources included in the instances.

Based on these three types of features —event dispersion, resource similarity and resource usage— 30 new metrics were added to the set of 38 relevant metrics selected to characterize the initial solution of CB-CTT instances (see Table 4.8). As a result, the final collection of metrics defined to characterize the instances consists of 68 metrics. Together, this collection of metrics and the performance of the meta-heuristics constitute the meta-data required for the construction of the per-instance algorithm selection model described in the next chapter.

5.5 Summary

In this chapter, we described the solution strategies of the four meta-heuristics selected to build the algorithm portfolio. The meta-heuristics were applied to improve the initial solutions of the collection of 5,679 non-optimally solved instances. As a result, we evaluated both the global effectiveness of the portfolio and the performance variation between the meta-heuristics. The algorithm portfolio obtained 1,098 new optimal solutions distributed as follows: 475 *Very Easy* instances, 410 *Easy* instances, 202 *Medium* instances, and 11 *Hard* instances. It also improved the overall quality of the initial solutions in 29.35%. Thus, it proved to be useful for the construction of the per-instance algorithm selection model.

To estimate the difficulty of algorithm selection for this portfolio, an additional analysis, measuring the performance variation between the meta-heuristics, was performed. In this analysis, it was observed that the standard deviation between the performance of the meta-heuristics is relatively small (of less than 1,000 in most instances) when compared to the costs assigned to constraint violations. Thus, it suggests that the construction of an accurate per-instance algorithm selection model will require the implementation of complex machine-learning methods.

In the final section of this chapter, we described three types of CB-CTT metrics, formulated to describe the initial solutions improved by the meta-heuristics. These metrics are relevant because, unlike the first collection of metrics (explained in Chapter 4), they describe relevant aspects of already-full timetables that are essential for the construction of an accurate selection mapping of features to algorithms $S(f(x), \mathcal{A})$.

Chapter 6

Algorithm Selection Model

In the last three chapters (Chapter 3: Instance Generator, Chapter 4: Feature Selection, and Chapter 5: Portfolio of Meta-heuristics), we have described the performed processes to obtain the required *meta-data* for our per-instance algorithm selection model. The first half of the meta-data corresponds to the collection of *CB-CTT metrics* (i.e., meta-features) formulated to characterize timetabling instances modeled according to the XHSTT data format. The second half corresponds to the *performance measures* of the four meta-heuristics applied to improve the solution of 5,679 non-optimally solved instances.

According to the framework defined for this thesis (see Figure 6.1), the last step for the construction of the per-instance algorithm selection consists on employing the obtained meta-data to create a selection mapping $S(f(x), \mathcal{A})$, of instance meta-features to algorithms' performance. Such selection mapping is created through a meta-learning process that applies machine learning methods to produce a computational model able to predict the best-performing algorithm for each instance to be solved.

In this chapter, we describe the computational approach employed for the construction of our per-instance algorithm selection model. First, we describe four machine-learning strategies already applied in the literature to perform algorithm selections tasks and compare their flexibility in the context of our problem. Second, we present the general structure of the per-instance algorithm selection model, which employs a two-stage

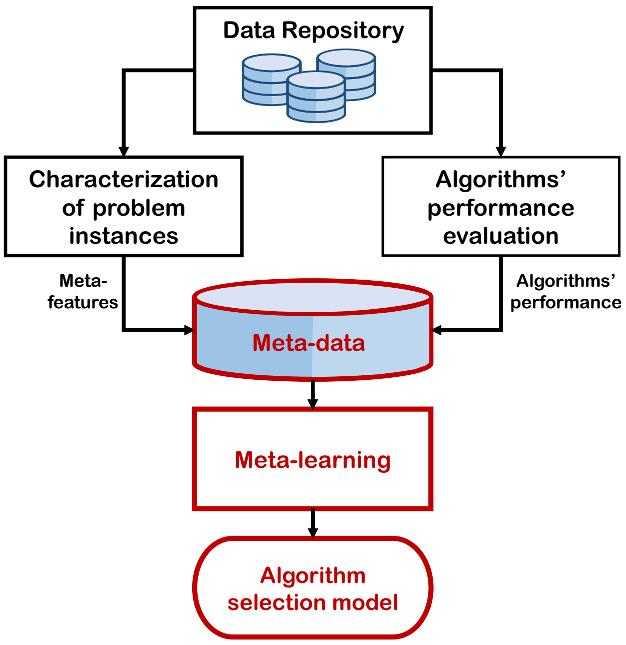

Data Repository
Characterization of problem instances
Algorithms' performance evaluation
Meta-features
Algorithms' performance
Meta-data
Meta-learning
Algorithm selection model

Figure 6.1: Meta-learning framework for the construction of the CB-CTT per-instance algorithm selection model (adapted from [2]). For reference purposes, the elements *Meta-data* and *Meta-learning*, employed to create the *Algorithm election model*, are highlighted in red.

hybrid setting that combines classification and regression methods. Finally, we assess the global effectiveness of the model and analyze its performance.

6.1 Algorithm Selectors

The goal of any per-instance algorithm selection model is to apply the best candidate algorithm to solve a given instance based on the performance variation of the algorithms in a portfolio. The computational method used to determine this variation has a major role in the effectiveness of the model, and in this thesis is referred as *selector*. This section describes four types of *selectors* and analyzes their suitability for the construction of a per-instance algorithm selection model in the Curriculum-Based Course Timetabling (CB-CTT) domain.

6.1.1 Multi-class Classifiers

A multi-class classifier is a mathematical function built from a collection of training instances where each instance belongs to one of a set of possible classes. The goal of this type of classifiers is, given a new instance, to predict the class to which this new instance belongs. For example, if a given instance i is better solved by the third of a portfolio of four algorithms denoted as $\{A^1, A^2, A^3, A^4\}$, then the expected output of the classifier should be $i : \{A^3\}$.

6.1.2 Multi-label Classifiers

Unlike multi-class classifiers, the goal of multi-label classifiers is to predict a set of labels for a single instance. Multi-label classifiers are more flexible and sophisticated since they assume that one instance can be equally associated with more than one label. Therefore, they can be applied to predict more than one algorithm for an instance. For example, if a given instance i can be equally solved by the first and the third algorithms of the same four-algorithm portfolio, $\{A^1, A^2, A^3, A^4\}$, then both related labels must be predicted for that instance as $i : \{A^1, A^3\}$.

6.1.3 Binary Classifiers

The goal of binary classifiers is to predict one of two possible classes to which an instance belongs. Thus, unlike multi-label and multi-class classification approaches, it cannot directly address the selection task for a portfolio of more than two algorithms. A common alternative, however, is defining a combination of binary classifiers —one classifier per class. By following this approach, the selection of an algorithm for a given instance (S_i^α) is expressed by a set of binary values: 0, do not select the algorithm α to solve instance i; or 1, select the algorithm α to solve instance i. For example, the set of predictions $\{S_i^1 = 0$, $S_i^2 = 0$, $S_i^3 = 1$, $S_i^4 = 0\}$ would select the third algorithm to solve instance i; while the set $\{S_i^1 = 1$, $S_i^2 = 1$, $S_i^3 = 0$, $S_i^4 = 0\}$, the first and second algorithm.

6.1.4 Regression Models

An alternative approach to classification-based selectors is regression, used to predict values for continuous variables. Because of its nature, regression methods are not suitable for performing classifications tasks. However, they can be applied to predict the performance of the algorithms.

In this type of *selectors*, a collection of regression models is trained. Each regression model is used to make a prediction Y_i^α, that is, the estimated performance of the algorithm α when applied to solve a given instance i. Then all predictions are evaluated according to a *comparison threshold* (ct) and transformed into binary values to perform the selection (S_i^α).

For example, if the predicted penalty values (used as the measure of performance) of four algorithms for a given instance i are $\{Y_i^1 = 35,\ Y_i^2 = 25,\ Y_i^3 = 24,\ Y_i^4 = 42\}$, then by comparing them without any threshold ($ct = 0$), the *selector* would choose the algorithm with the lowest predicted penalty value, $\alpha = 3$, to be applied to that instance, returning the following output: $[S_i^1 = 0\ ,\ S_i^2 = 0,\ S_i^3 = 1,\ S_i^4 = 0]$. However, if a comparison threshold of $ct = 5$ is defined, the algorithm $\alpha = 2$ would also be chosen by the *selector*, as it is close (within the range defined by the ct) to the lowest predicted performance for that instance; thus producing the following output: $[S_i^1 = 0\ ,\ S_i^2 = 1,\ S_i^3 = 1,\ S_i^4 = 0]$.

6.2 Discussion of Algorithm Selectors

The four *selectors* described in the previous section differ in their *easiness* of implementation and *flexibility* to address the algorithm selection task. These two aspects were analyzed to define the most suitable selector to be used for the construction of our per-instance algorithm selection model.

Figure 6.2 presents the structure of the *selectors* considered for the portfolio of meta-heuristics presented in Section 5.1; these are, ILS, LAHC, SA, VNS. As observed, multi-class and multi-label selectors are easy to implement since they require training a single classifier to predict the best meta-heuristic(s) for a given instance. On the other hand, binary classifiers require training a different classifier for each meta-heuristic in the *portfolio*, to produce a set of selections (S_i^α) that are combined into a single output.

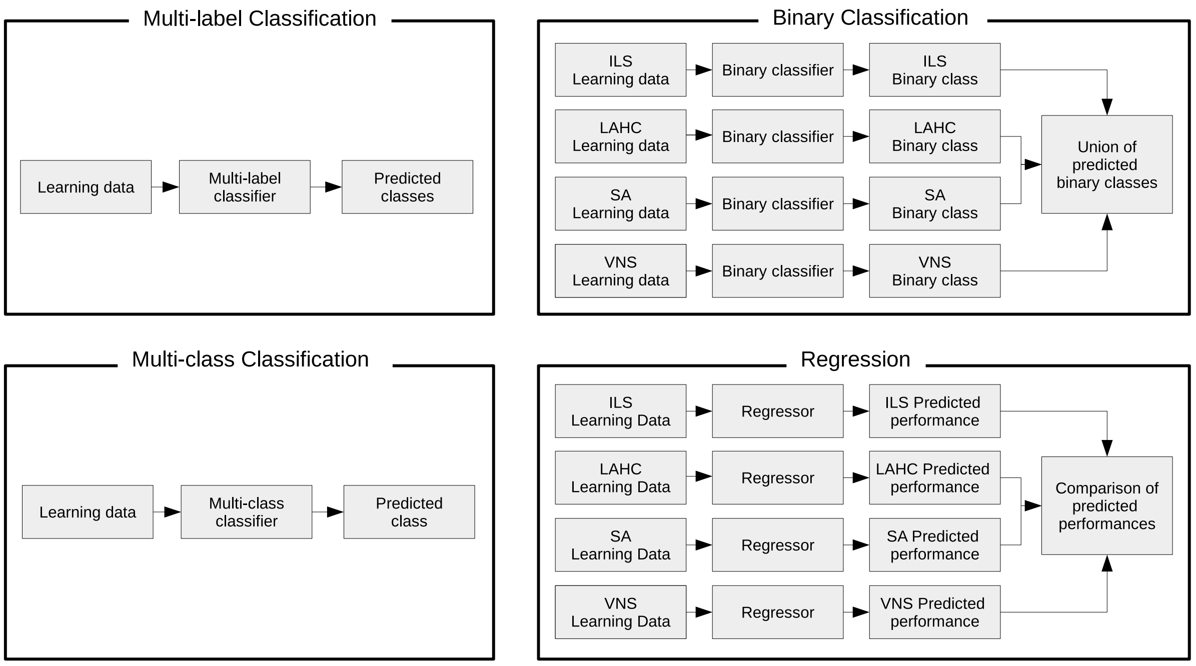


Figure 6.2: Set of algorithm *selectors* considered to be used for a portfolio of four algorithms.

The more complex type of *selector* is regression, which requires different models to predict the performance (Y_i^α) of the meta-heuristics (α) on a given instance (i), and a comparison threshold (ct) to transform the predictions into binary selections (S_i^α).

Despite its simplicity, multi-label classifiers use strict multiple assignments of labels, that are not flexible to handle *performance ties*. As explained in Section 5.3.2, a performance tie occurs when more than one meta-heuristic is able to find the best-known solution for a given instance. For example, if the meta-heuristics {ILS, VNS} are the best to solve a given instance i, and the predicted label obtained by a multi-label classifier is {VNS}, this prediction would be considered a misclassification error, even though it is partially correct and would finally lead to the best-know solution of that instance. Multi-class classifiers are not flexible either to handle *performance ties* because they only predict a single label per instance. Therefore, considering the same example, this type of selector would consider only one of these possibilities i : {ILS} or i : {VNS} as correct for the solution of instance i.

Due to their structure, which requires using an independent learner for each meta-heuristic in the portfolio, binary classifiers and regression models are more flexible to address *performance ties*. As each learner is trained on different data, all the possible combinations of selections (S_i^{α}) can be predicted and evaluated. Considering the same example, if the meta-heuristics {ILS, VNS} are the best to solve a given instance i, and the selection obtained with binary classifiers or regression models is $\{S_i^{ILS} = 1, S_i^{LAHC} = 0, S_i^{SA} = 0, S_i^{VNS} = 0\}$, then this prediction would be considered correct, as it would apply one of the possible algorithms that produce the best solution for such an instance.

Although more complex than binary classifiers, regression-based *selectors* provide a better understanding of the relationship between instance features and the performance of each meta-heuristic included in the portfolio. Besides, they can be adjusted to a higher degree with the definition of a *comparison threshold* to perform the selections. Because of these additional advantages, we chose the regression-based *selectors* to construct the per-instance algorithm selection model.

6.3 Setting of the Algorithm Selection Model

As described in Chapter 2, defining the type of setting is a crucial decision to design per-instance algorithm selection models. If the process of *selecting* an algorithm for solving a problem instance is computationally more expensive than solving the instance, then there is no point in doing so. Thus, a proper balance between accuracy and complexity is a decisive requirement of any selection model.

In Section 2.2, we described three common settings applied to construct algorithm selection models. *Per-portfolio* settings build models of entire portfolios (as a group); *per-algorithm* settings build individual models for the constituent algorithms of a portfolio; *hybrid* settings combine different machine-learning strategies to get accurate predictions. Hybrid settings are more computationally expensive; however, as they integrate different methods in a hierarchical order, they are able to obtain better predictions in low-variation portfolios.

As explained in Section 5.3.2, the performance variation within our portfolio of four meta-heuristics was statistically evaluated in terms of the standard deviation of performance —the lower the standard deviation, the more similar the performance of the meta-heuristics. In most of the instances, the observed standard deviation (see Figure 5.6) was less than 1,000, a relatively low-performance variation when compared to the penalty values assigned to constraint violations (a value of 1 for soft constraints and a value of 1,000 for hard constraints). Hence, to build a per-instance algorithm selection model suitable for this low-performance variation, we adopted a hybrid approach to improve the precision of the defined regression-based selector.

For supervised learning tasks, hybrid approaches are commonly structured using ensemble methods such as bagging, stacking, and voting, which combine the strengths of different machine learning algorithms, running in sequence or parallel, to improve the accuracy of the predictions of algorithms. However, for this ensemble methods to be useful, they must be accurate and efficient enough to not add more significant computational effort to the selection process of an instance.

For this thesis, we tested different configurations of ensemble methods combining the predictions of regression models trained using three machine learning algorithms: decision trees, multilayer perceptrons, and support vector machines. The tested configurations include the following:

- **Averaging**: Making multiple predictions using different machine learning models and taking the average of all predictions as the final prediction.
- **Weighted averaging**: Averaging the predictions of different machine learning models but assigning higher importance (weight) to the predictions of models with higher accuracy.
- **Stacking**: Making multiple predictions using different machine learning models (called base models) and training a meta-model to find the best combination of such predictions.
- **Bagging**: Combining the predictions of machine learning models trained on different subsets of data.

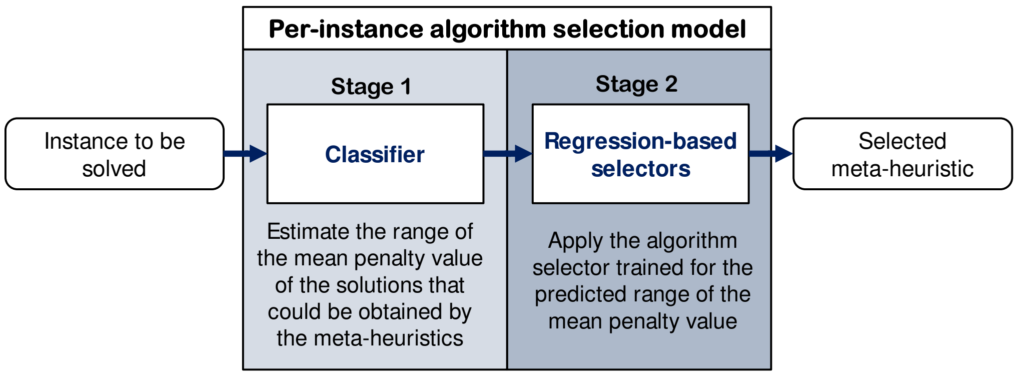

Per-instance algorithm selection model
Stage 1
Stage 2
Instance to be solved
Classifier
Regression-based selectors
Selected meta-heuristic
Estimate the range of the mean penalty value of the solutions that could be obtained by the meta-heuristics
Apply the algorithm selector trained for the predicted range of the mean penalty value

Figure 6.3: Structure of the algorithm selection model based on the FCTR forecasting technique.

- **Boosting**: Combining machine-learning models sequentially to correct the prediction errors of prior models in the sequence.

Although more structure-sophisticated, these ensemble methods proved to obtained less accurate predictions in terms of the explained variance score (described in Section 4.2.2) than the selection strategy defined for our model. In our experimental tests, an alternative approach proved to be more accurate and computationally simple, the FCTR (First Classification Then Regression) forecasting technique [93]. Therefore, it was chosen to structure the hybrid approach for our per-instance algorithm selection model. In general terms, this forecasting technique makes predictions following a two-stage sequence. In the first stage, a classifier estimates the range of possible values for a prediction. In the second stage, a regression model trained within the predicted range of values makes the final prediction.

According to the FCTR approach, the model shown in Figure 6.3 was defined to perform the per-instance algorithm selection for the CB-CTT instances. As shown, in the first stage, a classifier estimates the range (represented by a class) of the mean penalty value of the solutions that the meta-heuristics could find for a given CB-CTT instance. Then, in the second stage, a regression-based algorithm selector (as the one illustrated at the bottom right of Figure 6.2), trained in the estimated range of values, is applied to choose the meta-heuristic with the best expected performance. A detailed description of the classifier and regression-based selectors constructed for the per-instance algorithm selection model is presented next.

Table 6.1: Seven classes defined to predict the range of the mean penalty value of the solutions.

Class	Range of the mean penalty value	Number of instances
1	[0, 2,000)	2,340
2	[2,000, 19,000)	686
3	[19,000, 40,000)	445
4	[40,000, 60,000)	662
5	[60,000, 80,000)	521
6	[80,000, 100,000)	527
7	[100,000, ∞)	498

6.3.1 Stage 1 - Classifier

Because of the relative similar performance of the four meta-heuristics, a Gradient Boosting Tree (GBTree) classifier was trained to predict the range of the mean penalty value of the solutions improved by the meta-heuristics —each range represented as a class. To obtain a balanced distribution of classes for the classifier, we defined the seven ranges shown in Table 6.1 based on a statistical analysis of the performance data. However, as the performance data was not uniformly distributed (but biased towards low penalty values), the number of instances per class differed, as shown in the third column of Table 6.1. This distribution of classes was defined as the output to be predicted by the GBtree at the first stage of our per-instance algorithm selection model.

6.3.2 Stage 2 - Regression-Based Selectors

For the second stage of the model, seven regression-based selectors were trained, one for each class, according to the structure presented at the bottom right of Figure 6.2. In experimental tests of regression methods and parameters, linear regression proved to be as accurate as more complex methods (e.g., multi-layer perceptrons, support vector machines) to predict the low-performance variation between the meta-heuristics; therefore, it was chosen as the learning strategy to trained the selectors in the second stage of the per-instance algorithm selection model. Besides, to ensure that the selectors chose only one meta-heuristic, we set a comparison threshold of 0 ($ct = 0$).

6.4 Experimental Results

Once the general structure of our per-instance algorithm selection model has been described, we now present the experimental results obtained with the implementation of our model. Our experiments were conducted using the implementations and default parameters of scikit-learn[1], a machine learning library for the Python programming language [81]. The GBTree classifier was built using 100 decision trees as weak learners, and all regression-based selectors were fit using the ordinary least squares method.

The primary goal of per-instance algorithm selection is to predict the best algorithm for each instance to be solved. Researchers evaluate the performance of their models according to this goal, using measures that compare the predictions of their models to those of perfect selectors. In this section, two popular measures employed to evaluate algorithm selection models are discussed, then the results of the evaluation of our model are presented.

6.4.1 Discussion of Performance Measures

In the current literature, research works commonly report the performance of per-instance algorithm selection models using two measures: *accuracy* [65] and *closed SBS-VBS gap* [94].

Accuracy: It is a straightforward measure to assess algorithm selection models using a classification approach, where each label represents one algorithm in the portfolio. According to this representation, the accuracy calculates the performance as the proportion of labels correctly predicted for all instances.

Closed SBS-VBS gap: It is a measure that takes into account the performance of two baselines: the *single best solver* (SBS), that represents the algorithm with the best performance across a defined instance space; and the *virtual best solver* (VBS), that represents a perfect algorithm selector that chooses the best algorithm for each

[1]The description of the regression models and their default parameters can be found in the official documentation of scikit-learn at `https://scikit-learn.org/stable/supervised_learning.html`

instance. From these two baselines, this measure determines first the *SBS-VBS gap*, that is, the potential gain in the quality of the solutions that can be obtained by a perfect selector over the single-best solution strategy. It then calculates the *closed SBS-VBS gap* of a model as the fraction of the gap it closes.

Although designed with the same purpose, these measures evaluate algorithm selection models regarding two aspects: the precision of the selections (estimated through the *accuracy*), and the gain in performance (estimated through the *closed SBS-VBS gap*).

In recent years, two relevant competitions —the ICON Challenge on Algorithm Selection [95] and the Open Algorithm Selection Challenge [96]— have been organized to compare the strengths of algorithm selection approaches across different domains. Both competitions established the *closed SBS-VBS gap* as the performance measure (m) to evaluate any submitted selection model (s). This measure proved to be useful as it provided lower (m_{VBS}) and upper (m_{SBS}) bounds for the performance of the competitors, and assigned an implicit cost (or weight) to the selection errors (i.e., the potential gain of selecting the right algorithm to solve an instance).

Although practical models could make incorrect selections, the selection of such suboptimal algorithms might still imply a performance improvement respect to the single-best solver strategy —an improvement not considered by the accuracy. Therefore, to consider this fact in the assessment of our model, we evaluated its performance using the *closed SBS-VBS gap* ($\hat{m}_s$), as defined for the Open Algorithm Selection Challenge [97]:

$$\hat{m}_s = \frac{m_{SBS} - m_s}{m_{SBS} - m_{VBS}} \tag{6.1}$$

In this measure, a value of 1.0 corresponds to a perfect per-instance algorithm selection model that always chooses the best algorithm; 0.0 corresponds to a model that performs as the *single best solver*; and a negative value, corresponds to a model that performs worse than the *single best solver*.

Consider the following hypothetical example in the context of our CB-CTT instance space. If the sum of the penalty values obtained by the best-performing meta-heuristic

(m_{SBS}) across a given instance space is 4,500, and the sum of the penalty values obtained by a perfect algorithm selector (m_{VBS}) is 1800, then the *SBS-VBS gap*, which measures the potential gain to be obtained by a selection model, is 2,700. According to these values, if the sum of the penalty values obtained by a selection model s is 2,880, then its performance, closed SBS-VBS gap, is calculated as: $\hat{m}_s = (4,500 - 2,880)/(4,500 - 1,800) = 0.6$. A value that would indicate that the evaluated selection model s achieved a performance of 60% of that of a perfect algorithm selector.

6.4.2 Performance of the Algorithm Selection Model

The experimental results obtained from the implementation of the proposed FCTR algorithm selection model are summarized next. All selections were performed following a 10-fold cross-validation strategy. In this strategy, the collected meta-data (i.e., CB-CTT metrics and algorithms' performance) was split into ten random folds of data, from which nine folds were used to train the classification and regression models, and the remaining one to assess the performance of our model. By following this strategy, all instances were included once into the test set, ensuring a fair evaluation.

A major advantage of the defined FCTR approach is that it allows us to evaluate the performance of our per-instance selection model in detail. Each one of the seven classes represents a range of mean penalty values that indicate the solving difficulty of the instances for the portfolio of meta-heuristics. Table 6.2 presents the percentage of performance ties between the meta-heuristics, and the *closed SBS-VBS gap* ($\hat{m}_s$) for each class. The first seven rows correspond to the defined seven instance classes and the performance of their individual regression-based selectors; the row *Total* corresponds to the overall instance dataset and the performance of the general selection model.

Despite the low-performance variation within the portfolio of four meta-heuristics (analyzed in Section 5.3), the constructed per-instance algorithm selection model obtained better solutions for the instances than the single-best solver (SBS) strategy, closing the *SBS-VBS gap* ($\hat{m}_s$) to a different extent in each of the classes. As observed in Table 6.2, the selection performance in all the classes is between 0.275 and 0.435, which

Table 6.2: Distribution of performance ties and closed SBS-VBS gap ($\hat{m}_s$) in the defined instance classes and overall dataset.

Class	**Instances**	**Percentage of performance ties**				$\hat{\mathbf{m}}_\mathbf{s}$
		no-tie	**tie-2**	**tie-3**	**tie-4**	
1	2,340	8.7%	7.9%	27.9%	55.5%	0.435
2	686	14.0%	11.8%	42.0%	32.2%	0.501
3	445	33.9%	18.5%	24.0%	23.6%	0.534
4	662	38.1%	19.0%	25.7%	17.2%	0.404
5	521	49.1%	19.8%	22.5%	8.6%	0.456
6	527	48.2%	19.2%	23.1%	9.5%	0.297
7	498	51.6%	20.3%	23.5%	4.6%	0.275
Total	5,679	25.9%	13.7%	27.7%	32.7%	0.386

though it could be considered low it is similar to that obtained by selection models in other combinatorial problems. For example, in the two international algorithm selection competitions [16], the participant models exhibited a closed SBS-VBS gap between 0.516 and 0.634 in 2015; and between 0.04 and 0.62 in 2017. Both competitions were based on the repository ASlib [98], which includes diverse sets of instances from different problems, such as propositional satisfiability (SAT), quantified boolean formula (QBF), and constraint solving (CSP).

In the presented results, we notice that, except for *Class 6* and *Class 7*, our model exhibits a performance above 0.40 in the overall instance space. Despite being similar to *Class 5* in the number of instances and performance ties, in the last two classes, the per-instance algorithm selection is particularly challenging. To further analyze the causes of this differing performance, Table 6.3 presents three relevant measures related to the performance ($\hat{m}_s$) of our model: *SBS-VBS gap*, *F1 score*, and mean coefficient of variation (MCV).

As explained above, the *SBS-VBS gap* is the difference between the performance of two baselines, the single best solver (SBS) and the virtual best solver (VBS). It indicates the potential gain in performance that can be obtained by a perfect per-instance algorithm selector. The third column of Table 6.3, shows the total SBS-VBS gap and its distribution across the instance classes. Together, the first three classes encompass 61.1% of the instances but account for only 20.2% of the total potential

Table 6.3: SBS-VBS gap, F1 score, and mean coefficient of variation (CV), for the analysis of performance ($\hat{m}_s$) in the defined instance classes and overall dataset.

Class	**Instances**	**SBS-VBS gap**	**F1 score**	**MCV**	$\hat{\mathbf{m}}_\mathbf{s}$
1	2,340	47,852	0.931	46.8%	0.435
2	686	59,566	0.675	13.5%	0.501
3	445	150,699	0.736	1.9%	0.534
4	662	216,299	0.748	1.1%	0.404
5	521	259,100	0.549	1.1%	0.456
6	527	275 208	0.678	0.9%	0.297
7	498	269,564	0.853	0.8%	0.275
Total	5,679	1,278,288	0.798	21.46%	0.386

gain, while the last four account for the remaining 79.8%, due to their higher penalty values. The potential gain of each class can be considered a weight attached to its corresponding algorithm selector —the higher the potential gain, the more the effect of the selector on the global performance. Due to their high potential gains, the potential causes of the lower performance in the last two classes are analyzed next.

The *F1 score* is a statistical measure that calculates the performance of classifiers as the harmonic mean of the precision and recall, where 1.0 is the best value (perfect precision and recall) and 0.0 the worst. To analyze the effect of misclassifications of the GBTree after the first stage of our model, the fourth column of Table 6.3 presents the F1 score of each class (label) and the micro F1 score for the total instance dataset. The results show two aspects: *i*) there is no apparent correspondence between the F1 score and the performance ($\hat{m}_s$), and *ii*) the lowest F1 score (0.549) of the classifier did not produce a significant loss of performance ($\hat{m}_s$) in *Class 5* instances. Together, these aspects indicate that missclassifications reduce the performance of the model to a lower extent than the errors of the regression-based selectors applied in the second stage of our model.

The *coefficient of variation* is a statistical measure, defined as the ratio of the standard deviation to the mean, expressed as a percentage. Since it is a unit-free dispersion measure, independent of data magnitude, it was used to compare the relative variability of the meta-heuristics between different classes. For each instance, the coefficient of variation was calculated based on the solutions obtained by the meta-heuristics as follows: mean of the penalty values divided by the standard deviation of the penalty values. The mean coefficient of variation (MCV) of each class is presented in the fifth column of Table 6.3. Notice that, except for *Class 6* and *Class 7*, all instance classes exhibit a MCV over 1%, suggesting an empirical threshold of relative variability under which the performance of the regression-based selectors, in interplay with GBtree classifier, significantly decreases.

6.5 Summary

In this chapter, we described the per-instance algorithm selection constructed for the solution of CB-CTT instances, as an answer to our research questions, stated in the Introduction of this thesis: *i) How can meta-learning approaches be used to accurately relate the relevant features of CB-CTT instances to the performance of algorithms*?" and "*ii) How can a per-instance algorithm selection model be generated to apply the best algorithm to solve a given CB-CTT instance*?".

The design of the per-instance algorithm selection model designed to answer these questions was proposed as an alternative to popular ensemble methods applied to solve similar combinatorial problems. This design consists of a hybrid approach based on the FCTR technique. It first estimates the average performance that the meta-heuristics in the portfolio could have when solving a given instance (according to seven defined classes). Then, according to this estimation, it predicts and compares the individual performance of the meta-heuristics to select the one with the best expected performance to solve such an instance.

To evaluate the precision of the selections made, rather than reporting the average accuracy, the per-instance algorithm selection model was evaluated using the closed SBS-VBS gap, a performance measure used at international algorithm selection competitions. The experimental results show that our model obtains a performance of 0.386, within the range obtained by per-instance algorithm selection models in other combinatorial problems. Thus despite, being built with relatively computationally-cheap methods (i.e., decision trees and linear regression models), it proved to be useful for a portfolio of meta-heuristics with a relatively low-performance variation.

In the construction process of our per-instance selection model —structured according to the meta-learning framework— different elements of the Curriculum-Based Course Timetabling problem was analyzed, leading to relevant contributions to the current state of the art. These contributions include, *i)* the creation of a CB-CTT instance generator, *ii)* the formulation and evaluation of features to describe the main properties of timetabling instances, *iii)* the creation and evaluation of a portfolio of meta-heuristics, and *iv)* the design of a hybrid approach to build algorithm selector in low-performance portfolios. These contributions, and future research opportunities for the CB-CTT problem identified in this thesis, are explained in detail in the next chapter.

Conclusions

In this thesis, we have described the construction process of a per-instance algorithm selection model for solving educational timetabling instances, specifically for the problem called Curriculum-Based Course Timetabling (CB-CTT). The proposed model was constructed following the meta-learning framework, which maps the features that describe the main properties of a problem domain to the performance of a portfolio of algorithms in a structured manner. As a result of the process, different aspects of the CB-CTT problem domain were formally analyzed, leading to the following relevant contributions:

- The design of a parameterized CB-CTT *instance generator* to increase the size of benchmarking datasets which can be used to analyze the performance of future solving approaches.
- The formulation of a set of *complexity metrics* able to distinguish CB-CTT instance sub-spaces that share similar solving difficulty.
- The *performance evaluation* of a set solution methods proposed to solve CB-CTT instances.
- A *per-instance algorithm selection model* (based on the meta-learning framework), constructed to predict from a portfolio of algorithms, the one with the best-expected performance to solve a given CB-CTT instance.

In this chapter, we discuss the main findings obtained as a result of this thesis and provide directions for future research opportunities.

Main Findings

Throughout this thesis, we have introduced a new multi-model strategy to solve CB-CTT instances, modeled according to the XHSTT data format —a standard representation of timetabling instances based on an XML schema (see Section 1.2.2). This new strategy consists of the construction of a per-instance algorithm selection model to automatically select from a portfolio of algorithms, the one with the best expected performance to solve a given CB-CTT instance.

To construct the described model, we analyzed four different elements of the problem (i.e., instance, feature, algorithm, and performance spaces), and proposed multiple components (e.g., instance generator, metrics, performance measures) non-existent in the current literature. In this section, we discuss the main findings obtained from the overall construction process and present the general conclusions.

Instance Generator

A significant obstacle for research collaboration in the educational timetabling field is the scarce representation of instances in a standardized manner. Despite its higher modeling capability, the XHSTT data format used in this thesis has not been widely adopted in the current state of the art, as it is supported and maintained by a small group of researchers. As a result, only a limited benchmark of 25 instances have been collected to evaluate the performance of new solvers.

In this thesis, we created an XHSTT instance generator able to model 27 timetabling conditions using a combination of 152 parameters (see Chapter 3). In our experimental tests, this generator proved to be useful to produce instances of diverse solving complexity in short running times, on average, of 2.13 seconds per instance. These results indicate that this generator is effective enough to be applied in further research works to increase the instance space for the analysis of new solvers and the study of the effects that different timetabling conditions have over solution space of educational timetabling problems.

CB-CTT Metrics

As pointed out in Section 3.3.5, in the current state of the art, two types of XHSTT instance solvers can be distinguished: i) solvers proposed to generate initial solutions and, ii) solvers proposed to improve initial solutions. Therefore, two groups of CB-CTT metrics were formulated.

i) To characterize the hardness of generating initial solutions (see Chapter 4), we proposed the implementation of a feature selection-based methodology —within a problem space of 6,000 instances. We described the generated instances using four sets of metrics and constructed different regression models to predict the *penalty value* of their initial solutions. From the analysis of the most accurate regression models, the most relevant features were identified and interpreted. This interpretation led us to conclude that the initial hardness of CB-CTT instances is highly related to the percentage of reduction of the scheduling and allocation spaces (measured based on counting functions), and the slackness of resources (measured based on feature ratios).

ii)) To describe the initial solutions (see Section 5.4), we formulated three types of metrics, designed to characterize the initial arrangement of elements (i.e., times and resources) to be improved by perturbation-based meta-heuristics. These types of metrics consider the dispersion, similarity, and usage of resources in the timetables. The inclusion of these metrics increased the accuracy of our per-instance algorithm selection model, indicating their relevance for the prediction of the performance of solvers designed to improve the initial solutions of CB-CTT instances.

The information obtained from both groups of metrics proved to be useful to describe the combinatorial solution spaces represented in the XML trees containing the instances and their solutions. Thus, it can be applied to characterize the relevant properties of timetabling instances in further research works.

Algorithm Portfolio

Because of the scarce availability of solution methods, for the creation of the algorithm portfolio, we selected four competitive meta-heuristics that perturbs initial solutions using the same set of low-level heuristics (*LLHs*). Unlike related research works, in this thesis, we considered the performance variation within the portfolio of meta-heuristics, a relevant factor for the success of the per-instance algorithm selection model. Thus, we formally quantified it in terms of: *i)* the number of performance ties, and *ii)* the standard deviation of performance between the meta-heuristics.

The defined portfolio exhibited a low-performance variation in a significant amount of instances, mainly from the classes *Easy* and *Very Easy* (close to their optimal solution). Leading us to conclude that, within the timetabling domain, the diversity of the solution approaches in a portfolio depends more on the variety of *LLHs* than on the variety of exploration strategies.

Selection Performance

In the related algorithm selection literature, the performance of the models is often reported using the *accuracy.* This measure calculates the percentage of algorithms correctly predicted for solving a set of instances. However, it exhibits two significant drawbacks: *i)* a wrong evaluation of algorithm selections in case of performance ties, and *ii)* the impossibility to consider performance improvements due to suboptimal algorithm selections.

To overcome these limitations, in this thesis, the performance of our per-instance algorithm selection model was evaluated using the *closed SBS-VBS gap*, a measure used in international algorithm selection competitions. This standard measure allowed us to evaluate our model in an unbiased manner and compare it to the performance of similar models in the combinatorial domain. Because of these advantages, we regarded the *closed SBS-VBS gap* as an effective measure to assess the overall performance of per-instance algorithm selection models.

Algorithm Selection Model

This thesis presented the construction process of a per-instance algorithm selection model for the solution of CB-CTT instances according to the meta-learning framework.

The proposed model was designed to detect the relatively small performance variation of four meta-heuristics without adding significant computational effort to the solution of the instances. To meet this goal, a hybrid approach, combining the predictions of decision trees and linear regression models, was structured according to the First-Classification-Then-Regression forecasting technique.

In the defined instance dataset, this model achieved a selection performance $\hat{m}_s$ (*closed SBS-VBS gap*) of 0.386, similar to that obtained in other combinatorial problems, but variable across instance subspaces (as shown in Table 6.2). As explained, the empirical hardness of the instance subspaces implies potential gains of different magnitude that assigns —in the global assessment of the model— more weight to algorithm selection on instances with higher penalty values. In our experimental tests (see Section 6.4.2), the regression-based selectors struggled to make accurate predictions in the classes with the highest potential gains (*Class 6* and *Class 7*), reducing the global performance of our model to a greater extent. To examine the possible causes of these results, we compared the relative variability between the meta-heuristics in all the classes using the mean coefficient of variation (MCV), a unit-free measure of dispersion. This comparison indicated that the proposed regression-based selectors performed significantly better on instances with a MCV above 1% —as higher performance variations entail larger margin errors for the predictions of the selectors. Despite the lower performance variation within the portfolio, in the overall evaluation, our per-instance selection model proved to be useful to obtain better solutions than the single-best solver strategy using computationally cheap machine learning methods.

Future Work

Diverse aspects of the CB-CTT problem have been analyzed in this thesis, revealing future research opportunities to extend this work. Next, we discuss several of these opportunities, regarding four different aspects: *i)* data format, *ii)* instance characterization, *iii)* algorithms and heuristics, and *iv)* selection models.

Data Format

Due to the diverse conditions that define timetabling problems, in the current state of the art, there is not a clear consensus in the manner in which instances should be represented. Many formats have been used to formulate timetabling problems over the years. However, to facilitate research collaboration, there is a current need to develop a standard data format flexible enough to model the broad range of constraints defined by educational institutions worldwide.

The XHSTT data format employed in this thesis was selected because of its proven usefulness to represent instances from ten countries in the third international timetabling competition (run during 2011-2012). Because of its modeling capacities, this format can be used to create a definitive standard in the educational timetabling domain. However, to do so, three main requisites must be addressed in the future.

1. **Update Data Format.** The current XHSTT data format represents timetabling instances according to the hierarchical structure of an XML schema, used to ensure the syntactical integrity of the data. However, there are two significant disadvantages of this structure: i) relatively big instance sizes due to the large proportion of characters related to the format rather than the instance itself, and ii) considerable high parsing times. Currently, there is still room for improvement for the XHSTT format, which could be streamlined by the adoption of similar standard file formats like JavaScript Object Notation (JSON).

2. **Formulate New Types of Constraints.** The XHSTT format considers 16 types of constraints that can be applied to formulate a broad set of timetabling conditions. However, they were not able to represent all the conditions planned for the generated instances; for example, scheduling the starting time of all the lectures of a course every day at the same time. There is a research opportunity to increase the diversity of constraints included in the format, to accomplish this goal, a formal survey about the real-life conditions that define timetabling problems in educational institutions worldwide is required.

3. **Incorporate Graphical User Interfaces.** As observed in present surveys [10–12], current research works on the timetabling domain focus mainly on the development and testing of theoretical solution methods for particular instance datasets. However, not much progress has been made regarding the practical implementation of these methods.

   A primary requisite, not yet fulfilled, to make use of current algorithms and meta-heuristics in practical applications is the creation of graphical user interfaces (GUIs) to allow real users to solve instances without having expert knowledge. This requisite needs to be incorporated as an integral part of future standard formats to facilitate both: *i)* the process in which instances are translated into mathematical formulations, and *ii)* the process in which solutions are graphically represented. It is expected that useful GUIs will advance the representation of real timetabling instances, and lead to the creation of challenging datasets non-synthetically generated.

Instance Characterization

In this thesis, the metrics proposed to characterize the CB-CTT instance dataset were employed to predict the penalty value of the solutions with two purposes: i) estimate the empirical hardness of the instances, and ii) select the meta-heuristic with the best-expected performance to solve a given instance. However, there are still at least two possible applications for these metrics that remain to be explored in future research works.

1. **Guide Heuristic Perturbation**. Because of its nature, heuristic-based methods in the timetabling domain improve solutions by performing random perturbations in the arrangement of different elements (i.e., lecture scheduling and resource allocation); however, as the size of the solution spaces becomes larger, the success probability of this type of random permutations decreases. Therefore, to improve the rate of successful permutations —even in large solution spaces— the formulated CB-CTT metrics could be used to calculate the success probability of per-

mutations within different neighborhoods to perform first those with the highest success probability.

2. **Perform Sensitivity Analyses**. In this thesis, we presented a formal introductory analysis regarding the initial difficulty of the solution of timetabling instances based on the interpretation of empirical hardness models. This first analysis reported the size of scheduling and allocation spaces, and the slackness of resources, as the main factors of the hardness of the instances. Still, to extend these findings to practically support the decisions of educational planners, further sensitivity analyses are required to evaluate the effects that the availability of resources and application of constraints over the size and form of the solution space across the timetabling domain.

Besides these two applications, there is an additional research opportunity related to the applicability of the *counting functions* in which our collection of CB-CTT metrics are based. Counting functions are mathematical expressions used to calculate the number of possible combinations in which a given task can be performed. Therefore, in this thesis, they were formulated to determine the potential number of valid assignments of lectures to time slots (scheduling) and resources to lectures (allocation). Because of its nature, we formulated the set of defined counting functions to evaluate the 27 conditions presented in Appendix A. However, to spread its usage within the timetabling domain, this set needs to be extended to characterize instances defined by more diverse sets of requirements, like the ones included in the XHSTT-2014 dataset, collected from ten different countries.

Algorithms and Heuristics

In the current state of the art, two types of timetabling solvers can be found: i) solvers proposed to *generate* initial solutions and, ii) solvers proposed to *improve* initial solutions. This distinction is relevant because, even though both of them have been proposed to solve the CB-CTT instances, the first type starts the solution process from an empty timetable, while the second one starts from a non-optimal (but already full) timetable.

Because of the relatively scarce adoption of the XHSTT format, a research opportunity shared by both types is the development of new solvers to increase the diversity of the present *algorithm space*. However, both of them require to be diversified in different directions, as explained next.

1. **Generators of Initial Solutions.** Due to the NP-hard nature of many combinatorial problems [99], creating initial solutions for problem instances is a complex task, to which two different approaches are often applied: exact algorithms and construction heuristics. Exact algorithms (e.g., linear programming) are mathematical formulations that guarantee to find optimal solutions; however, they become impractical as the size of problem instances increases. Construction heuristics, on the other hand, are solution methods that attempt to obtain moderately good solutions at reasonable times.

   To the best of our knowledge, in the current literature, only two works have been proposed to generate initial solutions for timetabling instances in the XHSTT format: *i)* an integer programming formulation [72], and *ii)* the KHE solver [68] employed in this thesis. Because of its heuristic-based approach, the KHE solver proved to be useful to produce good initial solutions for the XHSTT-2014 dataset, in practical processing times (of less than 4 hours per instance). In contrast, the integer formulation, as stated by its authors, did not perform well in large instances and required larger running times (of 24 hours per instance) to get competitive solutions. As a consequence of these experimental results, the KHE solver has been used as the default solution approach in most of the related research works [70–73, 100]. However, there is a latent research opportunity to be explored, combining both types of approaches (exact and heuristic) for the creation of a hybrid solution strategy (known as *matheuristic*) that might be able to exceed the individual performance of these solution methods.

2. **Improvers of Initial Solutions.** One of the main conclusions of this thesis is

that for a portfolio to be effective, the performance variation between the algorithms across the problem space must be high. As explained in Chapter 5, in the combinatorial field, a popular approach for solving instances is the perturbation of initial solutions using meta-heuristics. Meta-heuristics are solution strategies that operate at two levels. At the low level, perturbation heuristics, known as low-level heuristics (*LLHs*), modify the arrangement of the elements of a timetable; at the high level, an exploration strategy, define the solution neighborhoods which are perturbed.

In the current literature, many exploration strategies have been applied to solve timetabling instances in the XHSTT format, but only a few *LLHs* to perturb the solutions. Therefore, there is a research opportunity to increase the set of available *LLHs*, that as concluded in this thesis, are the main source of diversity between meta-heuristics.

Selection Methods

The final result of this thesis is the algorithm selection model presented in Chapter 6. As explained in the chapter, different machine-learning processes were applied to analyze the collected meta-data and create the required selection mapping of instance features to algorithms. Therefore, in addition to the model, three research opportunities were identified as potential ways to improve algorithm selection in similar future works.

1. **Formulate Metrics for the Performance Variation.** The collection of CB-CTT metrics (144 metrics of seven different types) formulated to characterize both the instances and their initial solutions proved to be useful for the construction of regression models with low error rates. However, when applied to estimate the performance variation between the meta-heuristics, the combined effect of these low error rates, led to a significant percentage of sub-optimal algorithm selections.

   In the current state of the art, the general approach (adopted in this thesis) for the construction of per-instance algorithm selection models is the formulation of a descriptive set of features to estimate the performance of algorithms. Still, to improve the performance of the models, a new type of metrics specifically formulated to predict, not the performance, but the relative performance variation between the algorithms in a portfolio is a significant research opportunity to be addressed to minimize the adverse effects of combined error rates.

2. **Define Problem Sub-spaces.** As explained in Section 6.3, unlike other per-instance algorithm selection models for similar combinatorial problems, in this thesis, the prediction of algorithms was performed using multiple selectors, one for each type of seven defined classes. Because of the adoption of the First-Classification-Then-Regression forecasting technique, CB-CTT instances were classified according to the mean penalty value of their improved solutions —a measure of their empirical hardness. This classification allowed us to analyze the performance of the meta-heuristics in different problem sub-spaces, then to train algorithm selectors accordingly.

   Despite the usefulness of this novel approach, the classification step for this process was performed only on a statistical analysis of the performance of the algorithms without considering other potentially relevant features. Therefore, it remains for further research testing alternative approaches, such as clustering methods, to perform this classification considering a broader set of similarity factors to define problem sub-spaces.

3. **Evaluate the Potential Success of Selection Models.** Because of the popularity of machine learning methods and data mining techniques, the implementation of per-instance algorithm selection models has increased in the current literature. However, not many of the reported research works include a detailed analysis of the performance variation of the algorithms in a portfolio. A relevant analysis to evaluate the suitability for the construction of these models.

   A comprehensive evaluation of suitability must consider two main questions: *i) "what is the potential improvement of performance that a selection model would obtain* *over the application of the single-best algorithm in a given instance space?"*, and *ii) "how accurate a selection model should be to perform accurate selections in a defined algorithm portfolio?"* To the best of our knowledge, at present, no formal way to answer both questions has been proposed. Therefore, there is a research opportunity to formulate a statistical framework for this evaluation in future related works. Because of its relevance, this framework could be potentially adopted by the research community as a standard evaluation for the construction of per-instance algorithm selection models in the future.

Appendices

Appendix A

Conditions Modeled by the Timetabling Instance Generator

In Chapter 3, we described the design of the generator employed to create the dataset of timetabling of instances modeled according to the XHSTT format. In the following tables, we present the set of conditions that can be modeled by this generator.

	Condition	**Description**
1	Assign teachers	Classes must have an assigned teacher
2	Assign rooms	Classes must have an assigned room
3	Assign times	Classes must be scheduled the number of time slots required by its duration
4	Avoid clashes	Resources (i.e. curricula, teachers, and rooms) must not attend to different lectures at the same time
5	Split theory event	Specifies the valid set of lectures configurations in which classes of theory courses (which do not require a laboratory) can be scheduled
6	Split practice event	Specifies the valid set of configurations in which classes of practice courses (which require a laboratory) can be scheduled
7	Prefer teachers	Defines the subset of classes to which teachers can be allocated
8	Prefer rooms	Defines the subset of classes to which rooms and laboratories can be allocated
9	Prefer times	Limits a randomly selected percentage of classes to be scheduled only on a user-defined set of days
10	Teachers stability	Requires that all lectures derived from a class must be allocated to the same teacher
11	Rooms stability	Requires that all lectures derived from a class must be allocated to the same room
12	Courses stability	Requires that all lectures derived from a class which requires both a classroom (for theory instruction) and a laboratory (for practice activities) are allocated the same teacher
13	Single lecture	Requires that for each class only one lecture be scheduled per day
14	Daily lecture	Requires that a randomly selected percentage of classes be scheduled in a daily basis, within a user-defined set of days.
15	Link events	Requires that all lectures derived from a set of classes be scheduled simultaneously. This set of classes is defined by randomly selecting a class from each one of the active terms in the curricular plan
16	Working shifts	Requires that teachers be allocated only in the set of time slots defined by their work shifts
17	Study shifts	Requires that curricula be allocated only in the set of time slots defined by their study shifts
18	Idle times of part-time teachers	Specifies the range of daily idle time slots that part-time teachers must have

	Condition	Description
19	Idle times of curricula	Specifies the range of daily idle time slots that curricula must have
20	Busy days of full-time teachers	Specifies the number of working days that full-time teachers must give classes
21	Busy days of part-time teachers	Specifies the number of working days that part-time teachers must give classes
22	Busy days of curricula	Specifies the number of days that curricula must attend to classes
23	Daily workload of full-time teachers	Specifies the number of daily time slots that full-time teachers must give classes
24	Daily workload of part-time teachers	Specifies the number of daily time slots that part-time teachers must give classes
25	Daily workload of curricula	Specifies the number of daily time slots that curricula must attend to classes
26	Weekly workload of full-time teachers	Limits the number of weekly time slots that full-time teachers can be allocated
27	Weekly workload of part-time teachers	Limits the number of weekly time slots that part-time teachers can be allocated

-

Appendix B

Timetabling Modeling Using the XHSTT Data Format

XHSTT Instance Modeling of the Timetabling problem in the Computer Science Department at the UASLP.

Authors:
M.I.P. Felipe de la Rosa Rivera
Ph.D. José Ignacio Nuñez Varela

This document describes the process followed for solving the timetabling problem in the computer science department at the School of Engineering of the Autonomous University of San Luis Potosí (UASLP). The timetabling problem is modeled using the XHSTT format, an XML based schema proposed as an international standard for the third competition of timetable problems (ITC-2011).

This document has two purposes: i) explore the modeling capabilities of the XHSTT format for the formulation of a real instance within the Mexican context, and ii) illustrate the modeling and solution processes required to define an XHSTT instance. The formulation of an XHSTT instance requires four essential elements that must be defined according to the order followed in this document: *Times*, *Resources*, *Events*, and *Constraints*. For explanation purposes, each element is explained in a different section, then the solution process of the instance and results are discussed. The described XHSTT instance can be download from the following link:
https://www.dropbox.com/s/isn4a6j9jfdhmop/Instance_XHSTT_UASLP.xml?dl=0

1. TIMES

In the XHSTT format, the concept of *Times* refers to each of the intervals defined to perform the scheduling task (i.e., assign times to lectures). To illustrate this concept, consider the next table, which shows the *time grid* set for this instance —composed of five days and thirteen one-hour intervals (from 07:00 to 20:00). Each header (row and column) represents a *Timegroup* (i.e., a group of *Times* belonging to the same category), and each cell, a *Time* that must be defined in the instance.

	07-08	**08-09**	**09-10**	**. . .**	**17-18**	**18-19**	**19-20**
Lun	Lun_07-08	Lun_08-09	Lun_09-10	. . .	Lun_17-18	Lun_18-19	Lun_19-20
Mar	Mar_07-08	Mar_08-09	Mar_09-10	. . .	Mar_17-18	Mar_18-19	Mar_19-20
Mie	Mie_07-08	Mie_08-09	Mie_09-10	. . .	Mie_17-18	Mie_18-19	Mie_19-20
Jue	Jue_07-08	Jue_08-09	Jue_09-10	. . .	Jue_17-18	Jue_18-19	Jue_19-20
Vie	Vie_07-08	Vie_08-09	Vie_09-10	. . .	Vie_17-18	Vie_18-19	Vie_19-20

As observed, the *time grid* for this timetabling problem was defined 'horizontally' (i.e., days represented as rows, and time intervals as columns) because of the limitations of the software employed to visualize the instance and its solution (HSEval[1]), and the set of *Constraints* available in the XHSTT format.

[1] http://jeffreykingston.id.au/cgi-bin/hseval.cgi

In addition of the *Times* we defined six additional *Timegroups*, illustrated in the next table, to further apply timetabling constraints related to scheduling. In color green, a *Timegroup* called “Matutino” (further divided into two *Timegroups*, “Matutino_1” and “Matutino_2”) which encompass all *times* before 13:00; in color yellow, a *Timegroup* called “Vespertino” (further divided into two *TimeGroups* “Vespertino_1” and “Vespertino_2”) which encompass all *Times* after 13:00.

	07-08	**08-09**	**09-10**	**10-11**	**11-12**	**12-13**	**13-14**	**14-15**	**15-16**	**16-17**	**17-18**	**18-19**	**19-20**
Lun													
Mar													
Mie	Matutino_1			Matutino_2			Vespertino_1			Vespertino_2			
Jue													
Vie													

In the described instance, we defined first the *Timegroups* using the following syntax. (Note: remember that we defined the *time grid* ‘horizontally’, thus time intervals were defined as “days” —to be represented in columns— and days as *TimeGroups* —to be represented in rows).

```
<TimeGroups>
    <Day Id="H_07-08"><Name>Horario 07-08</Name></Day>
    <Day Id="H_08-09"><Name>Horario 08-09</Name></Day>
                    .
                    .
                    .
     <Day Id="H_18-19"><Name>Horario 18-19</Name></Day>
     <Day Id="H_19-20"><Name>Horario 19-20</Name></Day>
     <TimeGroup Id="Lun"><Name>Lunes</Name></TimeGroup>
     <TimeGroup Id="Mar"><Name>Martes</Name></TimeGroup>
     <TimeGroup Id="Mie"><Name>Miercoles</Name></TimeGroup>
     <TimeGroup Id="Jue"><Name>Jueves</Name></TimeGroup>
     <TimeGroup Id="Vie"><Name>Viernes</Name></TimeGroup>
     <TimeGroup Id="Matutino"><Name>Horario 07-13</Name></TimeGroup>
     <TimeGroup Id="Vespertino"><Name>Horario 13-20</Name></TimeGroup>
     <TimeGroup Id="Matutino_1"><Name>Horario 07-10</Name></TimeGroup>
     <TimeGroup Id="Matutino_2"><Name>Horario 10-13</Name></TimeGroup>
     <TimeGroup Id="Vespertino_1"><Name>Horario 13-16</Name></TimeGroup>
     <TimeGroup Id="Vespertino_2"><Name>Horario 16-20</Name></TimeGroup>
</TimeGroups>
```

After setting the *TimeGroups*, we defined each *Time* using the following syntax:

```
<Time Id="Lun_07-08">
    <Name>Lun_07-08</Name>
    <Day Reference="H_07-08"/>
    <TimeGroups>
        <TimeGroup Reference="Lun"/>
        <TimeGroup Reference="Matutino_1"/>
        <TimeGroup Reference="Matutino"/>
    </TimeGroups>
</Time>
```

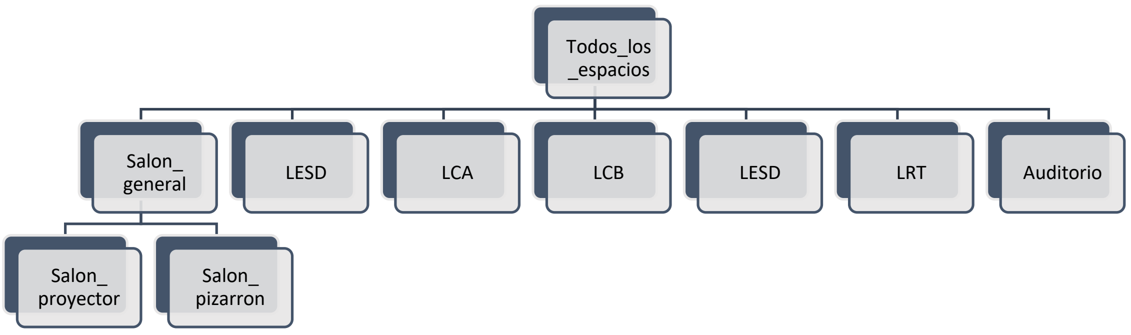
2. RESOURCES

A resource is defined as an element that must be present during a lecture. Commonly, at least three resources of a different type are required to be present during lecture: a physical space (room), a professor, and a curriculum (i.e., group of students sharing the same classes). For the UASLP timetabling instance, we defined these three *ResourceTypes* (Salon, Profesor, Generacion), at the beginning of the *Resources* section —as required by the XHSTT format— using the following syntax:

```
<ResourceTypes>
        <ResourceType Id="Salon"><Name>Tipo de recurso Salon</Name></ResourceType>
        <ResourceType Id="Profesor"><Name>Tipo de recurso Profesor</Name></ResourceType>
        <ResourceType Id="Generacion"><Name>Alumnos que comparten materias</Name></ResourceType>
</ResourceTypes>
```

2.1 Rooms

Prior to the inclusion of the rooms able to be assigned to the lectures, several *ResourceGroups* were defined to group rooms with similar features. The next figure shows the classification of rooms created with the *ResourceGroups.*



The *ResourceGroups* for the rooms were defined with the following syntax:

```
<ResourceGroups>
     <ResourceGroup Id="Salon_general">
          <Name>Salon con o sin proyecto</Name>
          <ResourceType Reference="Salon"/>
     </ResourceGroup>
     <ResourceGroup Id="Salon_proyector">
           <Name>Salon equipado con proyector</Name>
           <ResourceType Reference="Salon"/>
     </ResourceGroup>
     <ResourceGroup Id="Salon_pizarron">
           <Name>Salon NO equipado con proyector</Name>
           <ResourceType Reference="Salon"/>
      </ResourceGroup>
      <ResourceGroup Id="LESD">
           <Name>Laboratorio tipo LESD</Name>
           <ResourceType Reference="Salon"/>
      </ResourceGroup>
```

```
    <ResourceGroup Id="LCA">
        <Name>Laboratorio tipo LCA</Name>
        <ResourceType Reference="Salon"/>
    </ResourceGroup>
    <ResourceGroup Id="LCB">
        <Name>Laboratorio tipo LCB</Name>
        <ResourceType Reference="Salon"/>
    </ResourceGroup>
    <ResourceGroup Id="LRT">
        <Name>Laboratorio tipo LRT</Name>
        <ResourceType Reference="Salon"/>
    </ResourceGroup>
    <ResourceGroup Id="Auditorio">
      <Name>Salon con gran capacidad</Name>
      <ResourceType Reference="Salon"/>
    </ResourceGroup>
    <ResourceGroup Id="Todos_los_espacios">
        <Name>Todos los espacios fisicos para las clases</Name>
        <ResourceType Reference="Salon"/>
    </ResourceGroup>
<ResourceGroups>
```

After defining the *ResourceGroups*, each room was added to the instance using the following syntax:

```
<Resource Id="Salon I-01">
    <Name>Salon I-01</Name>
    <ResourceType Reference="Salon"/>
    <ResourceGroups>
        <ResourceGroup Reference="Salon_pizarron"/>
        <ResourceGroup Reference="Salon_general"/>
        <ResourceGroup Reference="Todos_los_espacios"/>
     </ResourceGroups>
</Resource>
```

As observed, besides the “Id”, “Name”, and “ResourceType”, each room is assigned to one or more *ResourceGroups.* For example, due to its features, the presented “Salon I-01” belongs to three *ResourceGroups:* “Salon_pizarron”, “Salon_general”, and “Todos_los_espacios”.

2.2 Professors

As with the rooms, different *ResourceGroups* were defined to group professors with similar features. Professors were grouped according to their expertise areas to define the set of classes they can give. The expertise areas correspond to those designated for bachelor’s degree programs in the field of computer science. A total of nine *ResourceGroups* were defined:

```
<ResourceGroup Id="Profesor_innovacion_y_desarrollo_tecnologico">
    <Name>Profesor de academia de innovacion y desarrollo tecnologico</Name>
    <ResourceType Reference="Profesor"/>
</ResourceGroup>
<ResourceGroup Id="Profesor_ingenieria_de_software_y_base_de_datos">
    <Name>Profesor de academia de ingenieria de software y bases de datos</Name>
    <ResourceType Reference="Profesor"/>
</ResourceGroup>
```

```
<ResourceGroup Id="Profesor_interaccion_y_videojuegos">
    <Name>Profesor de academia de interaccion y videojuegos</Name>
    <ResourceType Reference="Profesor"/>
</ResourceGroup>
<ResourceGroup Id="Profesor_matematicas_para_la_computacion">
    <Name>Profesor de academia de matematicas para la computacion</Name>
    <ResourceType Reference="Profesor"/>
</ResourceGroup>
<ResourceGroup Id="Profesor_programacion">
    <Name>Profesor de academia de programacion</Name>
    <ResourceType Reference="Profesor"/>
</ResourceGroup>
<ResourceGroup Id="Profesor_redes_y_ciberseguridad">
    <Name>Profesor de academia de redes y seguridad</Name>
    <ResourceType Reference="Profesor"/>
</ResourceGroup>
<ResourceGroup Id="Profesor_robotica_inteligente">
    <Name>Profesor de academia de robotica inteligente</Name>
    <ResourceType Reference="Profesor"/>
</ResourceGroup>
<ResourceGroup Id="Profesor_sistemas_de_hardware">
    <Name>Profesor de academia de sistemas de hardware</Name>
    <ResourceType Reference="Profesor"/>
</ResourceGroup>
<ResourceGroup Id="Profesor_tecnologias_multiplataforma">
    <Name>Profesor de academia de tecnologias multiplataforma</Name>
    <ResourceType Reference="Profesor"/>
</ResourceGroup>
```

Additionally, all the professors were included in a general *ResoureGroup* called “Todos_los_ profesores”.

```
<ResourceGroup Id="Todos_los_profesores">
    <Name>Todos los profesores que imparten clases</Name>
    <ResourceType Reference="Profesor"/>
</ResourceGroup>
```

After defining all *ResourceGroups* for the professors, each professor was added to the instance using the following syntax:

```
<Resource Id="29500">
    <Name>Altamirano Flores Jose Salomon</Name>
    <ResourceType Reference="Profesor"/>
    <ResourceGroups>
        <ResourceGroup Reference="Profesor_matematicas_para_la_computacion"/>
        <ResourceGroup Reference="Profesor_robotica_inteligente"/>
        <ResourceGroup Reference="Profesor_programacion"/>
        <ResourceGroup Reference="Todos_los_profesores"/>
    </ResourceGroups>
</Resource>
```

2.3 Curricula

As explained before, a curriculum refers to a group of students who must attend the same set of classes. Curricula are defined to ensure that students can enroll in the classes they require without clashes. In Spanish, we use the term “Generacion” to refer to a curriculum.

As shown next, only one *ResourceGroup* was defined, including all the curricula required to solve the instance.

```
<ResourceGroup Id="Todas_las_generaciones">
    <Name>Todas las generaciones de alumnos</Name>
    <ResourceType Reference="Generacion"/>
</ResourceGroup>
```

Each curriculum was included in the only *ResourceGroup* created to group curricula “Todas_las _generaciones”, and defined using the following syntax:

```
<Resource Id="Gen IC Sem-2 Gpo-1">
    <Name>Generacion IC Semestre-2 Grupo-1</Name>
    <ResourceType Reference="Generacion"/>
    <ResourceGroups>
        <ResourceGroup Reference="Todas_las_generaciones"/>
    </ResourceGroups>
</Resource>
```

3. EVENTS

Once all *Times* and *Resources* are set, they can be used to define the *Events* (i.e., classes to be scheduled). Each *Event* requires an “Id”, a “Name”, and a “Duration” that indicates the number of *times* that it must be weekly scheduled.

```
<Event Id="283001">
    <Name>Telematica A - Grupo 1</Name>
    <Duration>5</Duration>
```

If the *Event* has already a *preassigned time* —does not require being scheduled— the optional element “Time” must be included to specify the starting time of the *Event*. But, if it does not have a *preassigned time*, the element “Time” must be omitted. For example, the following syntax indicates that the *Event* "283001", have a *preassigned time*, thus it must start in the *Time* "Lun_07-08"

```
    <Time Reference="Lun_07-08"/>
```

After indicating *preassigned times*, if they exist, the resources required to be present in the lectures (meetings) of the *Event* are next defined. For explanation purposes, consider the *Resources* defined for the *Event* "283001", shown next:

```
    <Resources>
        <Resource Reference="Gen IC Sem-10 Gpo-1"/>
        <Resource>
            <Role>1</Role>
            <ResourceType Reference="Salon"/>
        </Resource>
        <Resource>
            <Role>2</Role>
            <ResourceType Reference="Profesor"/>
        </Resource>
    </Resources>
</Event>
```

Resources can be both preassigned or expected to be assigned, depending on the syntax used to define them. On the one hand, when the "Reference" attribute is present, it references a resource preassigned to the event. On the other hand, when the "Reference" attribute is absent, it indicates that a resource assignment is required.

The pre-assignment of resources was employed to indicate the curriculum to which each class in the instance belongs. Hence, as observed in this example, the "Reference" attribute was used to indicate that the *Event* "283001" belongs to the curriculum "Gen IC Sem-10 Gpo-1".

To perform a resource assignment, each *Event* must indicate the "Role" and "ResourceType" of the required *Resource.* The attribute "Role" is an identifier that specifies the function that the required resource will have within the *Event*, and "ResourceType" indicates the type of resource required. In the described instance, "Role 1" identifies the role of a room, whereas "Role 2", the role of a professor. As observed in the example, two resources are required: a *Resource* of the type "Salon" that will have "Role 1" (room) in the *Event*, and a *Resource* of the type "Profesor", that will have "Role 2" (professor).

4. CONSTRAINTS

Constraints are *hard* (mandatory) and *soft* (optional) conditions to be satisfied when performing the assignment of *Times* and *Resources* to the *Events*. The solution quality of an instance depends on the fulfillment of *Constraints*, therefore, all *Constraints* are evaluated and included in a cost function that assigns a penalty (or weight) to each constraint violation. Optimal solutions are those that fulfill all *Constraints*; feasible solutions are those that fulfill all hard constraints; and, unfeasible solutions are those that violate at least one hard constraint.

The XHSTT format includes sixteen types of *Constraints*, seven of them related to scheduling (i.e., time assignment) and nine of them related to allocation (i.e., resource assignment). All of them defined using the general syntax described next:

```
<AnyConstraint Id>
    <Name>
    <Required>
    <Weight>
    <CostFunction>
    <AppliesTo>
```

Like other elements, each *Constraint* requires an “Id” and a “Name”. The attribute “Required” can take two values, *true* to indicate that the fulfillment of the *Constraint* is *hard*, or false to indicate that is *soft*. “Weight” is an attribute that defines the penalty for each *Constraint* violation, this “Weight” must be an integer in the range (0, 1000]. “CostFunction” is an attribute that defines the way in which the penalties of constraint violations will be added to the cost function; it can be set as “Linear”, “Quadratic”, or “Step”. “AppliesTo” is an attribute that defines the elements of the instance to which the *Constraint* applies. Next, we describe the *Constraints* used to define the timetabling instance for the computer science department at the UASLP.

4.1 AssignResourceConstraint

The *AssignResourceConstraint* requires the assignment of *Resources* to *Events* that do not have a preassigned resource. We included two *Constraints* of this type, the first to request the assignment of professors (with a “Role 2”) to *Events*, the second to request the assignment of rooms (with a “Role 1”) to *Events*. Both constraints were defined as hard, with a penalty value of 1000.

```
<AssignResourceConstraint Id="Asignar_profesores">
    <Name>Asignar profesores a las clases</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Events>

            {List of events without a preassigned professor}

        </Events>
    </AppliesTo>
    <Role>2</Role>
</AssignResourceConstraint>


<AssignResourceConstraint Id="Assignar_salones">
    <Name>Asignar salones a las clases</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Events>

            {List of events without a preassigned room}

        </Events>
    </AppliesTo>
    <Role>1</Role>
</AssignResourceConstraint>
```

4.2 AssignTimeConstraint

The *AssignTimeConstraint* requires the assignment of *Times* to *Events* that do not have a preassigned time. We included a constraint of this type, also defined as hard, as shown next.

```
<AssignTimeConstraint Id="Assignar_horarios">
    <Name>Asignar horarios a las clases</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Events>

            {List of events without a preassigned time}

        </Events>
    </AppliesTo>
</AssignResourceConstraint>
```

4.3 SplitEventsConstraint

In the XHSTT format, *Events* (classes) are split into *sub-events* (lectures) which correspond to the total "duration" defined for each event. For example, an *Event* called "Finanzas" with a weekly duration of three *Times* can be split into the following lecture configurations: three one-*time* lectures (1-1-1), a two-*time* lecture and a one-*time* lecture (2-1), and a single three-*time* lecture (3).

A significant limitation of the XHSTT format is that once an *Event* (class) is split into *subevents* (lectures), it does not provide a *Constraint* to ensure that *subevents* are scheduled in a *time-stable* manner. Consider the two timetables presented next, which illustrate two different schedules for the *Event* "Finanzas", split into three one-time lectures (1-1-1).

	07-08	08-09	09-10
Lun	Finanzas		
Mar			Finanzas
Mie		Finanzas	
Jue			
Vie			

	07-08	08-09	09-10
Lun		Finanzas	
Mar		Finanzas	
Mie		Finanzas	
Jue			
Vie			

In the left table, the sub-events are scheduled each day at a different time. In the right table, the events are scheduled each day at the same time —in a *time-stable* manner— which is a crucial requirement for the solution of UASLP timetabling instance.

To preserve *time-stability*, we use the *SplitEventsConstraint* to avoid that *Events* be split into *sub-events*. Using this strategy, all *Events* were set as single lectures required to be consecutively scheduled in a *time-stable* manner in different days. Due to the 'horizontal' layout defined for the instance, each day a one-*time* lecture will be scheduled for each class.

A *SplitEventsConstraint* limits the number of *sub-events* that may be derived from a given *Event*, and on their durations. To avoid the split of Events into sub-events, we defined the following *Constraint*:

```
<SplitEventsConstraint Id="No_dividir_sesiones">
     <Name>Evitar que las clases se separen en sesiones</Name>
     <Required>true</Required>
     <Weight>1000</Weight>
     <CostFunction>Linear</CostFunction>
     <AppliesTo>
          <Events>

               {List of events that must not be split}

          </Events>
     </AppliesTo>
     <MinimumDuration>1</MinimumDuration>
     <MaximumDuration>5</MaximumDuration>
     <MinimumAmount>1</MinimumAmount>
     <MaximumAmount>1</MaximumAmount>
</AssignResourceConstraint>
```

The attributes “MinimumDuration” and “Maximum duration”, defines the possible “duration” for the *sub-events* that can be derived from an *Event,* whereas the attributes “MinimumAmount” and “MaximumAmount” limits the number of *sub-events*. As defined in this constraint, all the Events must be single lectures, with a “duration” from 1 to 5 *Times.*

4.4 PreferResourcesConstraint

The *PreferResourcesConstraint* specifies that some *Resources* are preferred to be assigned to certain *Events*. We used this constraint to define the type of professor and room required by each class.

As explained in Section 2.2, professors were grouped according to their expertise area in nine *ResouceGroups* defined for bachelors’ degrees in computer science. For the proper assignment of professors to the *Events*, we defined a different *PreferResourcesConstraint* for each expertise area. As an example, next we present the syntax employed to require the assignment of a professor with expertise in the area of “programacion”.

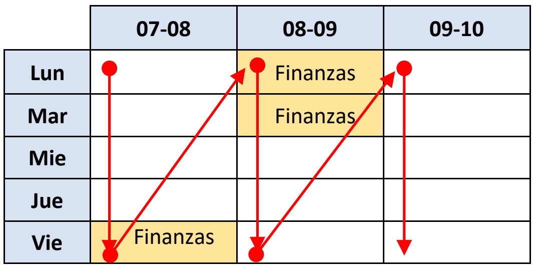
```
<PreferResourcesConstraint Id="Seleccionar_profesor_programacion">
     <Name>Seleccionar profesor de academia de programacion</Name>
     <Required>true</Required>
     <Weight>1000</Weight>
     <CostFunction>Linear</CostFunction>
     <AppliesTo>
          <Events>

            {List of events that requires the assignment of a professor with expertise in programacion}

          </Events>
     </AppliesTo>
     <ResourceGroups>
          <ResourceGroup Reference="Profesor_programacion"/>
     </ResourceGroups>
     <Role>2</Role>
</PreferResourcesConstraint>
```

In a similar manner, the *PreferResourcesConstraint* was employed to require the assignment of proper rooms to the events. For the proper assignment of rooms to the *Events*, we defined a different *PreferResourcesConstraint* for each type of room. As an example, we present the syntax employed to require the assignment of a laboratory LCA for the *Events*.

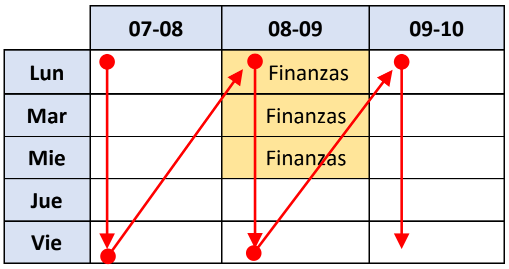
```
<PreferResourcesConstraint Id="Seleccionar_laboratorio_LCA">
    <Name>Seleccionar laboratorio LCA</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
    <Events>

          {List of events that requires the assignment of a LCA laboratory}

    </Events>
    </AppliesTo>
    <ResourceGroups>
        <ResourceGroup Reference="LCA"/>
    </ResourceGroups>
    <Role>1</Role>
</PreferResourcesConstraint>
```

4.5 PreferTimesConstraint

The *Times* defined for an XHSTT instance are considered a list of consecutive time intervals, following the sequence in which they were defined. In the formulated instance, *Times* were defined in the sequence indicated by the red arrows sown in the next tables. Because of the sequence of *Times*, *Events* were prone to be scheduled in an unstable time manner, starting on one column and ending on another.



	07-08	08-09	09-10
Lun		Finanzas	
Mar		Finanzas	
Mie			
Jue			
Vie	Finanzas		



	07-08	08-09	09-10
Lun		Finanzas	
Mar		Finanzas	
Mie		Finanzas	
Jue			
Vie			

The left table shows an example of a single lecture (with a duration of three times) scheduled in an unstable manner, staring in the *Time* "Vie_07-08" and ending in the *Time* "Mar_08-09". The right table shows a *time-stable* scheduling of the same event, starting in the time Lun_08-09 and ending in the Time Mie_08-09.

To ensure a *time-stable* scheduling of the *Events*, we employed the *PreferTimesConstraint.* Using this constraint, all the *Events* required to be scheduled one-*time* a day —each day at the same *Time*— were required to start on the row "Lun", thus, avoiding being assigned to more than one column.

The *PreferTimesConstraint* specifies that some *Times* are preferred to be assigned to certain *Events*. Therefore, *Events* were required to start in the row "Lun" using the following syntax.

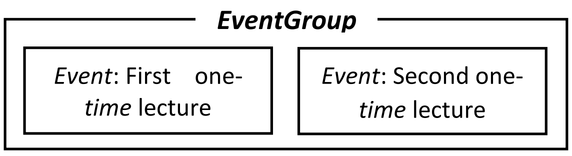
```
<PreferTimesConstraint Id="Primera_sesion_en_lunes">
    <Name>Primera sesion de clase en dia lunes</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Events>
            {List of events that requires to be scheduled one-time a day in a time stable manner}
        </Events>
    </AppliesTo>
    <TimeGroups>
        <TimeGroup Reference="Lun"/>
    </TimeGroups>
```

4.6 AvoidSplitAssignementConstraint

Though most of the *Events* require to be scheduled *one-time* per day, some of them —with a weekly duration of two *Times*— require to be scheduled as two-*times Events*. *Two-times Events* are expected to be scheduled in a single day, thus were formulated using a different strategy structured in three steps: *i)* defining the Events, *ii)* linking *Resources*, *iii)* linking *Times*. The first two steps are explained in this section, the third step is explained in the next section: "OrderEventsConstraint".

<u>Defining the *Events*</u>

Two-*times Events* were first split and defined as independent *Events* with a "duration" of one *time.* Then, these independent *Events* were grouped into the same *EventGroup* to further link them through the application of constraints. The next figure illustrates the structure defined for two-*times Events* in the XHSTT instance.



For example, prior to defining the one-*time* independent *Events* required by the *Event* "282001-L1" called "Redes A - Laboratorio - Grupo 1", we defined the following *EventGroup*:

```
<EventGroup Id="282001-L1">
    <Name>282001-L1</Name>
</EventGroup>
```

Then, we defined two one-*time Events* and included them into the created *EventGroup* as follows:

```
<Event Id="282001-L1-H1">
    <Name>Redes A - Laboratorio - Grupo 1 - H1</Name>
    <Duration>1</Duration>
    <Resources>
        <Resource Reference="Gen IC Sem-9 Gpo-1"/>
        <Resource>
            <Role>1</Role>
            <ResourceType Reference="Salon"/>
        </Resource>
        <Resource>
            <Role>2</Role>
            <ResourceType Reference="Profesor"/>
        </Resource>
    </Resources>
    <EventGroups>
        <EventGroup Reference="282001-L1"/>
    </EventGroups>
</Event>

<Event Id="282001-L1-H2">
    <Name>Redes A - Laboratorio - Grupo 1 – H2</Name>
    <Duration>1</Duration>
    <Resources>
        <Resource Reference="Gen IC Sem-9 Gpo-1"/>
        <Resource>
            <Role>1</Role>
            <ResourceType Reference="Salon"/>
        </Resource>
        <Resource>
            <Role>2</Role>
            <ResourceType Reference="Profesor"/>
        </Resource>
    </Resources>
    <EventGroups>
        <EventGroup Reference="282001-L1"/>
    </EventGroups>
</Event>
```

Using this syntax, the two one-*time Events* "282001-L1-H1" and "282001-L1-H2" were defined —belonging to the same *EventGroup* ("282001-L1") and requiring the same type of *Resources*.

Linking Resources

To ensure that the independent one-*time Events*, formulated to model two-*times Events*, be assigned the same resources (i.e., has the same professor and room), we employed the *AvoidSplitAssignementConstraint*. This constraint requires that the *Resources* assigned to all the *Events* in an *EventGroup* be the same.

We defined two *AvoidSplitAssignementConstraints*, one for the assignment of professors, and one for the assignment of rooms. The syntax of both constraints is presented next.

Constraint that requires the assignment of the same professor to both one-*time Events* of an *EventGroup*.

```
<AvoidSplitAssignmentsConstraint Id="Profesor_clases_dos_horas">
    <Name>Asignar el mismo profesor a clases de dos horas</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <EventGroups>

            {List of EventGroups defined to group the Events of two-times lectures}

        </EventGroups>
    </AppliesTo>
    <Role>2</Role>
</AvoidSplitAssignmentsConstraint>
```

Constraint that requires the assignment of the same room to both one-*time Events* of an *EventGroup*.

```
<AvoidSplitAssignmentsConstraint Id="Salon_clases_dos_horas">
    <Name>Asignar el mismo salon a clases de dos horas</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <EventGroups>

            {List of EventGroups defined to group the Events of two-times lectures}

        </EventGroups>
    </AppliesTo>
    <Role>1</Role>
</AvoidSplitAssignmentsConstraint>
```

4.7 OrderEventsConstraint

The O*rderEventsConstraint* requires that the *Times* of two events be scheduled so that the first *Event* ends before the second *Event* begins. Its syntax is

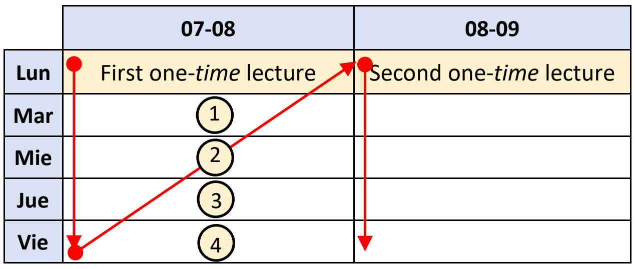
```
<OrderEventsConstraint Id=>
    <Name>
    <Required>
    <Weight>
    <CostFunction>
    <AppliesTo>
        <EventPairs>
```

This type of constraint is applied to *EventPairs*, where each *EventPair* must be defined as follows:

```
<EventPair>
      <FirstEvent Reference>
      <SecondEvent Reference>
      <MinSeparation>
      <MaxSeparation>
```

“FirstEvent” and “SecondEvent” contain references to the two *Events* whose times are to be linked. “MinSeparation” is the minimum number of *Times* that may separate the two events. “MaxSeparation” is the maximum number of times that may separate the two events.

The next table illustrates the example of a two-*times Event* required to be scheduled the same day as a single two-*times* lecture. As explained in the previous section, this type of *Event* was formulated using two independent one-*time Events* grouped in the same *EventGroup*. To ensure that both independent one-*time Events* have the same resources (professors and rooms), we applied the *AvoidSplitAssignementConstraint*; to ensure that they be consecutively scheduled, we use the *OrderEventsConstraint*.



	07-08	08-09
Lun	First one-*time* lecture	Second one-*time* lecture
Mar	1	
Mie	2	
Jue	3	
Vie	4	

As seen in the Table, in the defined sequence of *Times* (represented by the red arrows) the *Times* “Lun_07-08” and “Lun_08-09” are not consecutive but separated by a “distance” of four *Times* (represented with yellow circles); these are: “Mar_07-08”, “Mie_07-08”, “Jue_07-08”, “Vie_07-08”. Thus, to consecutively schedule the “Second one-*time* lecture” the same day after the “First one-*time* lecture”, we must define a separation of four *Times* between both lectures.

The consecutive scheduling of the first and second lectures of two-*times Events* was defined using the *OrderEventsConstraint*. This constraint was applied to ensure that the second lectures of two-*times Events* start four *Times* after the scheduling of their related first lectures. Next, we present the syntax of the constraint employed with this purpose.

```
<OrderEventsConstraint Id="Sesiones de dos horas">
      <Name>Ordenar sesiones de dos horas</Name>
      <Required>true</Required>
      <Weight>1000</Weight>
      <CostFunction>Linear</CostFunction>
      <AppliesTo>
            <EventPairs>
                  <EventPair>
                        <FirstEvent Reference={First one-time lecture of the Event}>
                        <SecondEvent Reference={Second one-time lecture of the Event}>
                        <MinSeparation>4</MinSeparation>
                        <MaxSeparation>4</MaxSeparation>
                  </EventPair>
```

```
        </EventPairs>
    </AppliesTo>
</OrderEventsConstraint>
```

As a result, of our three-step strategy (*defining the events*, *linking resources*, and *linking times*), all two-*times Events* were successfully formulated as two-time lectures.

4.8 AvoidClashesConstraint

AvoidClashesConstraint is a basic constraint applied to avoid that resources have clashes; that is, that they are scheduled to more than one *Event* simultaneously. As clashes of resources are not acceptable in the formulated instance, all the defined resources (rooms, professors and curricula), were included in a single *AvoidClashesConstraint* using the syntax presented next.

```
<AvoidClashesConstraint Id="Evitar_conflictos_de_recursos">
    <Name>Evitar conflictos de recursos</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Resources>

            {list of all rooms}
            {list of all professors}
            {list of all curricula}

        <Resources>
    </AppliesTo>
</AvoidClashesConstraint>
```

4.9 ClusterBusyTimesConstraint

The *ClusterBusyTimesConstraint* limits the number of *TimeGroups* during which a resource may be busy. In the formulated instance we employ this type of constraint to reduce the dispersion of the *Events* assigned to professors and students.

Consider the next table that shows the lectures (represented as gray squares) assigned to a professor. It may be desirable that the professor has a more "compact" distribution of lectures. For example, that all his lectures be scheduled in only one of two possible shifts, "Matutino" (in green) or "Vespertino" (in yellow), which were defined as different *TimeGroups* in Section 1:TIMES.

	07-08	08-09	09-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20
Lun	⬛	🟩	🟩	⬛	🟩	🟩	🟨	🟨	⬛	🟨	⬛	🟨	🟨
Mar	⬛	🟩	🟩	⬛	🟩	⬛	⬛	🟨	⬛	🟨	⬛	🟨	🟨
Mie	⬛	🟩	🟩	⬛	🟩	🟩	🟨	🟨	⬛	🟨	⬛	🟨	🟨
Jue	⬛	🟩	🟩	⬛	🟩	🟩	🟨	🟨	⬛	🟨	⬛	🟨	🟨
Vie	🟩	🟩	🟩	⬛	🟩	🟩	🟨	🟨	⬛	🟨	⬛	🟨	🟨

To require that the lectures assigned to the professors be assigned in only of the two defined *TimeGroups,* "Matutino" (Before 13:00) or "Vespertino" (After 13:00), we defined the following constraint.

```
<ClusterBusyTimesConstraint Id="Compactar_horarios_de_profesores_turno">
    <Name>Compactar horarios de profesores en un turno</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Resources>

            {List of professors}

        </Resources>
    </AppliesTo>
    <TimeGroups>
        <TimeGroup Reference="Matutino"/>
        <TimeGroup Reference="Vespertino"/>
    </TimeGroups>
    <Minimum>0</Minimum>
    <Maximum>1</Maximum>
</ClusterBusyTimesConstraint>
```

As observed, this constraint requires that the *Events* assigned to the professors be clustered in only one of two possible *TimeGroups*, "Matutino" or "Vespertino". This requirement is set using the attributes "Minimum" —with a value of zero— and "Maximum" —with a value of one— which indicates that at most one of the *TimeGroups* be used for *Event* scheduling of the professors.

Similarly, a second *ClusterBusyTimesConstraint* was defined to require that the *Events* assigned to each curriculum be clustered only in the "Matutino" or "Vespertino" shift. As students have more flexibility to attend to disperse scheduled *Events*, we set this constraint as optional —with a "Required" value set as "false" and a lower "Weight" with a value of 100.

```
<ClusterBusyTimesConstraint Id="Compactar_horarios_de_alumnos_turno">
    <Name>Compactar horarios de alumnos en un turno</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Resources>

            {List of curricula}

        </Resources>
    </AppliesTo>
    <TimeGroups>
        <TimeGroup Reference="Matutino"/>
        <TimeGroup Reference="Vespertino"/>
    </TimeGroups>
    <Minimum>0</Minimum>
    <Maximum>1</Maximum>
</ClusterBusyTimesConstraint>
```

Note: Depending on time availability of each professor, further *ClusterBusyTimesConstraints* can be applied to schedule the *Events* in more specific *TimeGroups*; such as the “Matutino_1”, “Matutino_2”, “Vespertino_1”, and “Vespertino_2”, described in Section 1:TIMES.

4.10 LimitWorkloadConstraint

Each *Event* has an associated *workload,* which for this instance corresponds to its duration. *Resources* are assigned to the *Events* without considering their *workload* unless a *LimitWorkloadConstraint* is defined.

The *LimitWorkloadConstraint* specifies the total *workload* that can be assigned to *Resources*, using the following syntax:

```
<LimitWorkloadConstraint Id>
    <Name>
    <Required>
    <Weight>
    <CostFunction>
    <AppliesTo>
    <Minimum>
    <Maximum>
```

The attribute “AppliesTo” indicates the *Resources* whose *workload* be limited. The workload of these *Resources* may range between the “Minimum” and “Maximum” values. In the formulated instance, two *LimitWorkloadConstraints* were defined. The first one, to set the *workload* of full-time professors (“profesor tiempo completo”), and the second one, to set the *workload* of part-time professors (“professor por asignatura”). Next, we present the syntax of both constraints.

```
<LimitWorkloadConstraint Id="Limitar_profesor_tiempo_completo">
    <Name>Limitar horas de clase de profesor de tiempo completo</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Resources>

            {List of full-time professors}

        </Resources>
    </AppliesTo>
    <Minimum>8</Minimum>
    <Maximum>10</Maximum>
</LimitWorkloadConstraint>
```

```
<LimitWorkloadConstraint Id="Limitar_profesor_por_asignatura">
    <Name>Limitar horas de clase de profesor por asignatura</Name>
    <Required>true</Required>
    <Weight>1000</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Resources>

            {List of part-time professors}

        </Resources>
    </AppliesTo>
    <Minimum>4</Minimum>
    <Maximum>25</Maximum>
</LimitWorkloadConstraint>
```

5. SOLUTION

The XHSTT format structures timetabling archives in two major branches. The first branch, *Instance*, corresponds to the XML tree that defines the formulation of the timetabling problem; the second branch, *Solution*, to the XML tree that describes the assignment of *Times* and *Resources* for all the *Events* of such a problem. The global syntax of a XHSTT timetabling archive is represented as follows:

```
<HighSchoolTimetableArchive>
    <Instance>
        {Definition of Times, Resources, Events and Constraints}
    </Instance>
    <Solution>
        {Assignment of Times and Resources to the Events}
    </Solution>
</HighSchoolTimetableArchive>
```

The described XHSTT instance can be download from the following link: https://www.dropbox.com/s/zkrw0kcpixz2qmi/Solution_XHSTT_UASLP.xml?dl=0

The formulated *Instance* (first branch of the XHSTT archive), was first syntactically and logically validated using the *KHE Timetabling Engine*[2] (a popular XHSTT solver), then also applied to produce an initial *Solution* for the *Instance*. In the next table, we summarize the constraint violations found in this initial *Solution* (second branch of the XHSTT archive).

Constraint	**Type**	**Violations**	**Penalty Value**
Assign Time Constraint	Hard	8	8,000
Cluster Busy Times Constraint	Hard	23	23,000
Cluster Busy Times Constraint	Soft	4	400
Limit Workload Constraint	Hard	107	107,000
		Total	138,400

[2] http://jeffreykingston.id.au/khe/

The total *penalty value* (i.e., the sum of the “Weights” of hard and constraint violations) is 138,400, a high value far from its optimal solution (i.e., a solution with a penalty value of zero). As observed, most violations correspond to *LimitWorkloadConstraints* defined to set the *workload* range for “Full-time” and “Part-time” teachers, which were particularly challenging for the *KHE Timetabling Engine*.

To improve the quality of the initial Solution, we applied a perturbation-based meta-heuristic called *Variable Neighborhood Search* (*VNS*). This meta-heuristic was proposed by the GOAL Team[3] of the Federal University of Ouro Preto and proved to be effective in previous experimental tests performed over a synthetically generated dataset of instances. For the improvement of the initial *Solution*, we defined a maximum running time of 3600 seconds. The constraint violations found in the *Solution* improved with the *VNS* meta-heuristic are presented next.

Constraint	Type	Violations	Penalty Value
Cluster Busy Times Constraint	Hard	1	1,000
Cluster Busy Times Constraint	Soft	3	300
		Total	1300

As observed, this improved *Solution* is very close to the optimal. Still, as it incurs in the violation of one *hard* constraint, it is considered as *unfeasible*. Thus, it remains to be further employed with alternative solution methods beyond the scope of this document.

The XML tree of a *Solution* is a list of the assignments of *Resources* and *Times* required by all the *Events.* The *Events* are described using the syntax shown in the next example, which corresponds to the *Event* "215001" ("Temas selectos de matematicas - Grupo 1").

```
<Event Reference="215001">
    <Duration>4</Duration>
    <Time Reference="Lun_13-14"/>
    <Resources>
        <Resource Reference="Salon I-11"><Role>1</Role></Resource>
        <Resource Reference="9779"><Role>2</Role></Resource>
    </Resources>
</Event>
```

The first attribute of an *Event* is its “Duration”, that if omitted corresponds to the entire duration of the *Event* —defined in the XML tree of the *Instance*. *Time Reference* indicates the starting *Time* assigned to each *Event*, while *Resource References* the resources assigned to perform a given “Role” (1 – room, 2 - professor) within the *Event*. It is important to point out that *preassigned Times* and *preassigned Resources* are not included as part of the *Solution* as they are conditions defined in the formulation of the *Instance*.

The scheduling of the *Events* can be visualized in the timetables of their associated *Resources*. For this example, the *Resources* associated to this event are three, a *preassigned Resource* curriculum "Generacion ISI-IC Semestre-1 Grupo-1"; and two *Resources* assigned during the solution process, the room "Salon I-11", and the professor "9979". Hence, this event must be included in the

[3] http://www.goal.ufop.br/software/hstt/

timetables of these resources, as illustrated in the next table: starting at the *Time* "Lun_13-14" and, according to its “Duration”, ending on the *Time* "Jue_13-14".

	07-08	08-09	09-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20
Lun							215001						
Mar							215001						
Mie							215001						
Jue							215001						
Vie													

An additional tool created by Jeffrey H. Kingston, author of the *KHE Timetabling Engine*, is the HSEval[4], a web-based evaluator of XHSTT archives. The HSEval asks for an XHSTT file and returns an HTML page with detailed reports and visual representations of the timetables contained in the *Solution* of an *Instance.*

Next, we present an example of the timetable generated with the HSEval for the Curriculum "Generacion IC Semestre-9 Grupo-1". In this timetable, “days” are represented as rows but have no headers.

Horario 07-08	**Horario 08-09**	**Horario 09-10**	**Horario 10-11**	**Horario 11-12**	**Horario 12-13**	**Horario 13-14**	**Horario 14-15**	**Horario 15-16**	**Horario 16-17**	**Horario 17-18**	**Horario 18-19**	**Horario 19-20**
	Computacion ubicua - Grupo 1		Administracion de bases de datos - Grupo 1	Sistemas de informacion A - Grupo 1	Computacion ubicua - Grupo 2	Redes A - Grupo 1			Redes A - Laboratorio - Grupo 1 - H1	Redes A - Laboratorio - Grupo 1 - H2	Proyecto integrador - Grupo 1	Proyecto integrador - Grupo 2
	Computacion ubicua - Grupo 1		Administracion de bases de datos - Grupo 1	Sistemas de informacion A - Grupo 1	Computacion ubicua - Grupo 2	Redes A - Grupo 1			Seminario I.C., I.I. - Grupo 1		Proyecto integrador - Grupo 1	Proyecto integrador - Grupo 2
	Computacion ubicua - Grupo 1		Administracion de bases de datos - Grupo 1	Sistemas de informacion A - Grupo 1	Computacion ubicua - Grupo 2	Redes A - Grupo 1					Proyecto integrador - Grupo 1	Proyecto integrador - Grupo 2
	Computacion ubicua - Grupo 1		Administracion de bases de datos - Grupo 1	Sistemas de informacion A - Grupo 1	Computacion ubicua - Grupo 2	Redes A - Grupo 1				Redes A - Laboratorio - Grupo 2 - H1	Redes A - Laboratorio - Grupo 2 - H2	
	Computacion ubicua - Grupo 1		Administracion de bases de datos - Grupo 1	Sistemas de informacion A - Grupo 1	Computacion ubicua - Grupo 2	Redes A - Grupo 1						

As observed, the timetable of this curriculum has assigned lectures in two *TimeGroups*, “Matutino” (before 13:00) and “Vespertino” (after 13:00), which constitutes a constraint violation for one of the defined ClusterBusyTimesConstraints. This constraint violation is reported by the HSEval, just below the presented timetable as follows:

Cluster Busy Times Constraint	**Constraint name**	**Point of application**	**Calculation**	**Inf.**	**Obj.**
Compactar_horarios_de_alumnos_turno	Compactar horarios de alumnos en un turno	Generacion IC Semestre-9 Grupo-1	100 * Linear(Matutino + Vespertino - 1)		100
Total (1 point)					100

[4] http://jeffreykingston.id.au/cgi-bin/hseval.cgi

6. SUMMARY AND LIMITATIONS

In this document, we have explained the formulation of an instance in the XHSTT format to solve a real timetabling problem. The set of elements included in the format (i.e., Times, Resources, Events, and Constraints) proved to be useful for modeling the timetabling conditions defined for the computer science department at the School of Engineering of the UASLP. However, as the XHSTT format lacks a constraint to require that lectures be scheduled in a *time-stable* manner (each day at the same time), we defined alternative strategies to model classes using a 'horizontal' time grid —days represented as rows and times as columns.

For solving the formulated instance, we applied a two-step solution approach. First, we generated an initial solution using the KHE timetabling engine, which incurred in many constraint violations, mainly related to the weekly *workload* defined for the professors. To improve the initial solution, as the second step, we applied the VNS meta-heuristic with a maximum running time of 3600 seconds. As a result, we obtained an improved solution with few constraint violations, nearly optimal.

Despite the promising results obtained with this formulation and its solution, it is important to point out that modeling classes using the described strategies limits the scheduling of lectures to consecutive timeslots. Hence, it does not allow the possibility of schedule alternate lectures in a *time-stable* manner. For example, scheduling a class with a weekly duration of three times, on alternate days, Monday (Lun), Wednesday (Mie), and Friday (Vie) at the same hour: Lun_08-09, Mie_08-09, Vie_08-09.

Finally, it is important to mention that as no user interfaces have been developed to handle XHSTT archives, currently, most of the time required to model and solve a timetabling instance is involved with data entry. Thus, to practically support the work of educational planners, this limitation is a significant work opportunity to be addressed in the future.

Bibliography

[1] L. Kotthoff, "Algorithm selection for combinatorial search problems: A survey," *Ai Magazine*, vol. 35, no. 3, pp. 48–60, 2014.

[2] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta, *Metalearning: Applications to data mining.* Springer Science & Business Media, 2008.

[3] S. Bouajaja and N. Dridi, "A survey on human resource allocation problem and its applications," *Operational Research*, pp. 1–31, 2016.

[4] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. D. Gaspero, R. Qu, and E. K. Burke, "Setting the research agenda in automated timetabling: The second international timetabling competition," *INFORMS Journal on Computing*, vol. 22, no. 1, pp. 120–130, 2010.

[5] T. B. Cooper and J. H. Kingston, "The complexity of timetable construction problems," in *International Conference on the Practice and Theory of Automated Timetabling*, pp. 281–295, Springer, 1995.

[6] J. R. Rice, "The algorithm selection problem," *Advances in computers*, vol. 15, pp. 65–118, 1976.

[7] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla: portfolio-based algorithm selection for sat," *Journal of artificial intelligence research*, vol. 32, pp. 565–606, 2008.

[8] T. Messelis and P. De Causmaecker, "An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 233, no. 3, pp. 511–528, 2014.

[9] C. Ansótegui, J. Gabas, Y. Malitsky, and M. Sellmann, "Maxsat by improved instance-specific algorithm configuration," *Artificial Intelligence*, vol. 235, pp. 26–39, 2016.

[10] N. Pillay, "A survey of school timetabling research," *Annals of Operations Research*, vol. 218, no. 1, pp. 261–293, 2014.

[11] S. MirHassani and F. Habibi, "Solution approaches to the course timetabling problem," *Artificial Intelligence Review*, pp. 1–17, 2013.

[12] V. Sahargahi and M. Drakhshi, "Comparing the methods of creating educational timetable," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 16, no. 12, p. 26, 2016.

[13] M. A. Muñoz, Y. Sun, M. Kirley, and S. K. Halgamuge, "Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges," *Information Sciences*, vol. 317, pp. 224–245, 2015.

[14] D. H. Wolpert, W. G. Macready, *et al.*, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[15] A. Loreggia, Y. Malitsky, H. Samulowitz, and V. A. Saraswat, "Deep learning for algorithm portfolios.," in *AAAI*, pp. 1280–1286, 2016.

[16] M. Lindauer, J. N. van Rijn, and L. Kotthoff, "The algorithm selection competitions 2015 and 2017," *Artificial Intelligence*, vol. 272, pp. 86–100, 2019.

[17] G. L. Pappa, G. Ochoa, M. R. Hyde, A. A. Freitas, J. Woodward, and J. Swan, "Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms," *Genetic Programming and Evolvable Machines*, vol. 15, no. 1, pp. 3–35, 2014.

[18] C. Lemke, M. Budka, and B. Gabrys, “Metalearning: a survey of trends and technologies,” *Artificial intelligence review*, vol. 44, no. 1, pp. 117–130, 2015.

[19] L. Di Gaspero, B. McCollum, and A. Schaerf, “The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3),” tech. rep., Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1. 0, Queen’s University, Belfast, United Kingdom, 2007.

[20] A. Bettinelli, V. Cacchiani, R. Roberti, and P. Toth, “An overview of curriculum-based course timetabling,” *Top*, vol. 23, no. 2, pp. 313–349, 2015.

[21] A. Bonutti, F. De Cesco, L. Di Gaspero, and A. Schaerf, “Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results,” *Annals of Operations Research*, vol. 194, no. 1, pp. 59–70, 2012.

[22] G. Post, L. Di Gaspero, J. H. Kingston, B. McCollum, and A. Schaerf, “The third international timetabling competition,” *Annals of Operations Research*, vol. 239, no. 1, pp. 69–75, 2016.

[23] G. Post, J. H. Kingston, S. Ahmadi, S. Daskalaki, C. Gogos, J. Kyngas, C. Nurmi, N. Musliu, N. Pillay, H. Santos, *et al.*, “Xhstt: an xml archive for high school timetabling problems in different countries,” *Annals of Operations Research*, vol. 218, no. 1, pp. 295–301, 2014.

[24] C. Gotlieb, “The construction of class-teacher timetables,” in *Proc. IFIP Congress*, vol. 62, pp. 73–77, 1963.

[25] D. J. Welsh and M. B. Powell, “An upper bound for the chromatic number of a graph and its application to timetabling problems,” *The Computer Journal*, vol. 10, no. 1, pp. 85–86, 1967.

[26] N. L. Lawrie, “An integer linear programming model of a school timetabling problem,” *The Computer Journal*, vol. 12, no. 4, pp. 307–316, 1969.

[27] P. V. Hentenryck and V. Saraswat, "Constraint programming: Strategic directions," *Constraints*, vol. 2, no. 1, pp. 7–33, 1997.

[28] L. Zhang and S. Lau, "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, 2005.

[29] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & operations research*, vol. 13, no. 5, pp. 533–549, 1986.

[30] R. Alvarez-Valdes, E. Crespo, and J. M. Tamarit, "Design and implementation of a course scheduling system using tabu search," *European Journal of Operational Research*, vol. 137, no. 3, pp. 512–523, 2002.

[31] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *et al.*, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[32] E. Aycan and T. Ayav, "Solving the course scheduling problem using simulated annealing," in *2009 IEEE International Advance Computing Conference*, pp. 462–466, IEEE, 2009.

[33] A. Schaerf and L. Di Gaspero, "Local search techniques for educational timetabling problems," in *Proceedings of the 6th International Symposium on Operational Research (SOR-01), Preddvor, Slovenia*, pp. 13–23, 2001.

[34] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & operations research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[35] S. Abdullah, E. K. Burke, and B. McCollum, "A hybrid evolutionary approach to the university course timetabling problem," in *2007 IEEE congress on evolutionary computation*, pp. 1764–1768, IEEE, 2007.

[36] J. H. Holland, "Adaptation in natural and artificial systems. an introductory analysis with application to biology, control, and artificial intelligence," *Ann Arbor, MI: University of Michigan Press*, 1975.

[37] H. Babaei, J. Karimpour, and A. Hadidi, "A survey of approaches for university course timetabling problem," *Computers & Industrial Engineering*, vol. 86, pp. 43–59, 2015.

[38] O. M. K. Alsmadi, S. Za'er, D. I. Abu-Al-Nadi, and A. Algsoon, "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, 2011.

[39] M. Dorigo, "Optimization, learning and natural algorithms," *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[40] C. Nothegger, A. Mayer, A. Chwatal, and G. R. Raidl, "Solving the post enrolment course timetabling problem by ant colony optimization," *Annals of Operations Research*, vol. 194, no. 1, pp. 325–339, 2012.

[41] P. Moscato and C. Cotta, "A modern introduction to memetic algorithms," in *Handbook of metaheuristics*, pp. 141–183, Springer, 2010.

[42] S. N. Jat and S. Yang, "A memetic algorithm for the university course timetabling problem," in *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, vol. 1, pp. 427–433, IEEE, 2008.

[43] L. Jourdan, M. Basseur, and E.-G. Talbi, "Hybridizing exact methods and metaheuristics: A taxonomy," *European Journal of Operational Research*, vol. 199, no. 3, pp. 620–629, 2009.

[44] P. Kostuch, "The university course timetabling problem with a three-phase approach," in *International Conference on the Practice and Theory of Automated Timetabling*, pp. 109–125, Springer, 2004.

[45] P. Cowling, G. Kendall, and E. Soubeiga, “A hyperheuristic approach to scheduling a sales summit,” in *International conference on the practice and theory of automated timetabling*, pp. 176–190, Springer, 2000.

[46] J. A. Soria-Alcaraz, G. Ochoa, J. Swan, M. Carpio, H. Puga, and E. K. Burke, “Effective learning hyper-heuristics for the course timetabling problem,” *European Journal of Operational Research*, vol. 238, no. 1, pp. 77–86, 2014.

[47] K. A. Smith-Miles, “Towards insightful algorithm selection for optimisation using meta-learning concepts,” in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 4118–4124, ieee, 2008.

[48] L. Kotthoff, P. Kerschke, H. Hoos, and H. Trautmann, “Improving the state of the art in inexact tsp solving using per-instance algorithm selection,” in *International Conference on Learning and Intelligent Optimization*, pp. 202–217, Springer, 2015.

[49] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, “Automated algorithm selection: Survey and perspectives,” *Evolutionary computation*, vol. 27, no. 1, pp. 3–45, 2019.

[50] K. Smith-Miles and T. T. Tan, “Measuring algorithm footprints in instance space,” in *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pp. 1–8, IEEE, 2012.

[51] J. Kanda, A. de Carvalho, E. Hruschka, C. Soares, and P. Brazdil, “Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features,” *Neurocomputing*, vol. 205, pp. 393–406, 2016.

[52] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis, “Towards objective measures of algorithm performance across instance space,” *Computers & Operations Research*, vol. 45, pp. 12–24, 2014.

[53] A. L. D. Rossi, A. C. P. de Leon Ferreira, C. Soares, B. F. De Souza, *et al.*, "Metastream: A meta-learning based method for periodic algorithm selection in time-changing data," *Neurocomputing*, vol. 127, pp. 52–64, 2014.

[54] C. Cui, T. Wu, M. Hu, J. D. Weir, and X. Li, "Short-term building energy model recommendation system: a meta-learning approach," *Applied Energy*, vol. 172, pp. 251–263, 2016.

[55] C. Cui, M. Hu, J. D. Weir, and T. Wu, "A recommendation system for meta-modeling: A meta-learning based approach," *Expert Systems with Applications*, vol. 46, pp. 33–44, 2016.

[56] M. Reif and F. Shafait, "Efficient feature size reduction via predictive forward selection," *Pattern Recognition*, vol. 47, no. 4, pp. 1664–1673, 2014.

[57] M. Kozielski, "A meta-learning approach to methane concentration value prediction," in *International Conference: Beyond Databases, Architectures and Structures*, pp. 716–726, Springer, 2015.

[58] L. P. Garcia, A. C. de Carvalho, and A. C. Lorena, "Noise detection in the meta-learning level," *Neurocomputing*, vol. 176, pp. 14–25, 2016.

[59] D. G. Ferrari and L. N. De Castro, "Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods," *Information Sciences*, vol. 301, pp. 181–194, 2015.

[60] M. Reif, F. Shafait, M. Goldstein, T. Breuel, and A. Dengel, "Automatic classifier selection for non-experts," *Pattern Analysis and Applications*, vol. 17, no. 1, pp. 83–96, 2014.

[61] R. Priya, B. F. de Souza, A. L. Rossi, and A. C. de Carvalho, "Predicting execution time of machine learning tasks using metalearning," in *2011 World Congress on Information and Communication Technologies*, pp. 1193–1198, IEEE, 2011.

[62] A. L. Dantas and A. T. R. Pozo, "A meta-learning algorithm selection approach for the quadratic assignment problem," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, IEEE, 2018.

[63] A. D. de León, E. Lalla-Ruiz, B. Melián-Batista, and J. M. Moreno-Vega, "Meta-learning-based system for solving logistic optimization problems," in *International Conference on Computer Aided Systems Theory*, pp. 339–346, Springer, 2017.

[64] L. M. Pavelski, M. R. Delgado, and M.-É. Kessaci, "Meta-learning on flowshop using fitness landscape analysis," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 925–933, ACM, 2019.

[65] A. E. Gutierrez-Rodríguez, S. E. Conant-Pablos, J. C. Ortiz-Bayliss, and H. Terashima-Marín, "Selecting meta-heuristics for solving vehicle routing problems with time windows via meta-learning," *Expert Systems with Applications*, vol. 118, pp. 470–481, 2019.

[66] U. o. T. DMPP Group, "Overview XHSTT-2014 (Instances and best solutions)." `https://www.utwente.nl/en/eemcs/dmmp/hstt/archives/XHSTT-2014/overview.html`, 2014. Last accessed 2017-05-05.

[67] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. Resende, and W. R. Stewart, "Designing and reporting on computational experiments with heuristic methods," *Journal of heuristics*, vol. 1, no. 1, pp. 9–32, 1995.

[68] J. H. Kingston, "A Software Library for High School Timetabling." `http://www.it.usyd.edu.au/~jeff/khe/`, 2016. Retrieved on November 2016.

[69] A. Kheiri, E. Ozcan, and A. J. Parkes, "Hysst: hyper-heuristic search strategies and timetabling," in *Proceedings of the ninth international conference on the practice and theory of automated timetabling (PATAT 2012)*, pp. 497–499, Citeseer, 2012.

[70] S. S. Brito, G. H. Fonseca, T. A. Toffolo, H. G. Santos, and M. J. Souza, “A sa-vns approach for the high school timetabling problem,” *Electronic Notes in Discrete Mathematics*, vol. 39, pp. 169–176, 2012.

[71] G. H. Fonseca and H. G. Santos, “Variable neighborhood search based algorithms for high school timetabling,” *Computers & Operations Research*, vol. 52, pp. 203–208, 2014.

[72] S. Kristiansen, M. Sørensen, and T. R. Stidsen, “Integer programming for the generalized high school timetabling problem,” *Journal of Scheduling*, vol. 18, no. 4, pp. 377–392, 2015.

[73] G. H. Fonseca, H. G. Santos, E. G. Carrano, and T. J. Stidsen, “Integer programming techniques for educational timetabling,” *European Journal of Operational Research*, vol. 262, no. 1, pp. 28–39, 2017.

[74] K. Leyton-Brown, E. Nudelman, and Y. Shoham, “Learning the empirical hardness of optimization problems: The case of combinatorial auctions,” in *International Conference on Principles and Practice of Constraint Programming*, pp. 556–572, Springer, 2002.

[75] A. Soghier and R. Qu, “Adaptive selection of heuristics for assigning time slots and rooms in exam timetables,” *Applied Intelligence*, vol. 39, no. 2, pp. 438–450, 2013.

[76] C. K. Teoh, A. Wibowo, and M. S. Ngadiman, “Review of state of the art for metaheuristic techniques in academic scheduling problems,” *Artificial Intelligence Review*, vol. 44, no. 1, pp. 1–21, 2015.

[77] K. Smith-Miles and L. Lopes, “Generalising algorithm performance in instance space: a timetabling case study,” in *International Conference on Learning and Intelligent Optimization*, pp. 524–538, Springer, 2011.

[78] H. Ochiai, T. Kanazawa, K. Tamura, and K. Yasuda, "Combinatorial optimization method based on hierarchical structure in solution space," *Electronics and Communications in Japan*, vol. 99, no. 8, pp. 25–37, 2016.

[79] R. Stanley, "Enumerative combinatorics vol. 1, cambridge univ," *Press, Cambridge*, 1997.

[80] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *Journal of machine learning research*, vol. 5, no. Oct, pp. 1205–1224, 2004.

[81] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[82] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual predictions with feature contributions," *Knowledge and information systems*, vol. 41, no. 3, pp. 647–665, 2014.

[83] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, pp. 4768–4777, 2017.

[84] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[85] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[86] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, vol. 29, pp. 1189–1232, 2001.

[87] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, *et al.*, "A comparison of the performance of different metaheuristics on the timetabling problem," in *International Conference on the Practice and Theory of Automated Timetabling*, pp. 329–351, Springer, 2002.

[88] P. Kostuch and K. Socha, "Hardness prediction for the university course timetabling problem," in *European Conference on Evolutionary Computation in Combinatorial Optimization*, pp. 135–144, Springer, 2004.

[89] K. Smith-Miles, R. James, J. Giffin, and Y. Tu, "Understanding the relationship between scheduling problem structure and heuristic performance using knowledge discovery," *Learning and Intelligent Optimization, LION*, vol. 3, 2009.

[90] N. Rodriguez-Maya, J. J. Flores, and M. Graff, "Predicting the rcga performance for the university course timetabling problem," in *International Symposium on Intelligent Computing Systems*, pp. 31–45, Springer, 2016.

[91] G. H. G. da Fonseca, Santos, *et al.*, "GOAL solver: a hybrid local search based solver for high school timetabling," *Annals of Operations Research*, vol. 239, no. 1, pp. 77–97, 2016.

[92] G. H. Fonseca, H. G. Santos, and E. G. Carrano, "Late acceptance hill-climbing for high school timetabling," *Journal of Scheduling*, vol. 19, no. 4, pp. 453–465, 2016.

[93] Y. Dong, W. Cheng, and S. Li, "A new regression method based on svm classification," in *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, vol. 2, pp. 770–773, IEEE, 2011.

[94] R. Amadini, M. Gabbrielli, and J. Mauro, "Portfolio approaches for constraint optimization problems," *Annals of Mathematics and Artificial Intelligence*, vol. 76, no. 1-2, pp. 229–246, 2016.

[95] L. Kotthoff, B. Hurley, and B. O'Sullivan, "The icon challenge on algorithm selection," *AI Magazine*, vol. 38, no. 2, pp. 91–93, 2017.

[96] M. Lindauer, J. N. van Rijn, and L. Kotthoff, "Open algorithm selection challenge 2017: Setup and scenarios," in *Open Algorithm Selection Challenge 2017*, pp. 1–7, 2017.

[97] M. Lindauer, J. N. van Rijn, and L. Kotthoff, "Open algorithm selection challenge 2017: Setup and scenarios," in *Proceedings of the Open Algorithm Selection Challenge* (M. Lindauer, J. N. van Rijn, and L. Kotthoff, eds.), vol. 79 of *Proceedings of Machine Learning Research*, (Brussels, Belgium), pp. 1–7, PMLR, 11–12 Sep 2017.

[98] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, *et al.*, "Aslib: A benchmark library for algorithm selection," *Artificial Intelligence*, vol. 237, pp. 41–58, 2016.

[99] M. R. Garey and D. S. Johnson, *Computers and intractability*, vol. 174. freeman San Francisco, 1979.

[100] L. N. Ahmed, E. Özcan, and A. Kheiri, "Solving high school timetabling problems worldwide using selection hyper-heuristics," *Expert Systems with Applications*, vol. 42, no. 13, pp. 5463–5471, 2015.