UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Solving Scheduling Problems under Disruptions

Alexandre Duarte de Almeida Lemos

Supervisor : **Doctor** Maria Inês Camarate de Campos Lynce de Faria
Co-Supervisor : **Doctor** Pedro Tiago Gonçalves Monteiro

Thesis approved in public session to obtain the PhD Degree in Computer Science and Engineering

Jury final classification: Pass with Distinction

2021

TÉCNICO LISBOA

UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Solving Scheduling Problems under Disruptions

Alexandre Duarte de Almeida Lemos

Supervisor : **Doctor** Maria Inês Camarate de Campos Lynce de Faria
Co-Supervisor : **Doctor** Pedro Tiago Gonçalves Monteiro

Thesis approved in public session to obtain the PhD Degree in Computer Science and Engineering

Jury final classification: Pass with Distinction

Jury

Chairperson : Doctor Mário Jorge Gaspar da Silva, Instituto Superior Técnico, Universidade de Lisboa

Members of the Committee :

Doctor Maria Inês Camarate de Campos Lynce de Faria, Instituto Superior Técnico, Universidade de Lisboa

Doctor Francisco António Chaves Saraiva de Melo, Instituto Superior Técnico, Universidade de Lisboa

Doctor Emir Demirović, Delft University of Technology, Holanda

Doctor Maria Margarida da Silva Carvalho, Faculté des arts et sciences, Université de Montréal, Canadá

Doctor Ricardo Lopes de Saldanha, SISCOD - Sistemas Cognitivos, SA

Funding Institutions -

Fundação para a Ciência e a Tecnologia, Departamento de Engenharia Informática of IST, Universidade de Lisboa

2021

Resumo

Os problemas de escalonamento são comuns em muitas aplicações desde fábricas a transportes passando por universidades. Muitas vezes são problemas de otimização onde queremos gerir da melhor forma recursos escassos. Como a realidade é dinâmica, disrupções inesperadas podem ocorrer e invalidar a solução original dum problema de escalonamento. Na literatura existem muitos métodos para lidar com incertezas. Estes podem ser resumidos em duas abordagens principais: (i) criar soluções robustas que não são afetadas pelas disrupções mais comuns; (ii) voltar a resolver o problema de raiz com as novas restrições.

O objectivo de criar soluções robustas é garantir que são válidas mesmo depois de ocorrerem as disrupçoes mais comuns. Por isso, requerem um estudo detalhado para prever os cenários disruptivos mais prováveis. A principal desvantagem é que a solução robusta pode ter menos qualidade (*e.g.* custo financeiro, satisfação dos clientes) para suportar uma disrupção que, embora provável, pode nunca chegar a acontecer. Independentemente da robustez da solução, podemos sempre necessitar de voltar a resolver o problema.

A maior parte dos métodos desenvolvidos com o objetivo de recuperar soluções após acontecerem disrupções limita-se a voltar a resolver o mesmo problema de raiz. Neste caso, utiliza-se uma função de custo adicional para garantir que temos a solução mais próxima da original, *i.e.* resolve-se o Problema das Perturbações Mínimas (MPP). Contudo, estes métodos requerem tempos de execução superiores para encontrar a nova solução, já que estamos a reolver um problema novo (com mais um critério de optimização). Isto pode ser mitigado usando parte da árvore de procura anterior. Além disso, a maior parte das abordagens usam funções de custo genéricas (*e.g.* distância de Hamming) que por isso podem não retratar a realidade.

Neste trabalho propomos novos algoritmos para resolver o MPP para dois problemas de escalonamento: escalonamento de horários universitários e escalonamento de comboios. No caso universitário, os algoritmos implementados foram testados com dados provenientes do Instituto Superior Técnico e da Competição Internacional de Horários de 2019. Um dos algoritmos propostos ficou no top 5 da competição. No caso dos comboios, os algoritmos foram testados com dados da Swiss Federal Railways e da PESPLib. A avaliação mostra que os algoritmos propostos são mais eficientes do que os descritos na literatura.

Resumindo, os algoritmos propostos nesta tese correspondem a um avanço significativo nas abordagens para resolver problemas de escalonamento, em particular quando sujeito a disrupções.

Palavras Chave

Problema das Perturbações Mínimas; Horários Universitários; Escalonamento de comboios; Satisfação; Programação Linear Inteira.

Abstract

Scheduling problems are common in many applications that range from factories and transports to universities. Most times, these problems are optimization problems for which we want to find the best way to manage scarce resources. Reality is dynamic, and thus unexpected disruptions can make the original solution invalid. There are many methods to deal with disruptions well described in the literature. These methods can be divided into two main approaches: (i) create robust solutions for the most common disruptions, and (ii) solve the problem again from scratch extended with new constraints.

The goal of creating robust solutions is to ensure their validity even after the most common disruptions occur. For this reason, it requires a detailed study of the most likely disruptive scenarios. The main disadvantage of creating a robust solution is a possible reduction in the overall quality (e.g., financial cost, customer satisfaction) to support the most likely disruptive scenarios that may never occur. Regardless of the robustness of the solution, we may need to solve the problem again.

Most of the methods developed to recover solutions after disruptions occur consist of re-solving the problem from scratch with an additional cost function. This cost function ensures that the new solution is close to the original. In other words, the methods solve the Minimal Perturbation Problem (MPP). However, all these methods require more execution time than the original problem to find a new solution. This can be explained by the fact that we solve a different problem (with more optimization criteria). One can mitigate this problem by re-using the search. Moreover, they use generic cost functions (*e.g.*, Hamming distance) that may have little significance in practice.

In this work, we propose novel algorithms to solve the MPP applied to two domains: university course timetabling and train scheduling. We tested our algorithms to solve university timetabling problems with data sets obtained from Instituto Superior Técnico and the 2019 International Timetabling Competition. One of these algorithms was ranked in the top 5 of the competition. When considering the train scheduling case study, we tested our algorithms with data from the Swiss Federal Railways and from PESPLib. The evaluation shows that the new algorithms are more efficient than the ones described in the literature.

Summing up, the proposed algorithms show a significant improvement on the state-of-the-art to re-solve scheduling problems under disruptions.

Keywords

Minimal Perturbation Problem; University course Timetabling; Disruption; Train Scheduling Problem; Satisfiability; Integer Linear Programming.

Acknowledgments

First of all, I would like to thank my supervisors, Professor Inês Lynce and Professor Pedro Monteiro, for all their guidance, support, and for always being there to help me solve my problems. In particular, during the low points of the international timetabling competition. Without them, I would not have been able to achieve all this. In the past years, I have learned so much about new topics, explored new continents, and it has been a real honor to be your Ph.D. student. I am looking forward to collaborating with you in the future.

I want to thank the members of comissão de acompanhamento de tese: Professor Inês Marques and Dr. Ricardo Saldanha for all the comments and suggestions about my thesis proposal.

I want to praise Professor Francisco Melo e Professor Helena Galhardas for their great courses and fruitful collaborations. Moreover, I would like to acknowledge the members of the SAT group for their support and interesting discussions. I want to thank, in particular, Filipe Gouveia that went through the Ph.D. program at the same time as me, and thus shared all my problems.

I want to thank Sofia Teixeira and Ana Sofia Correia for organizing the Ph.D. support meetings. Furthermore, I would like to thank Sofia Teixeira for all her help during the course of Advanced Topics on Artificial Intelligence.

I want to appreciate Professors David Matos, Maria Cravo, Mikolas Janota, Pedro Monteiro, and Inês Lynce for all their support in my teaching assistant experience.

I want to acknowledge all the help given to me from Serviços de Informática (DSI) in particular from Professor Luís Guerra e Silva, Sérgio Silva, and Luís Cruz. Moreover, I would like to thank Suzana Visenjou from Gabinete de Organização Pedagógica e Audiovisuais and Laurinda Dias from Área de Gestão de Recursos Humanos e Académicos for their time and valuable feedback.

Thanks very much, João Carraquico, Daniela Azul, and João Luís for all your support during these last years.

I want to thank my family and all my friends for supporting me on this journey.

Finally, I would like to thank Departamento de Engenharia Informática, Instituto Superior Técnico, Universidade de Lisboa, and Fundação para a Ciência e a Tecnologia (FCT) for partially supporting this work through the Ph.D grants (SFRH/BD/143212/2019).

Contents

List of Figures

List of Tables

Acronyms

ASP Answer Set Programming. 20, 23, 33

CNF Conjunctive Normal Form. 12–14, 21, 85, 89, 97, 134

COP Constraint Optimization Problem. 2, 10, 11, 13, 14

CSP Constraint Satisfaction Problem. 6, 10, 12, 13

GRASP Greedy Randomized Adaptive Search Procedure. xvi, 52, 61, 62, 66

HD Hamming distance. 24–30, 36, 37, 43, 53, 54, 70, 71, 88

ILP Integer Linear Programming. xv, 4, 6, 10–12, 23, 48, 57, 60, 61, 63–68, 73, 93, 127, 128

IST Instituto Superior Técnico. 4, 6, 33, 40, 41, 44, 57, 58, 61, 66, 73, 76, 93, 137

ITC International Timetabling Competition. 5, 6, 21, 22, 33, 57, 58, 76, 93, 106, 137

KUL Katholieke Universiteit Leuven. 20, 58

MaxSAT Maximum Satisfiability. xiv, 4, 6, 10, 13, 20, 22, 78, 81, 89, 92, 93, 97, 103, 115, 118, 119, 124, 127–129, 131, 133

MPP Minimal Perturbation Problem. 3, 10, 15, 24, 33, 44, 53, 76, 136–138

PDI Pentaho Data Integration. 57

PESP Periodic Event Scheduling Problems. 110, 112, 113, 127, 128, 133

RCL Restricted Candidate List. 52, 57

RCOM Compact room's timetable. 44, 47

SA Simulated Annealing. 22

SAT Boolean Satisfiability. 10, 12, 13, 20, 21

SBB Swiss Federal Railway. 5, 6, 110, 123, 126, 133, 134, 137

SCOM Compact student's timetable. 44, 53, 54, 69–71

TSOP Train Scheduling Optimization Problems. 4, 5, 14, 110, 111, 113–116, 128, 133

UCTTP University Course Timetabling Problem. 4, 14, 20, 40

WCNF Weighted Conjunctive Normal Form. 14, 93, 94, 117

WHD Weighted Hamming distance. 43, 53, 54, 70, 71

1

Introduction

Contents

Creating schedules is a decision problem [5] where the goal is to assign events to scarce resources and time slots, subject to a set of constraints and optimization objectives. Notwithstanding humans have been solving scheduling problems for innumerable years, there is little proof of formal processes for solving them until the middle of the eighteenth century [6]. Nowadays, these formal processes are used regularly in many industries. The goal is to create more efficient and profitable schedules. The resources, events, constraints, and objectives vary from organization to organization. Scheduling problems are common in many applications that range from factories [7] and transports [8] to universities [9].

Example 1 (Timetable)**.** Let us consider a simple schedule with one time slot available on each day of the week (except Sunday). The university has only two rooms with the same capacity r_1 and r_2 and four courses A, B, C and D. Each course corresponds to a single class with only one time slot of contact per week. Furthermore, all classes must be taught every week of the semester. For this reason, we can make a schedule for a single week (and all other weeks have the same schedule). Moreover, the classes have a set of constraints between them. The classes of the courses of A, B, and C have the same students, and thus they cannot overlap in time. The courses C and D share the teacher and thus cannot overlap in time. A possible solution is shown in Table 1.1. The solution requires allocating the classes of the courses A and D on Monday, B on Tuesday, and C on Wednesday. The classes of the courses A to C are taught in room r_1 and the class of course D in room r_2.

Most times, these problems are optimization problems where we want to find the best way to manage scarce resources. Therefore, Constraint Optimization Problem (COP) are particularly useful when modeling and solving this type of real-life problems. COPs have successfully been applied to problems that range from scheduling problems [9–11] to biological networks [12–14].

Example 2. Let us consider Example 1 again. Let us consider a new bi-objective goal: (i) minimize the number of rooms used[1], and (ii) ensure that there is a two days gap between the classes of course C and D. Both criteria are equally important. A possible optimal solution is shown in Table 1.2. The solution consists of teaching the class of the course D on Monday, A on Tuesday, B on Wednesday, and C on Thursday. Therefore, we can allocate all classes in the same room (r_1), and so r_2 is completely available for other aperiodic events. Also, the courses' classes C and D have a two-day gap between them.

[1] We want to have a room available for other aperiodic events.

Table 1.1: A simple schedule with one time slot available in each day of the week (except Sunday). The university has only two rooms with the same capacity r_1 and r_2 and four courses of a single class A, B, C and D. The courses A, B and C cannot overlap in time. The courses C and D cannot overlap in time.

Room	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
r_1	A	B	C			
r_2	D					

Table 1.2: A simple schedule with one time slot available in each day of the week (except Sunday). The university has only two rooms with the same capacity r_1 and r_2 and four courses of a single class A, B, C and D. The courses A, B and C cannot overlap in time. The courses C and D cannot overlap in time. The solution was designed with a bi-objective goal: (i) minimize the number of rooms used; and (ii) ensure that there is a two days gap between the classes of courses C and D.

Room	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
r_1	D	A	B	C		
r_2						

1.1 Why Solving Scheduling Problems Under Disruptions

When solving scheduling problems, unexpected disruptions may cause the problem to change, and thus the original solution may no longer be valid. Solving a new problem is now required, but solving the problem from scratch is unnecessarily expensive. Furthermore, solving the problem from scratch may produce a completely different solution, which in many cases is an undesirable nuisance for all actors involved.

There are two main approaches to tackle this scenario general: (i) solving the problem again from scratch with new constraints; (ii) create robust solutions for the most common disruptions. When considering specific domains (like train scheduling problems), many approaches use local search to explore neighbor solutions. The goal of these approaches is to mitigate the difficulties of solving from scratch.

The first approach focuses on changing the solution after the disruption occurs in order to generate a new feasible solution. When considering this approach, in many real-life instances, it is important to find a solution as similar as possible to the original one. This ensures that the new solution causes the smallest impact on the already implemented solution, thus solving the so-called Minimal Perturbation Problem (MPP). There are different methods proposed in the literature to solve MPP. The methods entail minimizing a distance metric between the new solution and the original one. However, these methods imply solving the problem from scratch each time a disruption occurs.

The second approach involves predicting the disruptions that may arise to create a robust solution. Therefore, a robust solution may be sub-optimal (*e.g.*, in terms of cost and customer satisfaction), but it is still valid if the predicted disruptions occur. For this reason, the solution may have less quality to support disruptions that may never occur but not be able to cope with unexpected disruptions that may actually occur. Regardless of the robustness of the solution, we may need to solve the problem again (re-solve). Therefore, the focus of this thesis is on re-solving the problem after disruptions occur.

Example 3. Recall Example 2. Additionally, consider that room r_1 has to be closed for maintenance (*e.g.* due to COVID-19 contamination) on Tuesday in a specific week. The solution must change - in particular, the classes of course A must be taught in a different room. Solving, once again, the problem from scratch could cause all classes to be taught in the room r_2. This solution satisfies all the constraints but causes unnecessary confusion to the teacher and students of the courses B, C,

and D. This type of disruptions are unpredictable in nature, and thus hard to mitigate with robust solutions.

The goal of the proposed algorithms is to reduce the resources required when solving the MPP. For this reason, the algorithms proposed in this thesis either only re-solve a small part of the problem (where the disruption occurs) or continue the search performed during the original search process. These algorithms are able to reduce the search space when solving the new problem and thus require fewer resources than a straightforward approach to find a new solution.

1.2 Original Contributions

The target of this work is to develop new efficient encodings and algorithms for different scheduling problems under disruptive scenarios. We evaluate our proposed solution with real-world benchmarks from University Course Timetabling Problem (UCTTP) and Train Scheduling Optimization Problems (TSOP). We test our approaches with different solvers of Integer Linear Programming (ILP) and Maximum Satisfiability (MaxSAT).

We start by focusing on the university course timetabling before any disruptions occur. The original objective was to solve the problem at Instituto Superior Técnico (IST). At IST the timetables are still generated by hand. After a survey of the constraints and optimization objective considered, we define the following goals: (i) improve the handmade solution in terms of room usage and (ii) solve the minimal perturbation problem.

In order to improve the handmade solution, we want to optimize the room occupation by determining the events allocated to each room while ensuring that the rooms have enough capacity to *seat* all people participating in those events. With this purpose, we propose three different algorithms [15]:

- A greedy randomized adaptive search procedure that is efficient but does not provide any type of quality assurance.
- A greedy algorithm that takes advantage of the nature of the monotone, positive, sub-modular cost function to ensure the solution is within 63% of the optimal.
- An ILP approach that uses decomposition techniques to find the optimal solution.

All these approaches are able to provide significant gains when compared with the handmade solution. The results of this work were published in the journal Operations Research Perspectives [15].

In addition, we propose and analyze two different integer programming models to encode the minimal perturbation problem. To validate the proposed models, disruptions are randomly generated based on the probability distributions obtained from the history of timetables over the last five years in IST. Overall, our models, combined with an incremental approach, are shown to be able to efficiently solve all problem instances. This work was published in the Journal of Scheduling [16].

Moreover, we apply the straightforward extension of these models to the data sets of International Timetabling Competition (ITC) 2019. However, the integer programming models only solve 1/3 of the benchmark due to memory constraints. In this context, we develop the *UniCorT* MaxSAT based tool for university course timetabling problems. *UniCorT* has strong pre-processing techniques to reduce the search space. This tool was ranked in the top 5 of the competition[2]. These results were published in the proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling [17].

We also propose an iterative version of *UniCorT*, which increases the domain of the problem (the number of time slots available to assign a class) in each iteration. This approach allows solving all instances of the competition and improving the results that lead to the top 5. Furthermore, we extend *UniCorT* with the ability to solve the minimal perturbation problem. This work was published in the proceedings of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research [18].

Finally, we propose an iterative learning algorithm that uses a MaxSAT solver to solve Train Scheduling Optimization Problems (TSOP). The proposed algorithm was tested with data sets from Swiss Federal Railway (SBB), and it is, on average, twice as fast as the best existing approach. The iterative learning algorithm increases the search space by relaxing the arrival time of the train (delay) when needed. This way, we can start solving the problem with a small arrival time domain and increase the domain on-demand. The iterative nature of our novel approach reduces the impact of time discretization on the encoding. We also analyze real schedule data from Switzerland and the Netherlands to create a disruption generator based on probability distributions. We use these disruptions to create an algorithm to solve the minimal perturbation problem. In order to validate our algorithm with PESPLib benchmark, we propose a novel encoding. This encoding achieves a significant improvement over existing SAT based solutions for solving the PESPLib instances.

1.3 Thesis Outline

The central subject of this thesis is the study of scheduling problems under disruptive scenarios. The work in this thesis can be divided into two stages: solving scheduling problems without disruptions and re-solving scheduling problems after disruptions occur. In this work, we consider two scheduling problems: university course timetabling and train scheduling problems. Even though both case studies have their own characteristics and constraints, they can be solved by similar high-level algorithms. Furthermore, the insights learned in the process of designing a solution to tackle one problem can help define better approaches to solve the other problem. For this reason, the first step was to develop state-of-the-art methods to solve scheduling problems without disruptions. The second step requires learning the most common disruptive scenarios in the different scheduling problems considered in this work. After creating the disruptive scenarios, we were able to define domain dependent metrics to evaluate the quality of the new solution for the re-solving algorithm.

[2]https://www.itc2019.org/home

The work developed in the thesis was motivated by three different cases studies: (i) the university course timetabling problem at IST; (ii) the university course timetabling problem from ITC; and (iii) train scheduling at SBB. Even though the first two case studies require solving the university course timetabling problem, their characteristics and requirements are significantly different. Therefore, different approaches must be considered. For this reason, the dissertation core is composed of three main parts. Therefore, the dissertation is organized as follows:

- **Chapter 2: Preliminaries**: This chapter provides the main concepts that will be used throughout the dissertation. It starts by introducing the notions of Constraint Satisfaction Problem (CSP), MaxSAT and ILP. Finally, we formally define scheduling problems and the minimal perturbation problem. These concepts are the basis for the related work chapter.

- **Chapter 3: Related Work**: This chapter describes the relevant related work on the topic of university timetabling, train scheduling, and minimal perturbation problems in general. The state-of-the-art algorithms shown in this chapter are going to be used later on to evaluate the contributions of the thesis.

- **Chapter 4: Instituto Superior Técnico Course Timetabling**: This is the first case study of this dissertation and the first chapter describing our contributions. This chapter presents various ILP encodings and greedy algorithms to solve the course timetabling problem of IST. The case study is divided into four parts: (i) profiling and cleaning the IST data set; (ii) automatically optimizing the room usage of the handmade timetables; (iii) solving the whole course timetabling from scratch; and (iv) re-solving course timetabling.

- **Chapter 5: International Timetabling Competition 2019**: This chapter describes the approach proposed to solve the ITC that was ranked among the five finalists of the ITC 2019 competition. This chapter builds on the case study's successful results at IST and generalizes it into *UniCorT*. *UniCorT* is a tool to solve and re-solve university course timetabling and student sectioning. The algorithms to section students into classes and different pre-processing techniques used in *UniCorT* are discussed. This chapter also compares *UniCorT* with all other approaches that finished in the top 5 of the competition.

- **Chapter 6: Train Scheduling**: This is the last case study. In this chapter, we propose three novel iterative algorithms to solve and re-solve the train scheduling problems. The goal of these algorithms is to avoid the problems associated with the discretization of time in seconds. This chapter is motivated by the Swiss Federal Railway (SBB) Crowd Sourcing Challenge. The proposed solution is compared with the other methods that participated in this challenge. The Open-data initiative SBB allows creating realistic disruptive scenarios to evaluate our algorithms. The algorithms proposed in this chapter are able to solve the PESPLib benchmark. However, the PESPLib benchmark needs to be converted before usage. This procedure allows comparing our solution with other approaches described in the literature.

- **Chapter 7: Conclusions and Future Work**: This is the final chapter of the dissertation. The goal of this chapter is to present the conclusions and to discuss possible future research directions.

2

Preliminaries

Contents

This chapter introduces the notations that are used throughout this thesis. We start by introducing the notions of Constraint Satisfaction Problem (CSP) (Section 2.1), Integer Linear Programming (ILP) (Section 2.2), Boolean Satisfiability (SAT), and Maximum Satisfiability (MaxSAT) (Section 2.3). Finally, we formally define scheduling problems (Section 2.4) in general and the Minimal Perturbation Problem (MPP) (Section 2.5).

2.1 Constraint Satisfaction

Definition 1 (CSP)**.** CSP [19] is defined as a triple $\Theta = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, where:

- $\mathcal{V} = \{V_1, .., V_n\}$ is a finite set of n variables;
- $\mathcal{D} = \{\mathcal{D}_{V_1}, .., \mathcal{D}_{V_n}\}$ is a finite set of n domains where each domain corresponds to the values a variable can take;
- $\mathcal{C} = \{C_1, .., C_m\}$ is a finite set of constraints that restricts the values the variables can take.

The solution (s) to a CSP is a complete assignment to the variables ($\mathcal{V}$) satisfying all the constraints in $\mathcal{C}$. A complete assignment to a CSP is an assignment to all the variables of the problem. An assignment is a partial mapping $m : \mathcal{V} \rightarrow \mathcal{D}$ which assigns for each $V \in \mathcal{V}$ a value for its domain $\mathcal{D}_{V_i}$. □

Example 4 (CSP)**.** Consider, for example, the following CSP:

- $\mathcal{V} = \{A, B, C, D\}$,
- $\mathcal{D}_A = \mathcal{D}_B = \mathcal{D}_C = \mathcal{D}_D = \{1, 2, 3, 4, 5, 6\}$,
- $C = \{\texttt{alldifferent}(A, B, C),\ C \neq D\}$.

The constraint `alldifferent` requires all the variables involved to have different values. A possible solution to this problem is: $\{(A, 1), (B, 2), (C, 3), (D, 4)\}$. This corresponds to the CSP encoding of Example 1.

A CSP can be generalized into an optimization problem with the addition of a cost function and it is defined as follows:

Definition 2 (COP)**.** A Constraint Optimization Problem (COP) can be defined as a 4-tuple $\Theta = (\mathcal{V}, \mathcal{D}, \mathcal{C}, f)$, where f is a cost function.

A solution to a COP is an assignment to all variables in such a way that all the constraints ($\mathcal{C}$) are satisfied and the value of the cost function is minimized. □

Example 5 (COP)**.** Consider, for example, a COP resulting from the extension of Example 4 with the following cost function: $f_c = A + B + C + D$.

The solution shown in Example 4 is not the optimal solution to this problem since intuitively D can be assigned the value 1. The solution that minimizes the cost function is $\{(A,1),(B,2),(C,3),(D,1)\}$.

The problem described in Definition 2 can be further generalized by considering a set of cost functions transforming the problem into a multi-objective optimization problem [20, 21].

There are different approaches to deal with multi-objective optimization problems. These approaches range from solving the problem with lexicographic ordering (*i.e.* ordering the optimization objectives based on their absolute importance) to computing a Pareto optimal solution.

Considering a lexicographic ordering, a solution will only be optimized using a less important objective if the solution is already optimal in terms of all other objectives.

When considering multi-objective optimization problems, an optimal solution in terms of a specific objective may not be optimal when considering another objective. This problem is exemplified later on (in Example 10). When considering multi-objective, one possible approach is to find a solution that is called Pareto optimal. A solution is Pareto optimal if and only if it is impossible to improve the solution in any objective without worsening the solution in a different objective. This concept can be defined as follows:

Definition 3 (Pareto Optimal Solution)**.** Consider O as a set of cost functions (objective) for which we want to optimize our solution. A solution s_0 is Pareto optimal if and only if there is no other feasible solution, s_1, such that $\exists_{o\in O} o(s_1) < o(s_0)$ and $\forall_{o\in O} o(s_1) \leq o(s_0)$. The set of all Pareto optimal solutions is called Pareto frontier. □

2.2 Integer Linear Programming

In this work, we propose different ILP models to solve university course timetabling. ILP is a mathematical optimization program [22, 23] for which: (i) all the variables ($\mathcal{V}$) have integers or Boolean domains ($\mathcal{D}$); and (ii) the constraints ($\mathcal{C}$) and the cost function are linear. For this reason, ILP can be seen as a specific case of COP [24].

Definition 4 (ILP)**.** Consider n real numbers $c_1,\ldots,c_n$; m real numbers $b_1,\ldots,b_m$; $m\times n$ real numbers $a_{1,1},\ldots,a_{n,m}$; and n integer variables $x_1,\ldots,x_n \in \mathcal{V}$. An integer linear program, in standard form, is formalized as follows:

$$\text{maximize:} \sum_{j=1}^{n} c_j x_j \tag{2.1}$$

subject to:

$$\sum_{j=1}^{n} a_{j,i} x_j \leq b_i \qquad \forall_{i=1}^{m} \tag{2.2}$$

$$x_j \geq 0 \qquad \forall_{j=1}^{n} \tag{2.3}$$

$$x_j \in \mathbb{Z} \tag{2.4}$$

Line (2.1) is the cost function, and lines (2.2) and (2.3) are different types of constraints. □

Karp [25] proved that the 0–1 integer linear programming variant of ILP is NP-complete. In 0–1 integer linear programming, all variables are binary, and there is no cost function (*e.g.*, a satisfaction problem). Later, Integer Linear Programming (ILP) has been proved to be NP-hard as well [26].

Example 6. Consider Example 4 with the following constraints:
$\varphi_h = \{\ C \neq D, \texttt{alldifferent}(A, B, C)\}$, $\varphi_s = \{(A = 1), (C = 1)\}$ and a cost function penalizing the number of unsatisfied φ_s. ILP does not support a *not equals* constraint (*e.g.*, $z \neq y$) natively. However, there is a workaround for discrete integer variables [27]. We introduce two new binary auxiliary variables ($b^1_{v_1,v_2}, b^2_{v_1,v_2}$) as indicators for $v_1 \leq v_2 - 1$ and $v_1 \geq v_2 + 1$, respectively. Finally, we only need to guarantee that $b^1_{v_1,v_2} + b^2_{v_1,v_2} = 1$. Therefore, we create two Boolean variables for every two variables in the *not equals* constraint. Let us consider $(v_1, v_2) \in \Omega$ as the set of two variables that must have different assignments. In this case, $|\Omega| = 4$. The optimization criteria, in this case, aim to minimize the value of A and C. As we are writing the statement as maximization, we consider the inverse (6 minus the value).

$$\text{maximize: } A - C \tag{2.5}$$

subject to:

$$(v_1 - v_2 - 1) \times b^1_{v_1,v_2} \geq 0 \qquad \forall_{(v_1,v_2)\in\Omega} \tag{2.6}$$

$$(v_1 - v_2 + 1) \times b^2_{v_1,v_2} \leq 0 \qquad \forall_{(v_1,v_2)\in\Omega} \tag{2.7}$$

$$\sum_{i=1}^{2} b^i_{v_1,v_2} = 1 \qquad \forall_{(v_1,v_2)\in\Omega} \tag{2.8}$$

$$A, B, C, D \in \{1, \ldots, 6\} \tag{2.9}$$

$$b^1_{v_1,v_2}, b^2_{v_1,v_2} \in \{1, 0\} \qquad \forall_{(v_1,v_2)\in\Omega} \tag{2.10}$$

Please note that the constraints are not linear and thus only integer programming.

2.3 Maximum Satisfiability

The propositional satisfiability (SAT) problem can be seen as a CSP where the domain of the variables is restricted to Boolean. In this thesis, we propose different SAT encodings for university course timetabling and train scheduling.

Definition 5 (SAT)**.** The propositional satisfiability (SAT) problem consists of deciding whether there is a truth assignment to the Boolean variables ($\mathcal{V}$) such that a given CNF formula is satisfied. A propositional formula in conjunctive normal form (CNF) is defined as a conjunction of clauses, where a clause is a disjunction of literals and a literal is either a Boolean variable $x \in \mathcal{V}$ or its complement $\neg x$. A formula is satisfied iff there is at least one assignment where all the clauses are satisfied. A clause is satisfied iff there is at least one literal satisfied. □

Nowadays, most SAT solvers apply conflict-driven clause learning algorithms [28, 29], which are based on the well-known Davis-Putnam algorithm [30] (see [31] for more details). However, we want more than just checking the satisfiability of the problem. For this reason, we make use of MaxSAT. MaxSAT is COP where the domain of the variables is restricted to Boolean. The simplest definition of MaxSAT is as follows.

Definition 6 (MaxSAT)**.** The MaxSAT problem is an optimization version of SAT, where the *objective* is to find an assignment that maximizes the number of satisfied clauses. □

Furthermore, we may need to slit the clauses into hard and soft. Therefore, we need to make use of partial MaxSAT.

Definition 7 (Partial MaxSAT)**.** A partial MaxSAT formula ($\varphi = \varphi_h \cup \varphi_s$) consists of a set of hard clauses (φ_h) and a set of soft clauses (φ_s). The *objective* in partial MaxSAT is to find an assignment such that all hard clauses in φ_h are satisfied, while maximizing the number of satisfied soft clauses in φ_s. □

Example 7. Recall Examples 4 (CSP) and 6 (integer programming). Let us consider the following Boolean variables A_i to D_i with $i \in \{1, \ldots, 6\}$. Each variable corresponds to the assignment of a class to a time slot i. When solving the same timetabling problem with SAT, the Conjunctive Normal Form (CNF) encoding for the hard clauses is the following:

$$\forall_{i \in \{1,\ldots,6\}} \neg A_i \vee \neg B_i \tag{2.11}$$

$$\forall_{i \in \{1,\ldots,6\}} \neg A_i \vee \neg C_i \tag{2.12}$$

$$\forall_{i \in \{1,\ldots,6\}} \neg B_i \vee \neg C_i \tag{2.13}$$

$$\forall_{i \in \{1,\ldots,6\}} \neg C_i \vee \neg D_i \tag{2.14}$$

The CNF encoding for the soft clauses is the a set with two unit clauses $\varphi_s = \{A_1, C_1\}$.

In addition, we may want to add different weights to the clauses to create a more sophisticated optimization function (*e.g.* lexicographic optimization). For this reason, in this thesis, we also consider the weighted variant of partial MaxSAT, which is defined next.

Definition 8 (Weighted Partial MaxSAT)**.** A weighted variant of partial MaxSAT has a function $w^{\varphi} : \varphi_s \to \mathbb{N}$ associating an integer weight to each soft clause. In this case, the *objective* is to satisfy all the clauses in φ_h and maximize the total weight of the satisfied clauses in φ_s. □

Most MaxSAT solvers [32, 33] are implemented calling a SAT solver iteratively to improve the quality of the solution. There are different algorithms to guide the search. In this work, we use the linear search with clusters algorithm [34]. This algorithm uses a linear search. In order to improve

performance, the algorithm uses heuristics to cluster the clauses of the objective function by weight. The weight is used to assign a level of importance while solving. This can be seen as lexicographic optimization of the terms of the objective function.

In general, we assume that all formulas are encoded into Weighted Conjunctive Normal Form (WCNF). Nevertheless, in this thesis we will write some constraints in pseudo-Boolean (PB) form for the sake of readability. PB constraints are nothing more than linear constraints over Boolean variables, and can be written as follows: $\sum q_i x_i \ \text{OP} \ K$, where K and all q_i are integer constants, all x_i are Boolean variables, and $\text{OP} \in \{<, \leq, =, \geq, >\}$. PB constraints can be easily translated into CNF [35]. In this work, we tested different CNF encodings for PB constraints.

The weighted MaxSAT problem can be converted to an ILP problem [36] and vice-versa. Lately, there has been work on merging the best of both worlds [37]: a SAT solver and an ILP [38] solver. MaxHS [39] one of the competitors of the 2020 MaxSAT evaluation and uses both a SAT solver and CPLEX (ILP solver). The best tool to solve PESP [40] is also a hybrid between SAT and ILP (discussed in the next chapter).

The idea behind this is to exploit the advantage of dealing with linear constraints in ILP solvers and the benefit of clause learning and unit propagation of the SAT solvers. The ILP solvers have more difficulty with implications and short clauses than SAT solvers. On the other hand, SAT solvers have more difficulty with pseudo-Boolean constraints. For example, in MaxHS, neither the SAT solver nor CPLEX has enough information to solve the entire problem. In this case, an overview of the process is as follows. The SAT solver is used to solving the decision version of the problem. CPLEX receives the solution from the SAT solver and optimizes it. This is the same method used by Borndörfer *et al.* [40]

2.4 Scheduling Problems

There are many scheduling problems in the world [5] that range from transportation (*e.g.* trains [8], airlines [41]) and university timetabling [42] to planning [43]. One can informally define a scheduling problem in general as the mapping of a set of events to a set of resources and time slots, subject to a set of constraints and optimization objectives. Therefore, one can see the scheduling problem as a COP. In this work, we consider two different scheduling problems: University Course Timetabling Problem (UCTTP) and Train Scheduling Optimization Problems (TSOP). The goal of the university course timetabling problem is to assign all classes to rooms and time slots subject to a set of constraints (see Example 1). The goal of train scheduling is to assign all trains to routes and time slots subject to a set of constraints (see the example below).

Example 8. Figure 2.1 shows a simplified railways. There are two train routes from *B* to *C* and therefore the railway supports two trains at the same time. *B* train station can receive two trains (from *A* and *F*). *B* and *C* are connection spots since the trains can depart to multiple routes. Consider that one train has to go from *A* to *C*. A possible solution to this routing problem is shown in red. Now, let us consider that the traveling time from v_1 to v_3 is 9 minutes, and from v_3 to v_4 / v_5 is 27 minutes.

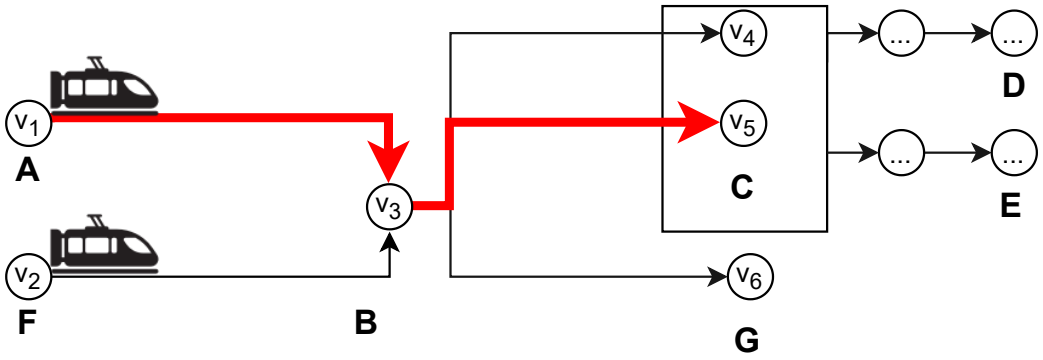


Figure 2.1: Part of the railway network where one train has to go from *A* to *C*. The bold red arrow represents a possible solution. Nodes and edges represent stations and sections of railway track, respectively. The square represents a junction point.

Assume the train cannot leave *A* before 9AM. Also, it has a connection with another train at *C*. The connection requires the train to arrive before 9:42AM to ensure the passengers can interchange with the train moving to *E*. Therefore, a feasible solution to this scheduling problem instance is the following timetable:

- The train departs from *A* at 9AM.
- To make the connection at *C*, the train stops in *B* for only 5 minutes.
- The train leaves at 9:14AM from *B* station, thus arriving at the destination at 9:41AM.

2.5 Minimal Perturbation Problem

Re-solving an optimization problem, considering the particular case of the Minimal Perturbation Problem (MPP), has many applications from timetabling [44–47] and train scheduling [48–50] to biological networks [12–14].

The Minimal Perturbation Problem (MPP) is the task of finding the closest new feasible solution to the problem based on a previously found solution that is no longer valid. There are two possible reasons for this to happen: (i) new constraints were added or (ii) the variables of the problem have changed. The MPP can be described as a COP since it has a cost function (the evaluation between the old and new solutions).

Definition 9 (MPP)**.** Consider the CSP $\Theta = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, with s_0 as a feasible solution. Consider another CSP Θ_N that was created based on Θ such that $\mathcal{V} \cup \mathcal{V}_\mathcal{N} \neq \emptyset \wedge \mathcal{D} \cup \mathcal{D}_\mathcal{N} \neq \emptyset \wedge \mathcal{C} \cup \mathcal{C}_\mathcal{N} \neq \emptyset$. These changes may cause the solution s_0 to be incomplete or even no longer feasible.

MPP is a COP with the cost function $F = \delta(s_0, s_1)$ where s_1 is the new solution, subject to the new sets of constraints and/or variables. The distance δ is a function that evaluates the differences between the two solutions. □

Note that adding new variables causes the solution to be incomplete. Adding new constraints or changing the variable's domain may cause to solution to be no longer feasible.

The function δ can be domain independent, applied to all problems (*e.g.* Hamming distance or HD for short), or specific to the problem's domain. Observe that there are simplified definitions of MPP in the literature. The formal definition of the different distance metrics is introduced in the next chapter. A simplified approach considers that $\mathcal{C} = \mathcal{C}_P \cup \mathcal{C}_N$, where $\mathcal{C}_P \cap \mathcal{C}_N = \emptyset$. In other words, the constraints cannot change, and only new constraints can be added.

Example 9 (MPP)**.** Consider, for example, the following CSP with s_0 as a valid solution:

- $\mathcal{V} = \{A, B, C, D\}$,
- $\mathcal{D}_A = \mathcal{D}_B = \mathcal{D}_C = \mathcal{D}_D = \{1, 2, 3, 4, 5, 6\}$,
- $\mathcal{C}_p = \{\texttt{alldifferent}(A, B, C),\ C \neq D\}$,
- $s_0 = \{(A, 1), (B, 2), (C, 3), (D, 4)\}$.

The new constraint $\mathcal{C}_N = \{D = B\}$ causes the solution s_0 to be no longer feasible, thus requiring the solution to change. A distance function δ measures the difference between the two solutions. In this example, two possible functions can be considered: the number of variables in which the value changed (Hamming distance) or the sum of differences between the values of the variables of the two solutions (Manhattan distance). Note that the HD can be applied to integer domains by only measuring the number of variables with different values. Considering δ as the Hamming distance, the optimal solution to the problem is $\{(A, 1), (B, 2), (C, 3), (D, 2)\}$.

This is the basic version of the MPP since different levels of importance can be added to the constraints. Thus, it is possible to extend Definition 9 to consider different types of constraints (hard and soft). Let us consider the CSP P, subject to the hard constraints $\mathcal{CH}_P$ and soft constraints $\mathcal{CS}_P$, which has s_0 as solution. s_0 is the solution that minimizes the number of unsatisfied soft constraints. Now, the need for re-solving the problem can arise from the set of hard constraints $\mathcal{CH}_N$, the set of soft constraints $\mathcal{CS}_N$ or both.

Definition 10 (MPP cont.)**.** Consider the MPP Definition 9. Furthermore, consider that the set of constraints now can be hard ($\mathcal{CH}$) or soft ($\mathcal{CS}$). Therefore, the goal of the MPP is now a multi-criteria problem. The cost function f has to take into account the distance, δ, between a new solution and the original one, and the number of unsatisfied soft constraints $\mathcal{CS}_N$. □

Example 10 (MPP cont.)**.** Consider, for example, the following CSP with s_0 as a valid solution:

- $\mathcal{V} = \{A, B, C, D\}$,
- $\mathcal{D}_A = \mathcal{D}_B = \mathcal{D}_C = \mathcal{D}_D = \{1, 2, 3, 4, 5, 6\}$,

- $\mathcal{CH}_p = \{\texttt{alldifferent}(A, B, C),\ C \neq D\}$
- $\mathcal{CS}_p = \{(A = 1), (C = 1)\}$,
- $s_0 = \{(A, 3), (B, 2), (C, 1), (D, 4)\}$.

The new constraint $\mathcal{CS}_N = \{(D = 1)\}$ causes the solution s_0 to be no longer optimal, thus it is possible to improve the solution. Consider δ_H as the Hamming distance (optimization objective o_1). In terms of distance, the optimal solution to the problem is $s_h = \{(A, 3), (B, 2), (C, 1), (D, 4)\}$ since the distance is 0. However, in terms of the number of unsatisfied soft constraints, (o_2) $s_c = \{(A, 1), (B, 2), (C, 3), (D, 1)\}$ is a better solution (as it satisfies one more soft constraint).

The solution s_h is Pareto optimal since it is impossible to find a solution that improves the cost function o_2 without worsening δ_H. The same explanation is valid for solution s_c given that both solutions are in the Pareto frontier.

3

Related Work

Contents

This chapter describes the relevant state-of-the-art approaches to solve the different scheduling problems organized as follows. Section 3.1 describes the methods to solve university course timetabling problems. Section 3.2 describes the methods to solve train scheduling problems.

3.1 University Course Timetabling

University Course Timetabling Problem (UCTTP) is known to be NP-complete [51–53]. University timetabling can be classified into two major categories: examination timetabling [54–56] and course timetabling problems [57, 58]. These categories are characterized along these lines:

- Examination timetabling - focuses on creating timetables for the examinations. These timetables must ensure that a student can attend examinations for which he/she is enrolled in. Furthermore, the timetables must ensure an instructor can attend, and we can assign multiple exams to the same room[1].
- Course timetabling:
  - Curriculum-based course timetabling [59] - focuses on creating timetables based on a predefined curriculum that the students must follow.
  - Post-enrollments timetabling [60] - deals with creating timetables based on the students enrollments.

In this thesis, we consider the whole course timetabling as one. The organization of timetabling competitions in the past has led to important advances in solving University Course Timetabling Problem (UCTTP) [42, 61]. In the literature, there are several different approaches to solve UCTTP, namely: constraint programming [54, 62–66], Answer Set Programming (ASP) [67], Boolean Satisfiability (SAT) [68], MaxSAT [69, 70], integer programming [44, 47, 71–79], genetic algorithms [80–82], multi-agent [83, 84], reinforcement learning [85], and local search [1, 64, 86–92].

Burke *et al.* [79] proposed an ILP based method to solve curriculum-based timetabling. The optimization objectives used considers, among others, the room capacity and curriculum compactness. The objective is to minimize the global number of students not seated, and to reduce the number of time gaps between classes of the same curriculum. Therefore, the focus of the compaction process lies only on the student's timetable. The minimization does not ensure a uniform distribution of the unseated students. The ILP method decomposes the problem into multiple sub-problems where only a part of the optimization objectives are used. These sub-problems can be used to compute bounds in their respective optimization objectives. In the end, the solution to the problem is computed based on the solution of each of the sub-problems. This is an exact method. However, the decomposition may lead to a sub-optimal solution.

Vermuyten *et al.* [93] proposed a two-stage approach to optimize student flows using ILP. The approach was tested using the real data from Katholieke Universiteit Leuven (KUL). The first stage focuses on assigning classes to time slots and rooms, and the second focuses on re-assigning rooms

[1] A detailed description of the constraints used can be found at: https://www.unitime.org/exam_description.php.

to classes with pre-defined time slots from the first stage. The second stage's main optimization objective is to assign rooms in such a way that congestions are avoided. The ILP implementation also adds some constraints regarding the compaction of the timetables from a student's perspective. The ILP implementation tries to avoid timetables with two or more hours without classes for the students. This type of constraint is essential for commuting students. However, the proposed method does not reduce the number of isolated classes (*i.e.*, days where a student timetable has only one class assigned). This type of decomposition is common since it reduces the problem complexity without losing any solutions [94]. Another approach [56] follows the same decomposition for examination timetabling: it applies a greedy heuristic to the first task (exams assigned to timeslots) and ILP to the second (exams assigned to rooms).

The multi-agent [83, 84] approach focuses on negotiating the assignment of classes to rooms and time slots in order to reach a feasible timetable while optimizing the preferences of the teachers. The agents representing the teacher have different ranks and seek to assign their classes according to their preference. The higher ranking teachers have priority in terms of having their preferences satisfied.

Beyrouthy *et al.* [95] studied the utilization of teaching space in order to improve room-size profiles when planning to build a campus. The study shows that most rooms have overcapacity (the number of seats of the rooms is larger than the number of students). Furthermore, it was shown that the location of the room has a direct effect on room utilization since both students and teachers prefer certain locations. Beyrouthy *et al.* [96] proposed methods to split classes in order to improve the room utilization.

Lindahl *et al.* [97] studied the impact of the number of time slots allowed on the quality of the timetable. To this end, linear programming models were developed to solve the timetabling problem with three optimization objectives: the number of time slots available, room usage (minimize the number of rooms used), and overall quality. The study of the number of time slots available is particularly important since it is easier to increment the number of time slots when adding new courses than to build more rooms. This approach could be seen as the basis for the new ITC 2019 work of Holm *et al.* [71]. Holm *et al.* adapted the approach from Lindahl *et al.* [97] to consider the existence of weeks, days, and complex pseudo-Boolean constraints. However, the additional complexity of ITC 2019 would make it impractical to solved as is. More details are given below.

Song *et al.* [89] proposed an iterative algorithm with three stages: initialization, intensification, and diversification to solve the course timetabling problem. The first stage finds a partial feasible solution using a greedy algorithm to allocate the maximum possible number of events. The intensification stage uses a simulated annealing method to find a local optimum. The final stage uses random perturbations (swap of classes) to improve the solution. The solution found is used as the new starting point of the next iteration.

In the context of SAT, Asín Achá *et al.* [70] proposed a CNF encoding with four types of decision variables to solve curriculum-based course timetabling with data from the ITC 2007. The authors proposed variables to describe: the day of the class, the hours of the class, the room of the class,

and finally, the different times a curriculum is taught. The problem from the ITC 2007 differs from the problem defined in the ITC 2019 [58]. For example, in the ITC 2019, the classes can be scheduled in different weeks.

In recent years, a significant improvement in solving course timetabling problems has been achieved [98]. Behind this progress lies the public data sets from ITC [59], which are a simplified version of the real timetabling problem at the University of Udine. For this reason, the progress made still presents a gap between theory and practice [42] since it does not capture the full complexity of the real-world problem. A new and more complex challenge arose in the form of the ITC 2019 [58], in order to reduce this gap. The addition of the notions of different weeks, pseudo-Boolean constraints, and student sectioning problems make all previous approaches outdated and challenging to adapt. The adapted solution proved to have difficulties in solving all the instances of the competition without significant changes in the overall tool.

The participants proposed different approaches to solve the ITC 2019 problem. The solutions in the top 5 ranged from MaxSAT (described in section 5) and mixed-integer programming with heuristics [71, 72, 99] to Simulated Annealing (SA) [1] and local search [86].

Edon Gashi *et al.* [1][2] proposed a SA approach which placed among the top 5. This approach starts with a pre-processing technique that removes all time options from the domain of the class that is incompatible with the set of possible rooms (details in Section 5.2.1).

Muller *et al.*[3] [64] proposed an iterative forward search algorithm [100] based on *UniTime*. The algorithm has similarities with local search, but operates over feasible partial solutions. The algorithm ends when a complete solution is found, and then two techniques are used to optimize the obtained solution: hill climbing and great deluge. This solution, like the one proposed in this work, splits the problem into two sub-problems: student sectioning and class assignment. This solver did not enter the competition since it is authored by the organizers of the ITC 2019.

Holm *et al.*[71, 99] proposed a mixed integer programming approach based on the work from Lindahl *et al.* [97]. As mentioned above, the complexity of the novel constraints, the existence of student sectioning problems, and the notion of weeks/days cause this solution to be impractical. Unfortunately, not much detail is provided regarding this approach that won the competition. When compared to the past approaches, the main difference in this approach is the decomposition of the problem. This decomposition allows the algorithm to solve the problem in parallel with strong heuristics to guide the search. Moreover, this approach takes up to ten days to optimize the solution.

Between ITC 2007 and ITC 2019, a different timetabling competition was organized. ITC 2011 [101] focused on solving high-school timetabling problems. The problem of high-school timetabling is similar to the problem of university course timetabling. Naturally, the constraints and data characteristics are different. Demirović *et al.* [102] proposed a new algorithm that combines local search with a novel MaxSAT based large neighborhood search. The goal is to jump-start the neighborhood based MaxSAT solver with an initial solution from the local search. The jump start process

[2]The code is available at `https://github.com/itc2019/edon-gashi`.
[3]The code is available at `https://github.com/itc2019/tomas-muller`.

is a common technique in both train scheduling and university course timetabling when solving the MPP.

This year, a new timetabling competition in sport timetabling took place (https://www.sportscheduling.ugent.be/ITC2021). This problem is a bit different from the ones discussed in this thesis. However, some approaches are similar to the ones proposed in this thesis.

3.2 Train Scheduling

The generation of railway timetables is known to be intractable for a single track [103]. Nonetheless, different methods have been proposed to effectively solve this problem [8, 104–107]. In the past, a few challenges have been organized [108, 109]. In the 2014 ROADEF/EURO challenge for the shutting yard operations, most participants submitted solvers implemented greedy algorithms [108]. The winner used a mix of ILP and greedy heuristics [110].

Recently, the SBB challenge [109] motivated the appearance of new approaches ranging from ILP and ASP [3] to a greedy algorithm. The proposed ILP solution decomposes the problem into two sub-problems: routing and scheduling. This decomposition may remove the optimal solution but drastically reduces the size of the problem at hand. The ASP solution uses a hybrid ASP solver with difference constraints (*e.g.* $u - v \leq d$ where $u, v, d \in \mathbb{Z}$). Furthermore, the optimization problem is solved with an approximation cost function that allows reducing the size of the problem. However, it may remove the optimal solution. The greedy algorithm, winner of the challenge, solves the most *critical* conflicts first. A conflict between two trains occurs when both trains occupy the same resource at the same time. The flexibility of the time constraints and the density of resources are used to determine the *criticality* of a conflict. The density of resources is defined by the number of trains that require the same resource at the same time.

In the context of SAT, Matos *et al.* [4, 11] proposed a binary search procedure which uses a SAT solver to get global minimum solutions concerning travel time and a procedure to compute a better upper bound for the solution value and speed up the search process. The resulting tool is able to solve all the instances from the PESPlib benchmark [111] without exceeding the memory limit (64 GB). However, it does not ensure optimality. Even so, this approach was able to improve the best currently known value in 7 out of 20 of the PESPLib benchmark.

3.3 Scheduling under Disruptions

There are two main lines of thought to coupe with the uncertainty involved in modeling dynamic real-world problems: (i) solution reformulation [112] and (ii) generating robust solutions [113].

Reformulation is the task of changing a solution in order to return feasibility. In some applications, changing a solution completely is a nuisance and expensive. Minimizing the changes facilitates the communication, to all parties involved, of the new solution. Therefore, ensuring the new solution is known and executed. Therefore, it is important to find a solution as similar as possible to the original. This task is known as the Minimal Perturbation problem [46]. Finding similar solutions is the

opposite problem of finding diverse solutions [114, 115]. Solving the MPP requires the use of metrics to evaluate the similarity of the solutions.

The second approach focuses on avoiding reformulation of the solution by obtaining a solution that can be valid after the disruption occurs. Therefore, the robust solutions may be sub-optimal but are valid for the most common future disruptions. The search for a robust solution has its advantages since it reduces the need for re-formulation of the solution, which can have some additional financial cost (*e.g.* overtime hours). Furthermore, customers dislike changes in their lives, and re-formulating the solution causes a reduction in customer satisfaction.

This section is organized as follows. Section 3.3.1 discusses the different metrics to evaluate the similarity between solutions. Section 3.3.2 focuses on the different techniques to find similar solutions. Section 3.3.3 focuses on robustness techniques and metrics. Section 3.3.4 discusses the application of reformulation techniques to university course timetabling. Finally, section 3.3.5 discusses the application of reformulation techniques to train scheduling.

3.3.1 Metrics

In the worst case, finding similar solutions is NP-complete [116]. One important aspect of finding similar solutions is evaluating their similarity.

There are two main types of metrics to evaluate the distance between two solutions: domain dependent and domain independent.

The domain dependent metric of a problem takes into account the specificities of the problem. However, these types of metrics are normally difficult to apply to different contexts as they require domain knowledge.

On the other hand, a domain independent metric can be applied to all problems without any personalization based on knowledge of the problem at hand. However, these metrics do not evaluate precisely the differences between solutions. One of the most common domain independent metrics used is Hamming distance [117].

Definition 11 (Hamming Distance)**.** The Hamming distance (HD) is a domain independent metric which evaluates the similarity of two solutions by comparing the values of different Boolean variables. Consider the solutions s_0 and s_1. The HD is given by:

$$\delta_H(s_0, s_1) = \sum_{v \in \mathcal{V}} s_0[v] \neq s_1[v] \tag{3.1}$$

where $s_i[v] \in \{0, 1\}$ is the value assigned to the Boolean variable v in the solution s_i. The HD is the sum of Boolean values ($s_0[v] \neq s_1[v]$). □

Example 11 (Hamming Distance)**.** Recall the CSP shown in Example 9:

- $\mathcal{V} = \{A, B, C, D\}$,
- $\mathcal{D}_A = \mathcal{D}_B = \mathcal{D}_C = \mathcal{D}_D = \{1, 2, 3, 4, 5, 6\}$,

Table 3.1: Number of students enrolled in each class.

A	B	C	D
289	363	243	236

- $\mathcal{C}_p = \{\text{alldifferent}(A, B, C),\ C \neq D\}$,
- $s_0 = \{(A, 1), (B, 2), (C, 3), (D, 4)\}$,
- $\mathcal{C}_N = \{D = B\}$.

The new constraint $\mathcal{C}_N = \{D = B\}$ causes the solution s_0 to be no longer feasible. There are two optimal solutions when considering the Hamming distance: $s_1 = \{(A, 1), (B, 2), (C, 3), (D, 2)\}$ and $s_2 = \{(A, 1), (B, 4), (C, 3), (D, 4)\}$. The optimal solutions have the following cost $\delta_H(s_0, s_1) = \delta_H(s_0, s_2) = 2$, since the two classes change the assignment.

Some logical-based encoding uses auxiliary variables which do not directly influence the model. To avoid this, usually, the Hamming distance (HD) is only applied to the main variables. The HD was used as a metric to evaluate the differences between solutions to the examination timetabling [118] and course timetabling [45]. This distance can be generalized to take into consideration the differences between the variables domains. This new distance is called Manhattan [119], and the definition is as follows.

Definition 12 (Manhattan Distance)**.** Manhattan distance can be considered a generalization of the Hamming since it has the same value when considering only Boolean variables. Consider the solutions s_0 and s_1. The distance between s_0 and s_1 is

$$\delta_M(s_0, s_1) = \sum_{v \in V} |s_0[v] - s_1[v]|. \tag{3.2}$$

□

Example 12 (Manhattan Distance)**.** Consider the CSP shown in Example 11, where $s_0 = \{(A, 1), (B, 2), (C, 3), (D, 4)\}$ is the initial solution and $s_1 = \{(A, 1), (B, 4), (C, 3), (D, 4)\}$ is the new found solution which minimizes the number of unsatisfied soft constraints. Considering δ as the HD $\delta_H(s_0, s_1) = 1$. However, if one considers δ_M as the Manhattan distance $\delta_M(s_0, s_1) = 2$.

Nevertheless, the HD (and other domain independent metrics) has its problems caused by being a generalist measure (see Example 13).

Example 13 (Domain Dependence)**.** Consider Example 11. Additionally, the number of students enrolled in these classes is shown in Table 3.1. Recall that both solutions have the same Hamming distance: $\delta_H(s_0, s_1) = \delta_H(s_0, s_2) = 1$. However, this value does not take into account the number of stakeholders (teachers and students) affected by the change. This is a multi-objective optimization

problem where the objectives are the HD and the number of stakeholders affected. s_1 is the optimal solution since it only affects 236 students. In this case, the solution is in the Pareto frontier.

Moreover, some solutions are different in theory but maybe the same in practice. These solutions are feasible and have the same cost (thus being considered the same in practice). To avoid generating and testing this solution, it is important to consider the concept of symmetries [19, 120] which is defined as follows.

Definition 13 (Symmetry)**.** Consider two different assignments to a variable. A value symmetry [121] is a bijective mapping m on the values such that if $v_1 = d_1, \ldots, X_n = d_n$ is a solution then $x_1 = m(d_1), \ldots, x_n = m(d_n)$ is also a solution. □

The definition above can be extended to constraint symmetry and solution symmetry [120]. The symmetries can be used to cut the problem search space. These types of cuts have already been applied to timetabling [122]. As Example 14 shows, not cutting these solutions from the search space can cause errors in the judgment of domain independent metrics.

Example 14 (Symmetry)**.** Consider the solution s_0 from Example 11. Let us assign these classes to two rooms. We can allocate all classes to one room since all time slots are different. For this reason, choosing to allocate all classes in the first room or choosing to allocate all classes in the second room are two different solutions. For a domain independent metric, these two solutions would be considered different. However, the difference between them has no real implications for the stakeholders.

There are different metrics in order to evaluate the proximity of the solutions that can be applied only to planning [123]. For example, Nguyen *et al.* [123] proposed a metric to compare two plans given by the ratio between the number of actions that appear in both plans and the total number of actions.

Petit *et al.* [114] proposed a framework to find diverse solutions to COP using domain independent metrics (Hamming, Manhattan, and Euclidean distances). The implemented solution was designed to maximize the ratio between the distance between the solutions and the loss in quality. The quality is evaluated based on the optimization goal expressed in the definition of the original problem.

Example 15 (Petit's Quality Ratio)**.** Consider again Example 10 where the solution $s_0 = \{(A, 3), (B, 2), (C, 1), (D, 4)\}$ is optimal when considering only the Hamming distance. Consider also the solution $s_1 = \{(A, 1), (B, 2), (C, 3), (D, 1)\}$ that is optimal in terms of the number of unsatisfied soft constraints (only 1 unsatisfied soft constraint). According to the proposed proposed ratio [114], solution s_1 has ratio 3, where solution s_0 has ratio 0. In this case, s_1 would be chosen since it has a better diversity over quality.

Hebrard *et al.* [124] proposed a different type of distance-based metric where the goal is to find solutions that have (or avoid) certain details of interest, which are called ideals (non-ideals). Each ideal or non-ideal corresponds to an assignment. For example, finding a solution where the class A is taught in the 1 time slot. The proposed metric evaluates the proximity of the solution to the ideals (non-ideals) by applying the HD for only the variables present in the sets. Furthermore, it is possible to specify if *all* or *at least one* of the elements of the sets should be considered in the solution. The metric is formally defined as follows:

Definition 14 (Preference Distance)**.** Consider a CSP problem where $\mathcal{V}$ is a set of variables and the respective solution s corresponds to a set of assignments. Consider also the sets of ideals and non-ideals $\mathcal{I}$ and $\mathcal{A}$, respectively. The level of importance of an assignment $(v, x) \in \mathcal{A} \cup \mathcal{I}$ can be customised by weight $w_{(v,x)}$.

The distance between a solution s and the sets $\mathcal{I}$ and $\mathcal{A}$ can be defined by the distance: $\otimes_{(v,x)\in\mathcal{A}\cup\mathcal{I}}\Theta(s,(v,x))$, where $\otimes = \{max, min\}$ and

$$\Theta(s,(v,x)) = \begin{cases} w_{(v,x)} * (s[v] \neq x) & (v,x) \in I \\ w_{(v,x)} * (|\mathcal{V}| - (s[v] \neq x)) & (v,x) \in A \end{cases} \tag{3.3}$$

The metric can evaluate the solution quality based on if *all* ($\otimes = max$) or *at-least one* ($\otimes = min$) of the elements of the sets should be considered. $s[v] \neq x$ is the HD applied between only two variables. □

Example 16 (Preference Distance)**.** Consider again the CSP where the variables $\mathcal{V} = \{A, B, C, D\}$ with the domains $\mathcal{D}_A = \mathcal{D}_B = \mathcal{D}_C = \mathcal{D}_D = \{1, 2, 3, 4, 5, 6\}$ are subject to the constraints $\mathcal{C} = \{\text{alldifferent}(A, B, C),\ C \neq D\}$. Additionally, consider that we want the solution with the ideal $A = 1$, with the non-ideal $D = 6$ and the weights $w_{(A,1)} = w_{(D,6)} = 1$. The distance to a solution s is computed by: $max(1 * (s[A] \neq 1), 1 * (|\mathcal{V}| - (s[D] \neq 6)))$. One solution that minimizes the distance is $\{(A, 1), (B, 2), (C, 3), (D, 1)\}$.

The HD and the metric described above have the same result if each variable appears exactly once in the set of ideals $\mathcal{I}$. In addition, this metric provides a comparison of partial solutions and the possibility of combining the objectives of finding similar (the ideals) and diverse (the non-ideals) solutions. The ideals and non-ideals can be seen as additional soft constraints that account for the distance between two solutions.

In the context of SAT, there is also work on solving the Distance-SAT problem [125, 126]. This problem consists of determining whether a propositional CNF formula admits a model that disagrees with a given partial interpretation on at most d variables. In other words, the goal is to find a new model that has an HD of at most d. The proposed algorithms focus on solving the problem from scratch with a distance metric.

3.3.2 Similar Solutions

In this section, the current state-of-the-art solutions to find similar or diverse solutions are discussed. Table 3.2 shows a summary of the distance metrics used in several methods found in the literature to find similar or diverse solutions and their respective applications. The computational complexity class for the problem of finding the solution which is the closest to an original solution is FPNP$^{[logn]}$ – complete [115, 127].

Constraint programming applies filtering algorithms to remove impossible values from the domain of the variables to speed up the search. These filtering algorithms can be associated with a specific constraint (*e.g.* alldifferent) to reduce the domains of the variables involved in the constraint. When considering the search for similar/diverse solutions, filtering algorithms are proposed to deal with the new distance constraint. The filtering algorithms proposed for the search for a similar solution can be easily changed for the search for a diverse solution and vice versa. The filtering algorithms discussed in this section ensure Generalized Arc Consistency (GAC) [128], *i.e.*, all the values in the domain of the variables are possible, taking into consideration the constraints in which the variables are involved.

Hebrand *et al.* [115] proposed a filtering algorithm for the Preference distance constraint. We can recall that this distance is based on a set of ideals and non-ideals (elements that we want to have or avoid in a solution). However, the filtering algorithm was designed to deal only with ideals. Nevertheless, with few changes, one can consider both. The algorithm starts by checking the lower bound for each ideal (the value for which the δ is minimal). Then, for each value of each variable's domain, the algorithm verifies if the corresponding ideal is within an acceptable distance (max). This is decided based on the lower bound: $\delta_H(v, i) - lb[v] > max$, where $v \in \mathcal{V}$, $i \in I$ and $lb[v]$ the lower bound for variable v. If the value exceeds the pre-defined limit (max), the value from the variable's domain is removed. If the sum of all lower bounds is larger than the maximum distance allowed there is no feasible solution to the problem. Note that the distance metric used to check the proximity of the ideals is the HD. The algorithm's time complexity is $\mathcal{O}(ndk)$, for each iteration of the search, where $n = |V|$ (the number of variables), d is the maximum domain size, and k the size of the ideals. In the worst case, $k = n$.

Example 17 (Filtering algorithm)**.** Let us consider again Example 16 but this time without the set of non-ideals. In this new example, the distance to a solution s is computed by: $1 * (s[A] \neq 1)$ and therefore one optimal solution is $\{(A, 1), (B, 2), (C, 3), (D, 1)\}$. Consider that $max = 0$. First, the lower bound of A is computed. The lower bound of variable A is $lb[A] = 0$ since the assignment $(A, 1)$ would cause the distance to be 0. The sum of the lower bounds is lower than max, and therefore the problem has a feasible solution. All the values different from 1 are removed from D_A as the values would cause the solution to have a distance greater than max. Nothing happens to all the other domains since this constraint (distance between a solution and the set of ideals) does not involve them.

Petit *et al.* [114] proposed a filtering algorithm for the Manhattan distance. The filtering algorithm is based on two input factors: the minimal acceptable value of diversity (similarity) and the maximally

acceptable loss of quality. Informally, if a given value in the domain causes the solution to exceed the two input factors, the value is going to be removed. This algorithm runs in $\mathcal{O}(k \sum_{i \in D} |D_i|)$, where k is the size of the original solution. Note that the original solution may be only a partial solution (*i.e.* the solution does not contain an assignment to all variables).

Example 18 (Filtering algorithm cont.)**.** Let us consider again Example 17 but now the Manhattan distance (δ_M) is used instead of the preference distance (δ_P). Moreover, instead of the set of ideals considered in Example 17, we consider $s_o = \{(A, 1), (B, 2), (C, 3), (D, 4)\}$ as the original solution. Additionally, consider that the constraint $C \neq D$ changes to $B > C$. Regarding the quality, we consider that it is not acceptable to lose quality while searching for a new solution. In terms of distance, we consider $\delta_M < 4$. Note that the search procedure is not the goal of this example. Let us assume that the search procedure chooses the variables to be assigned in alphabetical order.

The first set removes all values from the domain of variables for which the constraint $\delta_M < 4$ would be unsatisfied. For example, in $D_A = \{1, 2, 3, 4, 5, 6\}$ the values 5 and 6 would be removed since the assignments (A,5) and (A,6) would alone make $\delta_M < 4$ unsatisfied. Considering a heuristic that sets a preference for the original values, the first variable assigned is $(A, 1)$. Therefore, the value 1 is removed from the domain of B and C. The assignment $(B, 2)$ causes all values larger or equal than 2 to be removed from D_C. Now, the assignment $(C, 1)$ causes the solution to be already at a distance of $\delta_M = 2$. Therefore, the values 1, 2 and 6 can be removed from the domain of D_D since the assignments would exceed the distance limit. The last assignment is $(D, 4)$.

Barcelogic [129] approach uses a logic-based tool (SAT solver) to find close solutions. The variables domain, in this specific case, is Boolean. Therefore, the HD can be applied to evaluate the distance between two solutions. The constraints may require the use of auxiliary variables, and thus the HD should not be applied to the complete set of variables. The heuristic to assign values to variables first tries to assign values from the original solution. When an assignment of a value to a variable causes the current solution to have a higher cost than a previously found one, a new constraint is created, and the backtrack happens since no good solution will appear in this branch of the search tree. The new constraint ensures that the search in this branch stops. To sum up, this process modified a logic-based tool in order to find similar solutions. The heuristic guides the assignment of variables to be as close as possible to the original solution. However, it does not take full advantage of the initial search. For example, it could use the initial search tree to prune the search space by knowing which branch to visit.

Eiter *et al.* [127] proposed an off-line method to find multiple similar and diverse solutions. With a logic-based tool, a set S of solutions is computed. To find multiple solutions within a distance d from this set, it is possible to encode this problem as a Clique problem [130]. Consider a graph $G(V, E)$, where V is a set of vertexes corresponding to solutions, and E is a set of weighted edges corresponding to the distance between solutions. The objective is to find a clique whose weight is below d. Solving the clique decision problem is NP-complete.

Zivan *et al.* [131] proposed a two-stage method to solve MPP considering the HD as the distance metric. The first stage consists of a branch and bound scheme with backtracking to find the largest partial solution for which the HD is 0. The second stage of the algorithm assigns values to the remainder of the solution.

Independent of finding the complete solution feasible or not, the algorithm restarts to the first stage to find a new partial solution. If the algorithm finds a feasible solution, this solution is stored only if the value of the HD for this solution is better than the previously stored one. Note that a solution is infeasible if any hard constraints are not satisfied. The algorithm ends when the optimal solution (in terms of distance) is found. This algorithm only considers one soft constraint: the Hamming distance. All the constraints in the problem's domain are encoded as hard.

Example 19 (Zivan's method)**.** Consider the CSP explained in Example 18 with the following constraints: $\mathcal{C}_h = \{\ C \neq D, \text{alldifferent}(A, B, C)\}$. Recall that the domain of the variables is $\mathcal{D}_A = \mathcal{D}_B = \mathcal{D}_C = \mathcal{D}_D = \{1, 2, 3, 4, 5, 6\}$. As explained above, the optimal solution is: $\{(A, 1), (B, 2), (C, 3), (D, 4)\}$. However, the constraint $C \neq D$ was changed to $C > D$ and thus, the solution is no longer valid. The final search tree is shown in Figure 3.1, where ub and lb represent the values of the upper and lower bounds, respectively. The assignments made in the first and second stage are circles and rectangles, respectively.

The upper bound (*i.e.* the maximum value that δ_H can take) is 4. The lower bound is updated only when the original value is no longer in a variable domain. The first assignment made by the algorithm is $(A, 1)$. The alldifferent constraint and the first assignment cause the value 1 to be removed from the $\mathcal{D}_B$ and $\mathcal{D}_C$. In the next step, the algorithm assigns $(B, 2)$. For the same reason, the value 2 is removed from the domain of the other variables involved in the alldifferent constraint. The next step consists of assigning $(C, 3)$. Finally, when assigning $(D, 4)$, a conflict arises, and so the second stage of the algorithm starts. Therefore, the domain is updated: $\mathcal{D}_D = \mathcal{D}_D \backslash \{4\}$. As the domain of D does not contain four, the lower bound is 1.

At this stage, the algorithm searches for a different assignment to D. However, no feasible solution is found, and consequently, the backtracking process unassigns the value of C. The value 3 is removed from the $\mathcal{D}_C$. Therefore, D is the only unassigned variable with a value from the original solution in its domain. The first stage of the algorithm restarts with the assignment $(D, 4)$. The lower bound is still 1 since $\mathcal{D}_C$ cannot be assigned the original value. As there are no more unassigned variables with an original value in its domain, the second stage returns to the assignment $(C, 5)$. The upper bound is going to be updated with the value 1, and this solution is stored.

In the next step, the algorithm removes the assignment of C and D. The value assigned to D is removed from its domain to avoid revising this branch of the tree. As no other assignment is possible, the backtracking continues removing the assignment of B. The value assigned to B is removed from its domain, and the value (4) previously assigned in D is added to its domain. The search continues until it is possible to conclude that there is no solution with a lower distance.

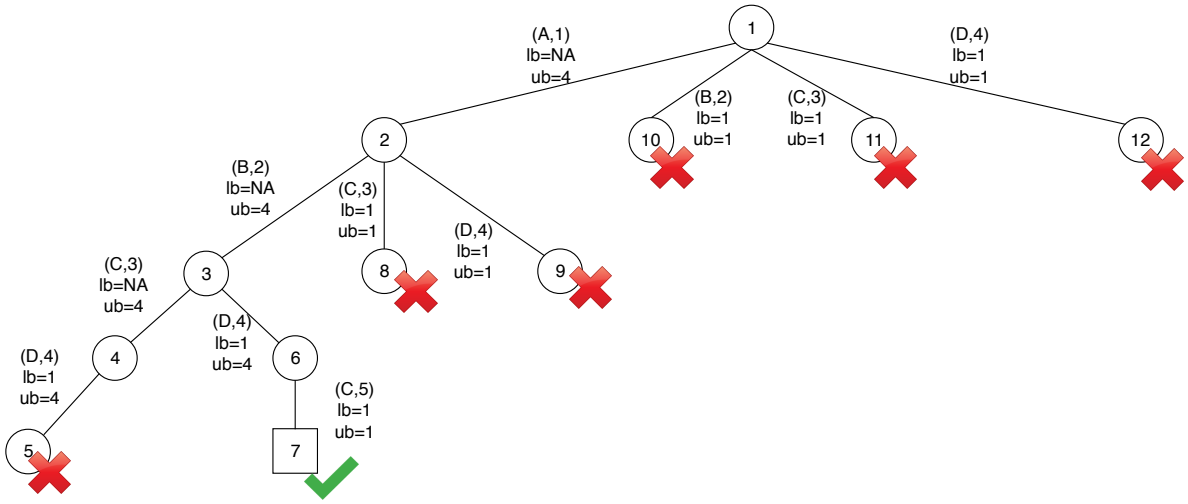


Figure 3.1: The final search tree for Example 19, where ub and lb represent the values of the upper and lower bound respectively. The assignments made in the first and second stage of the method proposed by Zivan *et al.* are circles and rectangles respectively.

3.3.3 Robustness

Besides the reformulation approach, one can prepare the solution in order to resist disruptions (changes in the original constraints). These types of techniques have been successfully applied to railway aperiodic timetabling [137]. However, this subject is not the main goal of this work, and thus only a general description is given. Finding a robust solution may have two main advantages: reducing the cost that re-solving brings (*e.g.* avoiding overtime payments) and improvement in customer satisfaction. For example, when considering railway aperiodic timetabling problems, the customers do not want to have their travel plans changed (even if it is the smallest impact). However, it is impossible to predict everything, and therefore the re-solving process is also needed. More details can be found elsewhere [138–140]. *Robust* optimization is a method to deal with the uncertainty present in real-life optimization problems. This type of solutions is designed from the beginning to be *robust*, avoiding the need to re-formulate the problem [112, 139, 141]. More formally, a *robust* solution for the CSP problem can be defined as follows.

Definition 15 (Robust Solution)**.** Consider a CSP P and a set Z of possible disruptions. A solution s is considered to be *robust* to disruptions in Z if and only if the solution s is still valid after all the disruptions happen. □

The solution resulting from applying this concept has a trade-off between quality and robustness. The quality of a solution may be worse in order to avoid the need for reformulation, as shown in Example 20.

Example 20 (Robust Solution)**.** Consider, for example, the following CSP: $\mathcal{V} = \{A, B, C, D\}$, $\mathcal{D}_A = \mathcal{D}_B = \mathcal{D}_C = \mathcal{D}_D = \{1, 2, 3, 4, 5, 6\}$, $\mathcal{CH}_p = \{\text{alldifferent}(A, B, C)\}$, $\mathcal{CS}_p = \{B = 1, D = 2\}$. Now consider $\mathcal{Z} = \{D = B\}$ as the single disruption we want to take into account in the solution. Additionally,

Table 3.2: Summary of the distance metrics used in the methods to find similar or diverse solutions and their respective applications.

		Metric					Objective		
		Domain Independent							
		Hamming	Manhattan	Euclidian	Other	Dependent Domain	Similar	Diverse	Application
Kingston *et al.*	[132]				✓		✓		Timetables
Müller *et al.*	[10]	✓					✓		Timetables
Phillips *et al.*	[44]					✓	✓		Timetables
Eiter *et al.*	[127]	✓				✓	✓	✓	Phylogenies Planning
Hebrand *et al.*	[124]	✓	✓		✓		✓	✓	Car Configuration
Petit *et al.*	[114]	✓	✓	✓				✓	Travelling Salesman Problem Sorting chords
Vadlamudi *et al.*	[133]					✓		✓	Planning
Abío *et al.*	[129]	✓					✓		Scheduling Tournament Curriculum-based Timetabling
Bailleux *et al.*	[126]	✓					✓		Planning
Lemos *et al.*	[13]					✓	✓		Biological Networks
Samaga *et al.*	[134]					✓	✓		Biological Networks
Merhej *et al.*	[12]					✓	✓		Biological Networks
Zivan *et al.*	[131]	✓					✓		Meeting Scheduling Problems
Roos *et al.*	[135]	✓					✓		Random CSP instances
Perez-Lopez *et al.*	[136]					✓	✓		Surgery Scheduling
This thesis		✓				✓	✓		Timetables

consider a lexicographic order where the disruptions are the first optimization objective.

A possible *robust* solution is $\{(A,3),(B,2),(C,1),(D,2)\}$. In case of occurring the disruption in Z, the solution still holds. However, if the disruption does not occur, the solution found is not optimal since B could be assigned the value 1.

There are different levels of robustness, depending on the number of disruptions and their likelihood, for which the solution remains valid. An example of a robustness metric [113] for a solution s (for a discrete domain problem) considering the resistance to a disruption of probability $p(z)$ is given by:

$$R^s_{F,Z} = \sum_{z \in Z} p(z) * F(z) \tag{3.4}$$

where $F(z)$ is a function which evaluates the satisfiability of the solution when a disruption z occurs.

Demirović *et al.* [142, 143] proposed a novel algorithm to obtain robust and recoverable team formation[4]. The team formation problem is known to be NP-hard. Demirović *et al.* proposed different algorithms with a detailed complexity analysis. These algorithms are designed to explore the trade-off notion between robustness and recoverability. The idea of recoverability is to guarantee that a solution can easily be repaired after disruptions. This notion is closely related to the idea of super-models used in this thesis.

Another solution can be a compromise between finding similar and robust solutions. One possibility is the use of the concept of supermodel defined[144] as follows.

[4]The goal of solving the team formation problem is selecting a team of agents with the minimum cost such that a specific set of skills is covered.

Definition 16 (Supermodels)**.** The solution s to a CSP is (m, n)-super solution if and only if the change of the values of at most m variables can be repaired by assigning other values to these variables, and modifying the assignment of at most n other variables. □

Finding a supermodel can be seen as a soft version of finding a robust solution. In this case, the solution does not need to be robust for disruptions but needs to be easily repaired. Similar to the robust solutions, a supermodel may be worse in terms of the solution quality. However, if the solution is not robust, one needs to re-formulate the solution. Finding super-models is a NP-complete task [145], if m and n are constants.

3.3.4 University Course Timetabling

This section focuses on the state-of-the-art approach to solve the MPP in an university timetabling setting. In the literature, there are several different approaches, namely: constraint programming [131, 146, 147], ASP [67], integer programming [44, 47, 147], and local search [45, 87, 132, 148].

Müller *et al.* [45] proposed the iterative forward search algorithm to solve the MPP applied to university course timetabling. Since this method is a local search method, it does not ensure completeness. Phillips *et al.* [44] used integer programming to solve MPP on instances from the University of Auckland. The proposed method tries to solve the problem in the smallest possible neighborhood. If a feasible solution is not found, then the neighborhood is gradually expanded until either a feasible solution is found or the neighborhood includes the whole search space.

Banbara *et al.* [67] proposed an ASP based tool to compute Pareto fronts with two objectives: (i) minimize the number of soft constraints unsatisfied and (ii) minimize the number of perturbations. The tool solves the MPP if and only if the disruption changes the course timetabling problem by adding or removing constraints. Therefore, all disruptions changing the domain of the problem cannot be solved by this tool.

Recently, Lindahl *et al.* [47] proposed a bi-objective integer programming model to solve MPP applied to curriculum-based course timetabling. The goal is to find Pareto-optimal solutions for these two objectives: (i) minimize the number of perturbations and (ii) minimize the number of soft constraints unsatisfied. The results show that the MPP solutions often have low quality and that allowing a few more perturbations can significantly improve the quality. Lindahl *et al.* [47] approach was evaluated with data sets from the ITC 2007.

Type of Disruptions

There are different types of disruptions that were already studied in the literature. Furthermore, we analyzed the disruptions that happened during the last 5 years at IST. Table 3.3 summarizes the different disruptions that can occur in the university course timetabling from both IST and literature. The most common disruptions that can occur in university are the following:

- *Room stability*: disruption forces all classes of a course to be scheduled in the same room.

Table 3.3: Summary and literature review of the most common disruptions in an university scenario.

Disruption	[47]	[44]	[45]	[67]	IST
Room Stability				✓	
Overlap/No Overlap					✓
Invalid Assignment	✓		✓		
Invalid/Preference Time	✓	✓			✓
Invalid/Preference Room					✓
Remove Room for a day	✓				✓
Remove Room		✓			✓
Insert / Delete Curriculum	✓				✓
Modify Enrollments		✓			✓
Modify # Classes					✓

- *Overlap* (*no overlap*): disruption refers to the addition of a constraint to forbid (force) classes to be taught simultaneously. For example, *no overlap* disruption may arise when the number of teachers assigned to a course reduces to one.
- *Invalid assignment*: disruption refers to a problem in the assignment of the class to a room or to a time slot. This disruption abstracts the origin of the problem (time or room assignment). However, in a real-world scenario, it is more common to find more specific disruptions.
- *Invalid time* and *invalid room*: are disruptions that make an assignment of a class to a time slot and a room invalid, respectively. These two last disruptions are more specific than the *invalid assignment* disruption. These disruptions can be further specified when only a time slot or a room is unavailable to only a specific set of courses. *Preference time* and *room preference* are the opposite disruptions of the *invalid time* and *invalid room*.
- *Remove room for a day* and *remove room*: cause a room unavailability for an assignment for a day and forever, respectively.
- *Insert / Delete curriculum*: adds/removes a set of courses. When a new curriculum is added, each class of the new courses cannot be overlapped in time. All constraints relating to classes are applied (*e.g.*, room capacity).
- *Modify enrollments*: modifies the number of students that are enrolled in a class, since it naturally changes between the execution of consecutive academic terms. This disruption may not require changes in the original solution since the difference may still be supported.
- *Modify the number of classes*: adds/removes the number of possible classes that a student can attend. The number of classes does not change for each student. However, the students have more/less flexibility in their choice of timetable. This disruption can be a side effect of the *modify enrollments* disruption.

Example 21. Let us consider the timetable shown in Figure 3.2a. The weekly timetable shows the different classes of the different courses a student can attend. Each student having this curricular plan must attend some classes from the four courses represented (A to D). For each course, a student must

Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30			B^{PB} (R2)		
8:30-9:00					
9:00-9:30	A^T (R1)		B^T (R1)	A^T (R1)	B^T (R1)
9:30-10:00					
10:00-10:30					
10:30-11:00	A^L (R4)	A^L (R3)	C^T (R1)	D^{PB} (R2)	C^T (R1)
11:00-11:30			D^T (R1)		D^T (R1)
11:30-12:00					
12:00-12:30	C^T (R1)		C^T (R1)	D^{PB} (R2)	
12:30-13:00					
13:00-13:30	D^T (R1)	C^{PB} (R2)			B^{PB} (R2)
13:30-14:00					

(a) Before

Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30			B^{PB} (R2)		
8:30-9:00					
9:00-9:30	A^T (R1)		B^T (R1)	A^T (R1)	B^T (R1)
9:30-10:00					
10:00-10:30					
10:30-11:00	A^L (R4)	A^L (R4)	C^T (R1)	D^{PB} (R2)	C^T (R1)
11:00-11:30			D^T (R1)		D^T (R1)
11:30-12:00					
12:00-12:30	C^T (R1)		C^T (R1)	D^{PB} (R2)	
12:30-13:00					
13:00-13:30	D^T (R1)	C^{PB} (R2)			B^{PB} (R2)
13:30-14:00					

(b) After

Figure 3.2: Timetable for a class of students (a) before and (b) after occurring two disruptions: (i) an *unavailability* constraint over room R3 and (ii) a *no overlap* constraint *w.r.t.* the two lab classes of course A. The colors represent the different rooms the classes are assigned to.

attend all theoretical classes and only one practical / laboratory class. For example, a student must attend all A^T (theoretical) classes in the schedule and only one of the two A^L (laboratory) alternative classes.

Consider the following disruptions applied to the given example: (i) room R3 is closed for renovations for a long period of time, and (ii) the number of teachers available to teach A^L is now only one. The second disruption means that classes A^L must be held at different times. Consider that we want to cause the smallest number of perturbations (δ) to the original solution. Disruption (i) reduces the domain of the problem. In this small example, one class (A^L) is assigned to R3. Therefore, the original solution (Figure3.2a) is no longer feasible. If one considers that there are only two laboratories (R4 and R3) and that R4 is already taken in the required time slot, then we conclude that the optimal solution requires two perturbations to the original solution (change room and time). Disruption (ii) adds a new *no overlap* constraint to the model. However, note that any solution containing the perturbations resulting from the first disruption already works out for this disruption. A possible solution is shown in Figure 3.2b.

The model proposed by Lindahl *et al.* [47] only considers the following disruptions: *remove room for a day*, *invalid time*, *insert curriculum*, and *invalid assignment*. With this set of disruptions, one can define an upper bound on the number of satisfied soft constraints since they cannot improve the quality of the original solution. The disruptions reduce the search space.

Example 22. Let us consider that the class A^L, as shown in Figure 3.2a, has 35 students enrolled. Rooms R4 and R5 have a capacity of 25 and 30 students, respectively. Consider that only these rooms are equiped with the laboratory requirements of the classes of course A. When generating the new timetable (based on the last semester), a disruption causes the number of students enrolled in A^L in room R5 to be reduced to only 25. This disruption causes the overall quality of the timetable to improve. Now all students attending the class A^L in room R5 can be seated. However, a small perturbation would further improve the quality of the timetable. If we switched the rooms of these two

Table 3.4: Review of the different disruptions and causes of train scheduling problems.

Cause	Disruption	Occurrence	Reference
Weather influences	Slowdown	Before/During	[150, 151]
Logistical problems	Block train	During	[151–153]
External influences	Block train	During	[152, 153]
Accidents	Block track/train	During	[107, 150, 151, 153–157]
Engineering work	Block track	During	[107, 153, 158]
Infrastructure problems	Block track	Before	[107, 150, 151, 153, 159]
Staffing problems	Partially considered		[150]
Rolling stock problems	Not considered		[150, 156, 157]

Table 3.5: Review of the different cost functions used.

Minimize	
Delay	[160]
Weighted delay with the number of passengers	[160]
Cost of changing track	[154, 156, 159]
Miss connections	[152]
Cost of compensation	[161, 162]
HD	[156]
Canceled trains	[158]
Additional connections for a passenger	[157]
Maximize	
Number of Passengers transported	[150]

classes, we would reduce the number of students above the ideal capacity from 10 to 5.

3.3.5 Train Scheduling

The generation of robust timetables (or how to recover timetables after a disruption) has been widely studied in the literature [48–50, 107]. The disruption can have multiple causes, from weather influences and accidents to staffing problems [149]. Nevertheless, we can summarize all causes into three kinds of disruptions: blocking a train, blocking a track, and changes in the travel speed of the train. Blocking a train is a disruption that affects only a single train but not the track. Blocking a track is a disruption that only affects the railway, and the trains are free to move around the rest of the track. Finally, the changes in the travel speed of the train (slowdown) are a disruption that affects a part of the track and forces the train to reduce its speed. A disruption can occur before or after (real-time [107]) the train departs from the station. Naturally, the mechanism to recover depends on location of the disruption. We have more flexibility if the train has not yet started its route. The three types of disruptions have different causes[5]. A summary of the related work is shown in Table 3.4.

The different causes of *block train* disruption are the following:

- *Logistical problems*: something unforeseen occurs in the planned routes. Therefore, logistical problems are, for example, delays on connecting trains or other services (*e.g.* buses).

[5]The real-time update of the disruptions and their causes can be accessed in https://www.rijdendetreinen.nl/en/statistics/causes.

- *External influences*: when outside forces impact the normal operation of the railway. External influences are, for example, police investigations and fire alarms.
- *Accidents*: the different types of accidents related to trains (*e.g.* collision, derail, run over). Accidents with trains arise during the travel of trains. Hence, the trains involved in the accident are delayed. No re-routing can reduce the delay.
- *Staffing problems*: strikes and sickness of the crew members may compromise the safe operation of the train. Ergo, staffing problems may cause a change in the crew's assignment to a train and cause delays in a train. In this work, we do not solve the crew's assignment problem [163, 164]. However, delays on trains due to strikes are considered. This cause is simulated by blocking a train for a period of time corresponding to the strike. This type of disruption can be mitigated by having redundant resources to assign in case of necessity.

The different causes of *block track* disruption are the following:

- *Accidents*: the different types of accidents related tracks (*e.g.* falling trees) may arise. The tracks become blocked and thus may cause a delay. We may solve this delay with the re-routing of the trains.
- *Engineering work*: unexpected engineering works on the tracks. Ergo, we need to wait or re-route the trains to avoid the tracks that cannot be used by any train.
- *Infrastructure problems*: the different railway infrastructure problems (*e.g.* signals, overhead wires) cause the need for engineering work. However, we know beforehand which tracks need work, and thus we can plan ahead.

The only cause of *slowdown* disruption is *weather influences*: the different weather patterns (*e.g.* rain, snow) can reduce the visibility and cause slippery tracks. Hence, the weather causes a reduction in train speed.

In this work, we do not consider the disruptions caused by *rolling stock problems*: a train may need replacing during its travel. This may be caused by a defective train. This problem can be mitigated by having redundant resources on key places on the track. These resources are used to re-supply the roots when needed.

Different cost functions are used in the literature when recovering timetables after disruptions. A summary of the different cost functions is shown in Table 3.5. The simplest cost function is the Hamming distance [156], *i.e.*, the number of variables that change the value between the original solution and the new feasible solution. However, the HD is domain independent and thus not a realist cost function. The most common cost function is to minimize the delay [160] or the operational cost of the delay [160–162]. The cost function can be even more detailed, additionally considering the impact of train changing the track [154, 156, 159]. A change of track by train has an impact on the operational cost and train speed.

The cost function can also focus on the impact on the passengers [150]. To reduce the impact of re-scheduling trains on the passengers, we can avoid the cancellation of trains [158], and reduce the

number of connections required by the passengers to arrive at their destination [157]. This type of cost function requires the *apriori* knowledge of the passenger routes.

As one can see, most papers focus on the optimization criteria that best capture their real-world problem. However, there is still a gap in the related work that would merge the best of all these criteria.

4

Instituto Superior Técnico Course Timetabling

Contents

This chapter describes our approaches to solve the University Course Timetabling Problem (UCTTP) at Instituto Superior Técnico (IST). This work was published in [15, 16]. There is still some work being prepared for submission [165]. The organization of this chapter is as follows. Section 4.1 describes and discusses the challenges, of course, the timetabling problem of IST, and the current handmade solutions. Section 4.2 formally describes the problem. Section 4.3 describes three different approaches to solve room usage optimization problems, which is one of the major problems in IST. Section 4.4 describes two different integer programming models to solve course timetabling under disruptive scenarios. Section 4.5 discusses the results obtained by the different methods. Finally, Section 4.6 concludes the chapter. Appendix A describes the data cleaning and transformation process.

4.1 A case study at Instituto Superior Técnico

This section presents the case study of timetables at Instituto Superior Técnico (IST)[1]. The current handmade timetables and their problems are the main motivation for this work.

IST is part of Universidade de Lisboa, which is the major university in Portugal and one of the largest in Europe. IST is a higher education institution focused on the fields of architecture, engineering, science, and technology and promoting excellence in teaching, research, development, and innovation activities.

IST offers 86 different higher education degrees with approximately $11\,412$ students enrolled (including graduation and post-graduation students) and $1\,200$ teachers. The classes are scheduled from 8:00 to 20:00, five days a week, without mandatory lunch breaks. These classes can have different durations, from one hour to three hours. IST has two main student campi: Alameda and Taguspark with 94 and 21 rooms, respectively. Each room has a specific capacity. The median capacity of the rooms in Alameda is 58 and in Taguspark is 48. The standard deviation for the room's capacity in Alameda and Taguspark are 31.25 and 64.28, respectively.

Besides offering degrees, IST hosts multiple aperiodic events (*e.g.* student-organized events, workshops, and conferences) on campus. Many times, these events are taking place on the same days as classes from different courses. All these new events must be scheduled, when required, in suitable rooms.

Presently, the generation of timetables is handmade. The timetables for the current year (2020/2021) were still generated by hand, based on the previous year's timetable. This approach reduces the scope of the problem to solve each year since we must change only a small set of events.

Solving university timetabling manually makes it particularly difficult to optimize space usage. The large variety of rooms available makes choosing the most suitable room for a class an even more important decision.

The main problems at IST, in terms of space usage, are (i) the difficult task of scheduling aperiodic

[1] https://tecnico.ulisboa.pt

Table 4.1: Data sets characteristics.

	Taguspark		Alameda	
	1^{st} Semester	2^{nd} Semester	1^{st} Semester	2^{nd} Semester
# Courses	108	101	1 022	1 015
# Degrees	8		78	
# Classes	290	246	2 111	1 611
# Students	51 523	42 578	283 745	219 217
# Hall Rooms	5		36	
# Rooms	21		94	

Table 4.2: Characteristics of the rooms: large hall R_{lh}, hall R_h and other rooms R_r.

	Alameda			Taguspark		
Type	R_{lh}	R_h	R_r	R_{lh}	R_h	R_r
Average Capacity	148	130	55.77	252	130	36.41
standart deviation	13.85	0	25.18	0	0	11.43
# of rooms	3	4	113	2	3	48

Table 4.3: Number of students versus number of seated students in the handmade solution.

Campus	**Alameda**		**Taguspark**	
Semester	1^{st}	2^{nd}	1^{st}	2^{nd}
Total number of seated Students	268 668	210 958	51 006	42 286
Total number of students	283 745	219 217	51 523	42 578

events and (ii) closing down a room due to unpredictable problems.

When considering these problems, it is important to establish the most compact timetable for each room beforehand, at the beginning of each semester. Typically these problems arise during the semester, and thus it is important to be able to re-optimize the timetable without changing the original time slots.

4.1.1 Current handmade timetables

The data used in this work is divided into four data sets corresponding to the classes at the Alameda campus and Taguspark campus for both semesters of 2016/2017. The differences between the data sets are summarised in Table 4.1. Note that the total number of students is the summation of each enrolment in a class. Therefore, this number includes double-counting of students. The data sets corresponding to the Alameda campus are larger, in particular for the first semester. The rooms are grouped by capacity and usage. Rooms are usually divided into three types: hall rooms, rooms (R_r), and laboratories. Here, only the first two are considered. Hall rooms can be split into two: large hall rooms (R_{lh}) and hall rooms (R_h). Table 4.2 shows the summary of the rooms and their average capacities for each campus. R_r is the most heterogeneous set of rooms.

The timetables at IST are mixed between curriculum-based timetabling and post-enrollment timetabling. The undergrads normally have to follow a strict curriculum plan, and therefore undergrads' timetables are curriculum-based. On the other hand, as students progress in their studies, they have more flexibility to choose what course to enroll. In this light, the generation of timetables for these courses can be considered a post-enrollment timetabling problem.

The handmade timetables for the academic year of 2016/2017 do not seat all the students enrolled in the classes. The number of students seated and the total number of enrolled students are shown in Table 4.3. The total number of enrolled students seated may be impossible to achieve due to space restrictions.

4.2 Problem Definition

These periods are separated into working days D. Each day has a set of consecutive working time slots of half an hour, $T \in \{0, ..., 23\}$. Let us consider a set of periods $P \in \{0, ..., 119\}$ corresponding to all possible time slots of a working week ($|P| = |D| \times |T|$). The university has a set R of rooms where classes can be scheduled. All university classes C (from different courses) have to be assigned to a time slot and to a room.

Consider a set of courses $Course$, with each course $course \in Course$ having a set of classes C_{course} in which a set of students (S) can be enrolled in. Each set C_{course} is composed by n disjoint subsets of classes ($C^{1,...,n}_{course} \subseteq C_{course}$) of different types (*e.g.* theoretical, practical, laboratory). A student enrolled in the course $course$ must attend exactly one class of each set C^i_{course}. Each subset of classes $C^i_{course} \subseteq C_{course}$, where $1 \leq i \leq n$, has a value $overlap_{C^i_{course}}$ associated, with $0 < overlap_{C^i_{course}} \leq |C^i_{course}|$, where $overlap_{C^i_{course}}$ represents the number of classes of C^i_{course} that can be overlapped. In other words, $overlap_{C^i_{course}}$ represents the smallest number of teachers[2] in charge of classes in C^i_{course}.

Furthermore, a class $c \in C$ is characterized by: a set of enrolled students ($S_c \subseteq S$) of size std_c; a set of suitable rooms ($R_c \subseteq R$); a set of suitable days ($D_c \subseteq D$); a set of suitable time slots ($T_c \subseteq T$); and its duration ($len_c > 1$). In this case, we consider that each individual class consists of a single meeting. Furthermore, we consider that all classes are taught in all weeks of a semester.

Each student $s \in S$ has a set of courses $Course_s \subseteq Course$ where they are enrolled in. Since all weeks have basically the same classes, one can generate the timetable for one week and generalize for the weeks of the whole semester.

Finally, each class has to be assigned to a suitable room (R_c). Each room r, with $r \in R$, has an ideal capacity, $cap_r > 0$. To ensure a fair distribution for students, we add a slack variable of α, representing the percentage of students enrolled in a class for which it is acceptable to be over the ideal capacity of the assigned room (*i.e.* overbooking).

The concepts described in this section can be converted into decision variables. However, the definition of these variables depends on the model (discussed further on).

Furthermore, the problem is subjected to a set of constraints ($constraint_c$ is the set of constraints relating to class c) that can be divided into hard or soft constraints. The constraints are defined as follows:

1. *Classes to time slots*: All classes must be assigned to the corresponding time slots.

[2]At IST, the assignment of teachers to classes is only performed after the schedule is created. Therefore, this number is computed based on the timetables from the previous year.

2. *Consecutive time*: A class must be taught in consecutive time slots on the same day.

3. *Class to rooms*: All classes that require room assignment ($R_c \neq \emptyset$) must be assigned to exactly one room.

4. *Student's conflicts*: All students must be able to attend the classes for which they are enrolled. This way, one can avoid some complex curricular rules since the students represent a specific path in the curricular plan. Otherwise, for each course, it would be required to encode multiple combinations of classes that a student can attend.

5. *Room conflicts*: A room can have at most one class scheduled per time slot per day.

6. *Teacher conflicts*: At most $overlap_{C^i_{course}}$ classes from the same course can be taught at the same time.

7. *Capacity*: Ensures that, in the worst case, only α% of the students enrolled may not be seated.

8. *Invalid Room* (r, c): The assignment of the class c to the room r is invalid.

9. *Invalid Time* (d, t, c): The assignment of the class c to the slot t in day d is invalid.

10. *Overlap* $(c1, c2)$: The classes $c1$ and $c2$ have to be assigned to the same time slot of the same day.

11. *Remove room* (r): The room r cannot be used. This constraint can be easily generalized to remove room r for short periods of time.

These constraints were provided by domain experts of the IST academic services. Some of these constraints have already been studied in the literature [59, 67, 70]. Furthermore, many universities consider a part of these constraints as soft. For example, *Invalid Time* has been in the past considered as a soft constraint [59, 67, 70].

In addition to the different constraints, one needs to define an optimization objective. The optimization objective used should depend on the timing of the disruption. If it occurs before the start of the semester, one may want to take advantage of this disruption to improve the solution further, even if it causes more perturbations in the original solution. On the other hand, if the disruption occurs during the semester, one wants to minimize the possible impact.

Consider four different optimization objectives listed below. The proposed approach assumes a lexicographic order between the different optimization criteria. This way, one can first optimize the number of perturbations and then improve the solution's quality (in tie-break scenarios).:

- *HD*: The goal is to minimize the number of classes that have different assignments (both room and time assignments). Domain independent metrics, such as this one, are not fair or equitable for all actors involved [44].

- *Weighted Hamming distance (WHD)*: WHD [44] is a weighted version of the Hamming distance. This metric allows us to compute the number of students affected by the perturbations.

Therefore, the goal is to minimize the impact of the perturbations in the student's academic performance. This objective is applied at IST to evaluate the new timetable when disruptions occur during the semester.

- *Compact room's timetable (RCOM)*: The goal is to optimize room usage by minimizing the number of gaps in a room's timetable. There are many different possible metrics that one can consider (see section 4.3.2).
- *Compact student's timetable (SCOM)*: The goal is to minimize the number of transitions from free to occupied (gaps) in a student's timetable. This objective is applied by hand at IST to evaluate the new timetable at the beginning of the semester. In the past, a similar metric has been proposed [79, 93]. However, our metric refers to the student's timetable and not a predefined curriculum.

Example 23. Let us consider again Example 21 shown in Figure 3.2a, with the disruptions: (i) the classroom R3 is closed and (ii) the class A^L cannot be overlapped. These disruptions cause the solution to be infeasible. As explained above, the optimal solution to MPP has value 2, if one applies the Hamming distance. However, this may not be the best approach. When generating the new timetable at the beginning of the semester, one can use the disruption to improve the overall quality. At IST, the academic services try to use the disruptions to improve the value of SCOM.

There is more than one optimal solution to this problem when considering the Hamming distance. The solution shown in Figure 3.2b is one. Another possible solution is to change A^L from Tuesday 10:30 to Tuesday 11:00. This allows the schedule for a student to be improved, as it reduces the number of gaps. This solution is shown in Figure 4.1a.

However, the timetable still has room for improvement. If one assumes that the group of students that attend D^{PB} and A^L is disjoint, one can overlap these classes, which would allow the course of A^L to be always preceded by A^T. This would also reduce the time students must spend at university. On the other hand, this solution reduces the number of choices a student can make. This solution is shown in Figure 4.1b and requires more perturbations.

4.3 Room Usage Optimization

The well-known phrase "space, like time, is money" shows the importance of optimizing the usage of rooms. The lack of space is one of the major timetabling challenges in IST. Therefore, we start by addressing this problem.

Example 24. Let us consider the timetables for two rooms—referred simply as Room 1 and Room 2—depicted in Figure 4.2. For the sake of simplicity, we assume that both rooms have the same capacity; the events shown in the timetables correspond to classes of different courses, and their names are

Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30					
8:30-9:00			B^{PB}		
9:00-9:30			(R2)		
9:30-10:00	A^{T}			A^{T}	B^{T}
10:00-10:30	(R1)		B^{T}	(R1)	(R1)
10:30-11:00			(R1)		C^{T}
11:00-11:30	A^{L}		D^{T}	D^{PB}	(R1)
11:30-12:00	(R4)	A^{L}	(R1)	(R2)	D^{T}
12:00-12:30	C^{T}	(R4)	C^{T}		(R1)
12:30-13:00	(R1)		(R1)	D^{PB}	
13:00-13:30	D^{T}	C^{PB}		(R2)	B^{PB}
13:30-14:00	(R1)	(R2)			(R2)

(a) Optimization Objectives: HD and SCOM.

Time/Weekdays	Monday		Tuesday	Wednesday	Thursday		Friday
8:00-8:30							
8:30-9:00				B^{PB}			
9:00-9:30				(R2)			
9:30-10:00	A^{T}				A^{T}		B^{T}
10:00-10:30	(R1)			B^{T}	(R1)		(R1)
10:30-11:00				(R1)			C^{T}
11:00-11:30	A^{L}	D^{PB}		D^{T}	D^{PB}	A^{L}	(R1)
11:30-12:00	(R4)	(R2)		(R1)	(R2)	(R4)	D^{T}
12:00-12:30	C^{T}			C^{T}			(R1)
12:30-13:00	(R1)			(R1)			
13:00-13:30	D^{T}		C^{PB}				B^{PB}
13:30-14:00	(R1)		(R2)				(R2)

(b) Optimization Objective: SCOM.

Figure 4.1: Two different timetables for a class of students after occurring two disruptions: (i) an unavailability constraint over the room R3; (ii) a no overlap constraint relating to the two lab class of A. The colors represent the different rooms were the classes are assigned.

immaterial for our discussion. The objective is to exchange events between these two rooms to reduce the time gaps between events in the same room without changing the existing timetable (*i.e.*, the time and duration of the existing events). We note that, by maintaining the timetable, any constraints involving the student's curriculum (not having overlaps between courses offered to the same students, for example) are not violated. Suppose a new event requires a room for the whole Wednesday afternoon; with the current set-up, it is impossible to allocate such a room. However, it is possible to move some events from Room 2 to Room 1 in order to ensure one free (available) room on Wednesday afternoon. Figure 4.3 shows one possible *tighter* timetable.

In this problem, we consider that classes have predefined time slots, and therefore, there is no need to consider curriculum-based constraints. The goal is to assign all classes to rooms. The predefined class schedule (day and time slots) is represented with an incidence matrix A. $A^{c}_{d_c,t}$ equals 1 if class c is scheduled in the time slot $t \in T_c$ of day d_c and 0 otherwise.

The Boolean variable $x_{c,r} \in \{0,1\}$ is equal to 1 if and only if the class $c \in C$ is assigned to the room $r \in R$, and 0 otherwise. A room is considered to be *occupied* in a time slot if and only if a class is assigned to that room in that time slot.

The $seated_{c,r}$ value corresponds to the number of students seated in case class c is assigned to room r. Note that $seated_{c,r}$ is given a value even if this class c is not assigned to room r. Formally:

$$seated_{c,r} = \begin{cases} students_c & \text{if } cap_r \geq students_c \\ cap_r & otherwise \end{cases} \tag{4.1}$$

4.3.1 Constraints

When assigning the class to rooms, the constraints considered are as follows:

- *Room Conflicts*: A room can have at most one class scheduled per time slot per day. Formally, for all $r \in R$, $d \in D$ and $t \in T$,

$$\sum_{c \in C} x_{c,r} \times A^{c}_{d,t} \leq 1. \tag{4.2}$$

Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30	MD (t)				
8:30-9:00			CDI (t)	CDI (t)	
9:00-9:30					CDI (t)
9:30-10:00	MO (t)	FIO (t)		Com (t)	
10:00-10:30					ESof (t)
10:30-11:00					
11:00-11:30	MD (t)	AMS (t)	MO (t)	AMS (t)	
11:30-12:00					
12:00-12:30					
12:30-13:00	CDI (t)				
13:00-13:30					
13:30-14:00					
14:00-14:30	ASA (t)			ASA (t)	
14:30-15:00		TCom (t)			EO (t)
15:00-15:30					
15:30-16:00		EO (t)		TCom (t)	
16:00-16:30					PEst (t)
16:30-17:00					
17:00-17:30	PEst (t)	IPM (t)		IPM (t)	
17:30-18:00					ACED (t)
18:00-18:30					
18:30-19:00	ACED (t)		ACED (t)		
19:00-19:30					
19:30-20:00					

(a) Room 1.

Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30			GRS (t)	CNVir (t)	
8:30-9:00					
9:00-9:30					
9:30-10:00		IAED (t)	MO (t)	GRS (t)	
10:00-10:30					MO (t)
10:30-11:00					
11:00-11:30		Micr (t)			
11:30-12:00				Micr (t)	
12:00-12:30					
12:30-13:00					
13:00-13:30				LD (t)	MC (t)
13:30-14:00		IRC (t)			
14:00-14:30			SDis (t)		
14:30-15:00		IRC (t)			EEC (t)
15:00-15:30					
15:30-16:00		AL (t)	CMU (t)		
16:00-16:30				SPO (t)	
16:30-17:00					
17:00-17:30			ACED (pb)		
17:30-18:00					
18:00-18:30					
18:30-19:00					
19:00-19:30					
19:30-20:00					

(b) Room 2.

Figure 4.2: Timetables for two rooms.

Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30	MD (t)				
8:30-9:00			CDI (t)	CDI (t)	
9:00-9:30					CDI (t)
9:30-10:00	MO (t)	FIO (t)	MO (t)	Com (t)	
10:00-10:30					ESof (t)
10:30-11:00					
11:00-11:30	MD (t)	AMS (t)	MO (t)	AMS (t)	
11:30-12:00					
12:00-12:30					
12:30-13:00	CDI (t)				
13:00-13:30					
13:30-14:00					
14:00-14:30	ASA (t)		SDis (t)	ASA (t)	
14:30-15:00		TCom (t)			EO (t)
15:00-15:30					
15:30-16:00		EO (t)	CMU (t)	TCom (t)	
16:00-16:30					PEst (t)
16:30-17:00					
17:00-17:30	PEst (t)	IPM (t)	ACED (pb)	IPM (t)	
17:30-18:00					ACED (t)
18:00-18:30					
18:30-19:00	ACED (t)		ACED (t)		
19:00-19:30					
19:30-20:00					

(a) Room 1.

Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30			GRS (t)	CNVir (t)	
8:30-9:00					
9:00-9:30					
9:30-10:00		IAED (t)		GRS (t)	
10:00-10:30					MO (t)
10:30-11:00					
11:00-11:30		Micr (t)			
11:30-12:00				Micr (t)	
12:00-12:30					
12:30-13:00					
13:00-13:30				LD (t)	MC (t)
13:30-14:00		IRC (t)			
14:00-14:30					
14:30-15:00		IRC (t)			EEC (t)
15:00-15:30					
15:30-16:00		AL (t)			
16:00-16:30				SPO (t)	
16:30-17:00					
17:00-17:30					
17:30-18:00					
18:00-18:30					
18:30-19:00					
19:00-19:30					
19:30-20:00					

(b) Room 2.

Figure 4.3: Reorganized timetables for the same two rooms, which now include a free room during Wednesday afternoon.

- *Capacity*: The number of attending students must be less than or equal to the ideal capacity of the room where the class is scheduled. Formally, for all $r \in R$,

$$students_c \times x_{c,r} \leq cap_r \forall_{c \in C, r \in R}. \tag{4.3}$$

Besides the constraints above, since the goal of this work is to optimize room usage, it is important to define metrics to correctly evaluate the usage of a room (RCOM), as detailed in the next section.

4.3.2 Compact Timetables: Metrics Definitions

Different metrics can be applied to evaluate the usage of a room, for example, in terms of the "compactness" of the corresponding occupation timetable. The simplest concept of compactness is, perhaps, the percentage of free space in a schedule. The percentage of free space can be computed as

$$\text{free-space} = \frac{1}{|R||D||T|} \sum_{c \in C} \sum_{t \in T} \sum_{d \in D} \sum_{r \in R} x_{c,r} \times A^c_{d,t},$$

where $|X|$ denotes the cardinality of set X. However, since the average of all free spaces is constant, as the number of occupied slots is also constant, this metric cannot be used.

One alternative is to calculate the variance in the amount of occupation across rooms. The variance can be computed by:

$$\text{Var} = \frac{\sum_{r \in R}(x_r - \mu)^2}{|R|},$$

where, x_r is the number of free time slots in room r and μ is the average number of free time slots.

The objective would then be to have high variance, as this would mean that some rooms are significantly more occupied than others, instead of a uniform occupation of space. However, this metric fails to grasp the difference between a very sparse timetable and a very compact one. A sparse timetable has many more transitions between vacancies and occupations than a compact one.

Such an observation suggests that the number of transitions — *i.e.*, the number of times a room changes status from vacant to occupied and vice versa — may be a useful metric. To this end, one can define the auxiliary Boolean variable $rcom_{c,d,t,r} \in \{0, 1\}$ which is equal to 1 if and only if the occupation of the room changes from `occupied` to `free` or vice versa, and 0 otherwise. And the goal would be to minimize the value of this auxiliary value, *i.e.*:

$$minimize \sum_{r \in R} \sum_{t \in T} \sum_{d \in D} \sum_{c \in C} rcom_{c,d,t,r} \tag{4.4}$$

However, this metric considers all holes in a timetable equally bad, which may not be true. For example, a hole corresponding to exactly one time slot (half an hour) is actually worse than a hole of four time slots (two hours) since no event can be scheduled in the former. Such an issue can be addressed by applying a weighted metric: the weight is inversely proportional to the vacancies' size, as it is easier to schedule new events in longer vacancies.

While the latter metric maybe, perhaps, the one that best captures our desired notion of "compactness", none of the metrics is actually perfect since none is better across all problems.

Example 25. Consider three rooms, r_1, r_2, r_3, with the same capacity ($cap_{r_1} = cap_{r_2} = cap_{r_3}$) and four classes ($c_1$ to c_4) with different durations. Further consider that we want to minimize the number of transitions as shown in statement 4.4. According to this metric, the two timetables shown in Fig-

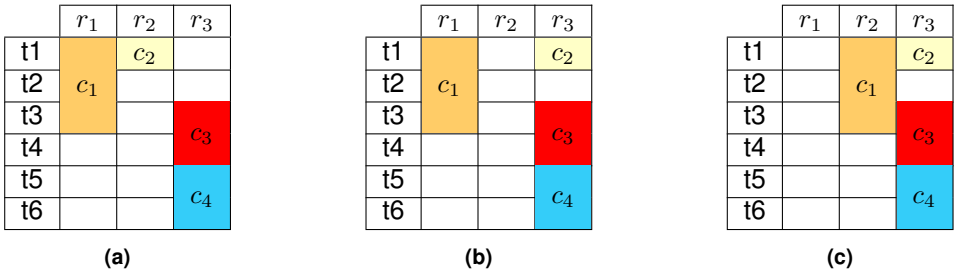


Figure 4.4: An optimal timetable (a), an optimal timetable with a free day (b) and an optimal timetable after closing down r_1 (c).

ures 4.4a and 4.4b are both optimal (with three transitions each). Apart from these, there are other optimal timetables that can be obtained. Consider now that a problem arises and the events in r_1 (*i.e.*, the event c_1) need to be assigned to a different room. One solution for this problem is shown in Figure 4.4c. If the timetable in Figure 4.4b is the "original" one, it requires fewer modifications to attain the solution. However, we can equally imagine other cases in which the timetable in Figure 4.4a is closer to the final configuration than Figure 4.4b. Finally, consider two new aperiodic events l_5 and l_6, with $sTime_{l_5} = t_2$, $eTime_{l_5} = t_5$, and $sTime_{l_6} = t_1$, $eTime_{l_6} = t_2$, as the respective starting and finishing times. If the original timetable is the one shown in Figure 4.4a, then no change to the previously allocated classes is required. However, if the original timetable is the one shown in Figure 4.4b, then class c_2 must change from room r_3 to r_2 before adding the new events. As it is impossible to predict future disruptions, it is not guaranteed that, for example, obtaining an empty day is always the best solution.

Example 25 shows that it is not possible to have a dominant optimization criterion. Each optimization criterion has its own advantages and disadvantages. Therefore, the optimization criterion should match the specific application. For this reason, we discussed with the stakeholders (IST academic offices) to choose which criteria to use.

4.3.3 ILP Formulation

An encoding using ILP, which was implemented to solve the problem of assigning classes to rooms. The ILP implementation is complete, which ensures the eventual discovery of the optimal solution to the problem. To solve this problem, a two-stage method is used where the goal is to: (i) maximize the number of seated students and (ii) minimize the number of transitions.

We use this decomposition into a two-stage approach to reduce the formula size and search space. This type of two-stage approach is common in integer programming approaches to solve university course timetabling problems [75, 93, 166]. The gains in performance of applying a two-stage approach versus a monolith approach are well documented. Vermuyten *et al.* [93] proposed a two-staged ILP approach to optimize students' flows in university course timetabling. The two-staged ILP approach reduced 170 times the execution time of the monolithic algorithm. The results of this work are no exception. Note that in this case, we are also forcing a lexicographic optimization.

First Stage: Maximizing the Number of Students Seated

$$\text{maximize:} \sum_{c \in C} \sum_{t_c \in T_c} \sum_{r \in R} seated_{c,r} \times x_{c,r} \times A^c_{d_c,t_c} \tag{4.5}$$

$$\text{subject to:} \sum_{r \in R} x_{c,r} = 1 \qquad \forall_{c \in C} \tag{4.6}$$

$$\sum_{c \in C} x_{c,r} \times A^c_{d,t} \leq 1 \qquad \forall_{r \in R, d \in D, t \in T} \tag{4.7}$$

$$cap_r \geq x_{c,r} \times (students_c - students_c \times \alpha) \qquad \forall_{r \in R, c \in C} \tag{4.8}$$

$$x_{c,r} \in \{0, 1\} \qquad \forall_{r \in R, c \in C} \tag{4.9}$$

In this section, we describe our first approach to solve stage (i) of the method.

As previously mentioned, the objective function 4.5 maximizes the number of students seated, where the value of $seated_{c,r}$ is defined in 4.1.

The problem needs two constraints to ensure the correct allocation of the classes. First, it is necessary to ensure that a class is allocated into exactly one room in the predefined time and day (Constraint 4.6).

Constraint 4.7 is required to ensure that there is no more than one class in each room in a specific slot. Finally, Constraint 4.8 ensures that, in the worst case, only a percentage α of the students enrolled cannot be seated. The reason for this constraint is explained further on.

Second Stage: Compactness

The second stage (ii) of the ILP implementation consists of a re-execution of the program with a different objective and adding a new constraint. Now, the goal is to compact the found timetable.

The old maximization statement is replaced by the Constraint 4.10 where BEST is the value obtained in the first stage. The new optimization statement 4.11 minimizes the number of time gaps in a room timetable (independently of the size of the time gap) by counting the number of transitions. The optimization statement 4.11 is not linear. However, it is easy to convert it to a linear formulation through the use of auxiliary variables and constraints as shown in [167]. The ILOG CPLEX solver [38] automatically transforms this type of statements.

$$\sum_{c \in C} \sum_{t_c \in T_c} \sum_{r \in R} seated_{c,r} \times x_{c,r} \times A^c_{d_c,t_c} = \text{BEST} \tag{4.10}$$

$$\text{minimize:} \sum_{r \in R} \sum_{d \in D} \sum_{t \in T} rcom_{c,d,t,r} \tag{4.11}$$

$$rcom_{c,d,t,r} = \begin{cases} 0 & \text{if } \sum_{c \in C} x_{c,r} \times A^c_{d,t+1} = \sum_{c \in C} x_{c,r} \times A^c_{d,t} \forall_{c \in C, t \in T, r \in R, d \in D} \\ 1 & \text{otherwise} \end{cases} \tag{4.12}$$

$rcom_{c,d,t,r}$ is a Boolean variable that represents the number of transitions of the a room. Therefore, we check if there is any transition occupied to free (vice-versa) all pairs of two sequential time slots.

4.3.4 Greedy Approaches

In this section, the proposed two greedy algorithms are described. Greedy heuristics [23], such as Greedy Randomized Adaptive Search Procedure (GRASP), have successfully been applied to course timetabling [90–92] and examination timetabling [55, 56]. The greedy algorithm is guided through a cost function that sums all the constraints violated. This method does not ensure any type of confidence in terms of finding an optimal solution. Furthermore, the optimization objective of this algorithm never considers room optimization.

Greedy Algorithm

The algorithm is guided through a cost function. Algorithm 1 shows the general idea behind the greedy search implemented. The algorithm ends when all classes have been allocated. The algorithm chooses the class with the smallest cost to allocate at each iteration (lines 7-14). At the end of each iteration, a class is assigned, and that information is stored in the set $allocated$ (line 17) to ensure that the same class is not going to be re-assigned in the next iteration (line 8). The cost function originally considered to evaluate a timetable Q for room r was:

$$conflict(Q_r) \times w + transition(Q_r) + idealCAP(Q_r) \tag{4.13}$$

where the function $conflict$ returns the number of overlapping classes (Constraint 4.2); the function $transition$ returns the number of transitions (computed as shown in Constraint 4.12) and the function $idealCAP$ returns the number of students above the ideal capacity in the allocated class. The value of the weight w has to satisfy the following constraint:

$$w \geq 24 \times 5 + \sum_{c \in C} len_c \times students_c. \tag{4.14}$$

This constraint ensures leaving a class without a room is worse than the level of compactness ($24 * 5$ is the worse case in terms of transitions) and having students exceeding the ideal capacity. The idea was to guide the search to compact timetables. However, this cost function does not have any type of optimality guarantees. Furthermore, the complexity causes the solution to be worse in the most important aspects: the number of conflicts and the number of students exceeding the capacity. Therefore, the objective of compactness was only used in cases where the number of students seated in two different classes was the same. This change requires modifying Algorithm 1, by the addition of a new *if* for the case where $cosT = cost$, this way ensuring that the compaction process only happens when there are two classes with the same cost. The objective is to have a timetable where all students can have a seat in the classes. Moreover, it is desirable to guarantee the quality of the solution as discussed further on.

To produce some guarantees in terms of optimality, another cost function was used.

$$\sum_{c \in C} \sum_{t \in T_c} \sum_{r \in R} (students_c - seated_{c,r}) \times x_{c,r} \times A^c_{d_c,t} \tag{4.15}$$

Algorithm 1: Greedy Algorithm based on a cost function

Input: A set of rooms R, a set of classes L
Output: A list of assignments to rooms Q

```
▷ Q is composed by pairs (c, r), c ∈ C and r ∈ R ;
alloc ← 0 ▷ number of allocated classes;
allocated ← ∅ ▷ set of allocated;
while |C| > alloc do
    cost ← MAX_VALUE;
    class ← −1 ▷ Class ID;
    room ← −1 ▷ Room ID;
    foreach c ∈ C do
        if l ∉ allocated then
            foreach r ∈ R do
                costT ← cost(Q_r, l);
                if costT < cost then
                    cost ← costT;
                    class ← c;
                    room ← r;
                end
            end
        end
    end
    if class ≠ −1 then
        alloc ← alloc + 1;
        Q ← (c, r);
        allocated ← allocated ∪ {c};
    end
    break ▷ unable to allocate all classes
end
```

This cost expression is simply based on the number of students that could not be seated in this assignment. Algorithm 1 does not allow conflicting assignments, and therefore they are not described in the cost function. When considering the slack α, the algorithm also does not allow the allocation of classes to rooms for which Constraint 4.8 is violated.

Theoretical implications It is possible to compute the approximated value of the optimal solution using a greedy algorithm if the benefit function is sub-modular, positive and monotone. The benefit function describes the advantages of performing an action.

Let us consider Γ as a finite set of variables. A function f is considered monotone if and only if it has the following property:

$$\forall_{\Xi \subseteq \Gamma} f(\Xi) \leqslant f(\Gamma) \tag{4.16}$$

In order to be considered sub-modular [168, 169], the function f must have the following property:

$$\forall_{\Xi,\Upsilon \subset \Gamma, \gamma \in \Gamma \setminus \Upsilon} f(\Xi \cup \{\gamma\}) - f(\Xi) \geqslant f(\Upsilon \cup \{\gamma\}) - f(\Upsilon) \tag{4.17}$$

Informally, one can think of a sub-modularity benefit function as a function where adding an element to the smallest set has a benefit greater than adding it to the biggest set. In order words, the benefit of adding elements to the same set reduces (or stays the same) every time we add a new element.

When one considers a greedy algorithm with a monotone, positive, sub-modular function, the resulting solution is approximated within $1 - e^{-1}$ to the optimal solution [170].

It is possible to define a benefit function which is sub-modular and monotone. When using the cost expression 4.15, the implicit benefit function f is basically the number of students seated per assignment. Formally, the benefit of assigning class c is $f(c) = seated_{c,r_c} \times x_{c,r_c} \times A^c_{d_c,t}$, where r_c is the room where the class was assigned. Assigning a class to a room is always positive, as the total number of students with seats increases. Differently, when using function 4.13, the benefit can be negative since the number of transitions and conflicts are considered. Not considering conflicts *i.e.*, only assigning the class to a room if no conflict exists makes function 4.15 monotone since it has the property 4.16.

Function 4.15 is sub-modular[168] since in the best case scenario, the benefit will always be the same (all students of the class can be seated in the room). There is no room large enough in the worst case so that the benefit will be smaller. Recall property 4.17, where Γ is the set of unassigned classes and $\Xi, \Upsilon \subset \Gamma$. The way the search is performed, the smaller set Ξ will always have a larger number of rooms available than Γ. When adding γ to Ξ, the algorithm will choose the room that maximizes the number of students seated, which will be at least as good as the assignment to Υ (as Υ has a smaller number of rooms available but all rooms available in Υ are also available in Ξ). This way, we can see that property 4.17 is ensured.

To sum up, the function 4.15 was constructed to be a positive, monotone, and sub-modular function. Therefore, the solution will be within 63% of the optimal, in terms of students seated [170].

4.3.5 Greedy Randomized Adaptive Search Procedure

The GRASP shown in Algorithm 2 has two main stages: (i) a random greedy algorithm (line 3) and (ii) a local search (line 4). As suggested in [55], we used two stages to optimize different objectives similar to the process used in our ILP approach. The ending condition of the algorithm is either the number of iterations (line 2) or the time limit (which ever comes first). At the end of each iteration, the algorithm stores the new solution if and only if it is better than the one found before (line 5).

The Random Greedy Procedure

The goal of the greedy algorithm is to maximize the number of students seated, and it is guided by the cost function expressed in 4.15. The degree of randomness can be customized by changing the size of the Restricted Candidate List (RCL). The Restricted Candidate List (RCL) is filled with classes which have the best cost at the time. The allocations are chosen randomly from the RCL. The $size$ 0 represents a greedy algorithm in its pure form (without randomness), and 1 represents a completely

Algorithm 2: GRASP Algorithm

Input: Maximum number of iterations (max_it), size of RCL ($size$), a set of classes C and a set of rooms R.
Output: Best solution ($best$).

```
1 best ← ∅ ▷ current best solution ;
2 for i = 0, ..., max_it do
3 |   solution ← greedyRandom(size, C, R);
4 |   solution ← Local(solution);
5 |   best ← UpdateSolution(solution, best);
6 end
```

random algorithm. The process ends when a complete solution is found, which does not violate any hard constraints.

Local Search

The goal of the local search is to minimize the number of transitions without deteriorating the quality of the solution in terms of students seated. This is achieved by swapping classes between rooms where the number of students seated increases or stays constant. Swapping can never deteriorate the quality of the current solution. This stage ends when no more possible swaps can take place without deteriorating the quality of the solution.

4.4 Course Timetabling

In this section, we present two different models to solve the university course timetabling problem described above. Table 4.4 summarizes the constraints used in the different models and the encoding for the most common disruptions. Table 4.5 summarizes the optimization statements used in the different models. An in-depth description of the models is presented below. Note that Gurobi 8.1 does not support a *not equals* constraint (*e.g.* $z \neq y$). Recall that there is a work around for discrete integer variables [27]. We introduce two new binary auxiliary variables (b_1, b_2) as indicators for $y \leq z - 1$ and $y \geq z + 1$, respectively. Finally, we only need to guarantee that $b_1 + b_2 = 1$. Example 6 shows how this process can occur in practice.

Let us consider an arbitrary decision variable $\mathcal{Y}$, represented by y in the original problem and by $\overline{y}$ in the MPP. The HD optimization objective counts the number of decision variables that changed value, *i.e.* $y \neq \overline{y}$, independently of their meaning and domain. Auxiliary variables are not considered in the optimization objective since it would add a bias to the result. The WHD optimization objective adds a weight (the number of students affected by the change in a decision variable) to HD. Finally, SCOM counts the number of gaps in a student's timetable. This value is computed through the usage of auxiliary variables (described below).

4.4.1 BOOLEAN Model

In the past, different integer programming models have been proposed to solve university course timetabling problems [47, 171]. These models typically use Boolean variables to indicate the assign-

Table 4.4: Constraints in the BOOLEAN and MIXED models.

	BOOLEAN	MIXED
1.	$\forall_{c \in C} \sum_{d \in, D_c} \sum_{t \in, T_c} a^c_{d,t} = len_c$	$\forall_{c \in C}\ a^c \geq 0$
2.	$\forall_{c \in C, d \in D, t, t_1, t_2 \in T, t < t_1 < t_2}$ $a^c_{d,t_1} = 1$ iff $a^c_{d,t} + a^c_{d,t_2} = 2$	$\forall_{c \in C}$ $a^c + len_c \leq \lfloor \frac{a^c}{\lvert T \rvert} + 1 \rfloor \times \lvert T \rvert$
3.	$\forall_{c \in C} \sum_{r \in R_c} x_{c,r} = 1$	
4.	$\forall_{s \in S, d \in D, t \in T}$ $\sum_{c \in C_s} a^c_{d,t} \leq 1$	$\forall_{c_1 \in c_s, c_2 \in C_s, s \in S}$ $v_{c_1,c_2} + v_{c_2,c_1} \geq 1$
5.	$\forall_{r \in R, d \in D, t \in T}$ $\sum_{c \in C} x_{c,r} \times a^c_{d,t} \leq 1$	$\forall_{c_1 \in C, c_2 \in C}$ $s_{l_1,l_2} \times (v_{c_1,c_2} + v_{c_2,c_1}) \leq s_{c_1,c_2}$
6.	$\forall_{d \in D, t \in T, c \in C}$ $\sum_{C^i_c \in C_c} \sum_{c \in C^i_c} a^c_{d,t} \leq overlap_{C^i_{course}}$	$\forall_{c \in C}$ $\sum_{C^i_c \in C_c} \sum_{c_1, c_2 \in C^i_c} o_{c_1,c_2} \leq overlap_{C^i_{course}}$
7.	$\forall_{c \in C, r \in R}\ (std_c - std_c \times \alpha) \times x_{c,r} \leq cap_r$	
8.	$x_{c,r} = 0$	
9.	$a^c_{d,t} = 0$	$a^c\ != (d \times \lvert T \rvert) + t$
10.	$\forall_{d \in D, t \in T}\ a^{c_1}_{d,t} = a^{c_2}_{d,t}$	$a^{c_1} = a^{c_2}$
11.	$\forall_{c \in C}\ x_{c,r} = 0$	

Table 4.5: Minimization objective for the BOOLEAN and MIXED models.

	BOOLEAN	MIXED
HD	$\sum_{t \in T, d \in D, c \in C} a^c_{t,d} \neq \overline{a^c_{t,d}}+$ $\sum_{r \in R, c \in C} x_{c,r} \neq \overline{x_{c,r}}$	$\sum_{c \in C} a^c \neq \overline{a^c}$ + $\sum_{r \in R, c \in C} x_{c,r} \neq \overline{x_{c,r}}$
WHD	$\sum_{t \in T, d \in D, c \in C} std_c \times (a^c_{t,d} \neq \overline{a^c_{t,d}})+$ $\sum_{r \in R, c \in C} std_c \times (x_{c,r} \neq \overline{x_{c,r}})$	$\sum_{c \in C} std_c \times (a^c \neq \overline{a^c})$ + $\sum_{r \in R, c \in C} std_c \times (x_{c,r} \neq \overline{x_{c,r}})$
SCOM	$\sum_{s \in S} \sum_{d \in D} \sum_{t \in T} scom_{s,d,t}$	$\sum_{s \in S} \sum_{c \in S_c} scom_{s,c}$

ment of classes to rooms and time slots.

This model is an extension of the room usage optimization model proposed above. The main difference lies in the possibility of changing the time assigned to a class. Therefore, this model has only one Boolean decision variable representing the assignment of a class to a room. Note that A is a pre-defined matrix and thus we changed it (from now on) to a which represents a variable.

On the other hand, the models proposed by [171] and applied by [47] use a three-index Boolean variable to indicate the assignment of classes to rooms and time slots at the same time. However, the models are considerably simpler in other respects. For example, they consider all classes to have the same unitary duration. The usage of a three-index model for our case study is inefficient since it requires a large number of variables ($|P| \times |L| \times |R|$) of which most of them are always with the value 0. Our BOOLEAN model separates the room assignment from the time assignment, which reduces the number of unnecessary variables. This separation also allows us to consider the disruptions only

for the variables involved (as we will see in section 5.3.3).

Nevertheless, these models can be easily generalized to solve the problem at hand. However, this model requires quadratic constraints to encode the room conflict constraint. One can try to mitigate the cost of the quadratic constraints by linearizing these constraints with the use of auxiliary variables and constraints [27]. This way, we can avoid the multiplication of two decision variables. We denote this version of the model BOOLEAN'.

Decision Variables

The schedule of a class (day and time slots) is represented by an incidence matrix a, where $a^c_{d,t}$ equals 1 if and only if a class c is scheduled in the time slot $t \in T_c$ of day $d \in D_c$. The assignment of a class to a room is represented by the Boolean variable $x_{c,r} \in \{0, 1\}$; it is equal to 1 if and only if the class $c \in C$ is assigned to room $r \in R$.

Auxiliary Variables

The variable $scom_{s,d,t}$ is equal to 1 if and only if the timetable of student s has a transition from occupied (attending a class) to free or vice versa.

Constraints

Table 4.4 summarizes the constraints used that can be explained as follows. Constraint 1 ensures that each class $c \in C$ is assigned to len_c time slots in $|D_c|$ days. Therefore, the variable $a^c_{d,t} \forall_{d \in D_c, t \in T_c}$ is assigned the value 1 exactly $|D_c| \times len_c$ times.

Constraint 2 ensures that the assignment of time slots for each class must be consecutive without gaps. For example, if a class is taught at $t_1 = 1$ and in time slot $t_2 = 5$ all slots in between $(2, 3, 4)$ must also be occupied by the same class. Note that the previous example is only feasible if and only if $len_c = 5$.

Constraint 3 ensures that each class $c \in C$ with $R_c \neq \emptyset$ must be assigned to exactly one room. Constraint 4 ensures that at most one class from a student's timetable in taught in each time slot of a day. Constraint 5 ensures that all rooms can have at most one class scheduled per time slot per day. Constraint 6 ensures that at most $overlap_{C^i_{course}}$ classes from the same course can be taught at the same time. Constraint 7 ensures that, in the worst case, only α% of the students enrolled may not be seated.

Finally, to encode disruptions in this model, one needs the following constraints. Constraint 8 ensures that it is impossible to assign class c to the room r. Constraint 9 ensures that class c cannot be assigned in the slot t of the day d. Constraint 10 ensures that two classes are assigned at the same time slots in the same days. Constraint 11 ensures that no class is assigned to room r.

4.4.2 MIXED Model

When creating mixed integer programming models for job-shop scheduling [172] it is common to define an integer variable to define the start time of a job on a machine. One can propose an

INTEGER model with only one integer variable to define the start period ($|P|$) of a class in a room. An INTEGER model would require fewer variables than the BOOLEAN model. However, it would also cause unnecessary constraints since some constraints do not depend on both class-room and class-time assignments. Moreover, it makes the application of decomposition techniques impossible [94].

For this reason, we propose the MIXED model that uses a Boolean variable for the assignment of rooms and an integer variable for the schedule.

The global number of variables of the MIXED model is smaller when compared to the BOOLEAN model since it only needs an integer variable for each class instead of a Boolean variable for each time slot. This model removes some constraints from the BOOLEAN model. The implementation ensures that the constraint of *room conflicts* is not quadratic through the use of auxiliary variables.

Decision Variables

The starting period of a class c is represented by an integer variable $a^c \in [1, \ldots, |P|]$. The assignment of a class to a room is described by the Boolean variable $x_{c,r} \in \{0, 1\}$. It is equal to 1 if and only if class $c \in C$ is assigned to room $r \in R$.

Auxiliary Variables

Additionally, we need the following variables:

- v_{c_1,c_2} is equal to 1 if and only if c_1 is taught before c_2;
- o_{c_1,c_2} is equal to 1 if and only if c_1 is taught at the same time as c_2;
- s_{c_1,c_2} is equal to 1 if and only if c_1 is taught in the same room than c_2;
- $scom_{c,s} \in \{0, 1, 2\}$ represents the number of transitions from free to occupied in the timetable of student s before/after class c.

Constraints

Table 4.4 summarizes the constraints used that can be explained as follows. First, one needs to ensure that all classes have a valid start period (Constraint 1).

In the MIXED model, the usage of an integer variable for the schedule ensures the class is taught in consecutive time slots. However, it is necessary to ensure all slots of the same class are taught on the same day (Constraint 2). Therefore, the last time slot of the class has to be taught until the last time slot of the day. The first free time slot after the end of the class c is $a^c + len_c$. The expression $\lfloor \frac{a^c}{|T|} + 1 \rfloor$ determines the day of the week from the period. Now, if one multiplies the day with the number of time slots per day ($|T|$), one obtains the first time slot of the next day.

Constraint 3 ensures that each class has exactly one room assigned (as explained above).

Constraint 4 ensures that all students can attend the classes in which they are enrolled. This constraint uses the auxiliary variable v_{c_1,c_2} to ensure that c_1 is taught before or after c_2. Therefore, the classes c_1 and c_2 can never overlap.

Constraint 5 deals with *Room conflicts*. Once again, we use the auxiliary variable v_{c_1,c_2} to ensure order between the classes. In this case, the constraint is only applied when the classes are assigned to the same room (s_{c_1,c_2}).

For Constraint 6, we use the auxiliary variable o_{c_1,c_2} which ensures that the two classes are taught at the same time. o_{c_1,c_2} can be defined by ensuring the result of adding v_{c_1,c_2} with v_{c_2,c_1} is zero.

Constraint 7 ensures that the room's capacity is respected (as explained above for the BOOLEAN model).

Finally, to encode disruptions in this model, one needs the following constraints. Constraint 8 ensures that it is impossible to assign class c to the room r.

For Constraint 9 one needs to convert the time slot $t \in T$ to periods by summing t with the result of multiplying the day d with the number of time slots per day. This way, one can ensure that class c cannot be assigned in the slot t of the day d.

The two last constraints can be easily modified to encode *preference time* and *room preference* disruptions. Constraint 10 ensures that the two classes have the same start period. Constraint 11 ensures that no class is assigned to room r.

4.5 Experimental Results

4.5.1 Experimental Setup

The solution was tested using the data sets from the courses taught at IST. We also have the current handmade solution that can be used as a starting point for the search algorithm.

The algorithms for room usage optimization are implemented in Java and are available from https://github.com/ADDALemos/Compacter. The programs using Pentaho Data Integration (PDI) software and Cleenex prototype, which were used to transform and clean the data, are available in https://github.com/ADDALemos/CleanDataPrograms. The data used to test the system was obtained through the FenixEdu™system public API[3] which is in use at IST.

The data is encoded with ITC-2019 XML format [58]. The code for solving the MPP and data sets are available at `https://github.com/ADDALemos/MPPTimetables/releases/tag/J.Scheduling19`. The program was implemented in C++, using the XML parser *RAPIDXML*[4] to read the timetabling.

The tests were run using the *runsolver* tool [173] with a time limit of 600 seconds and a limit of 3 GB of memory. The tests were performed on a computer running Ubuntu 14, with a 2.6 GHz CPU and 64 GB of RAM. The GRASP algorithm was executed with a maximum number of iterations of 1000 (the same as in [92]). The GRASP algorithm was tested with different sizes of RCL. The solution with the best results in terms of quality is the one with a RCL size of 60% of the total number of classes. This coincides with the value used in [92] for the complete university timetabling problem. The implementation was executed on a computer running *Ubuntu 14*, with a 2.6 GHz CPU at 2.6 GHz and 64 Gb of RAM. The ILP model was implemented using the library of ILOG CPLEX (version

[3] http://fenixedu.org/dev/api/

[4] *RAPIDXML* is available from `http://rapidxml.sourceforge.net/manual.html`

12.7.1.0) [38] and *Gurobi* [174]. The *Gurobi* solver was run with parameters *GRB_IntParam_Threads = 3* and *GRB_IntParam_Presolve = 0* for best performance.

4.5.2 Benchmark of IST

In this section, we examine the quality of the proposed models using the real-life instances described below. To characterize these data sets, it is important to define room *frequency* [175] and room *utilization* [175]. Room *frequency* denotes the ratio between the number of time slots used and the total number of time slots available. Room *utilization* denotes the ratio between the number of seats used and the total number of seats available considering all time slots on all days.

Table 4.6: Data sets characteristics.

	ITC 2007 [59]	KUL [93]	Alameda		Taguspark	
			1^{st} Semester	2^{en} Semester	1^{st} Semester	2^{en} Semester
# Instances	21	1	5	5	5	5
# Days	5	5	5	5	5	5
# Slots per day	5.38	6	26	26	26	26
# Students[a]	5 606.2	19 690	283 745	219 217	51 523	42 578
# Courses	89	UKN[b]	1022	1015	108	101
# Classes per Week	313.77	396	2111	1611	290	246
AVG Slots per Class	1	1	2.9	2.99	2.9	2.9
STDEV Slots per Class	0	0	0.5	0.52	0.5	0.5
AVG Enrollment per Class	61.67	49.7	77.45	77.22	63.24	58.6
STDEV Enrollment per Class	45.78	37	79.7	79.94	50.6	48.6
# Rooms	16.08	56	43	43	26	26
AVG Capacity by Room	114.71	UKN[c]	52.74	52.74	49.8	49.8
STDEV Capacity by Room	91.35	UKN	51.56	51.56	64.2	64.2
Frequency	76.42	13.2	49.36	39.56	28.49	27.4
Utilization	41.61	-	64.74	54.63	21.0	20.1

[a]This value includes double counting of the students since this number is the sum of all enrollments in all the classes.
[b]KUL generates the timetables post-enrollment. Therefore, the information about the courses is not necessary.
[c]It is considered that the rooms have sufficient capacity for the class.

Table 4.6 shows a comparison between the data sets present in the literature. One can observe that the data set from IST is considerably larger than the data sets from ITC and KUL. Furthermore, the data sets from ITC and KUL consider all classes with the same length, which can significantly simplify the encoding. Note that the data set from KUL does not specify different capacities for each room. For this reason, the *utilization* metric cannot be applied.

When analyzing the data set from Alameda campus of IST, one can see that room *utilization* is higher than room *frequency*. This can be easily explained by the need for classes with overbooking (discussed above).

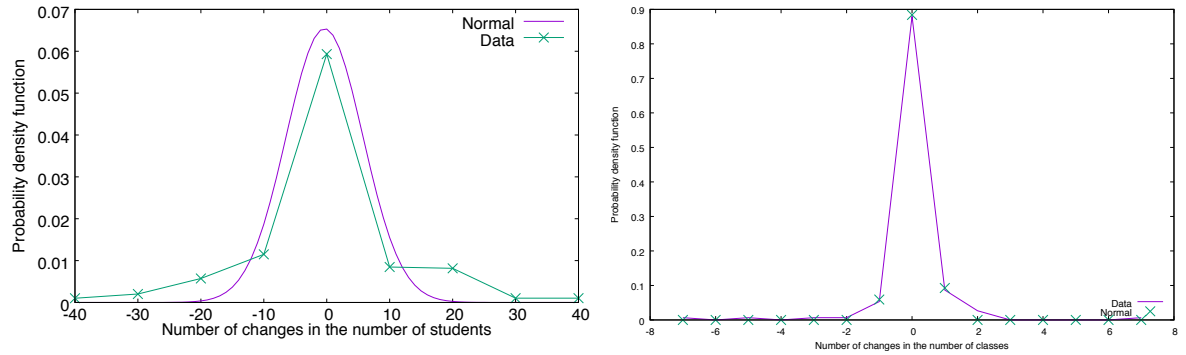
Recall that at IST, we consider that the rooms are available for use only on working days ($|D| = 5$) between 8am and 8pm, corresponding to a total of 12 hours and 24 slots ($|T| = 24$) of half an hour, corresponding to 120 periods ($|P| = 120$).

4.5.3 Generating Disruptions

To test our approach, we first analyzed the timetables from the last 5 years to identify which disruptions are the most common. These disruptions generally occur at the beginning of the new

Table 4.7: Average percentage of disruptions in the last 5 years.

Type of Disruption	Preference/Invalid Time Assignment	Preference/Invalid Room Assignment	Modify Enrolments Up (Down)	Overlap	Modify Number of Classes Insert (Remove)	Insert Curriculum
%	21	25	25 (27)	11	14 (8)	<1



(a) (b)

Figure 4.5: Normal distribution that best fits the data sets of fluctuations in the (a) students enrollments (with mean of -0.3 and standard deviation of 6), and (b) the number of classes (with mean of 0.04 and standard deviation of 0.45).

semester when a new timetable is generated. To compute the likelihood of a disruption to occur, we take into account the number of perturbations (Hamming distance) occurring from one year to another over the total number of variables.

The average percentage of disruptions by type is shown in Table 4.7. Note that *time* and *room assignment* represent the probability of a specific class to have a preference or a constraint in time or room assignment, respectively. We use these results to generate the number of disruptions of the same type randomly. We randomly generated the disruptions following a uniform distribution.

The information shown in Table 4.7 is not enough for generating the full set of disruptions. In some cases, it is also necessary to correctly quantify the perturbations in a given class/course. This is the case of two specific disruptions: *modify enrollments* and *modify the number of classes*.

For example, when *modifying enrollments*, we need to decide in which class the number of students changes (based on the value from Table 4.7) and the actual amount to increase/decrease. One could analyze the changes and uniformly select one of these values since these values are not equiprobable. Therefore, we used the Microsoft Excel Solver [176] to estimate the parameters of a normal distribution that would best fit our data sets. Figure 4.7a shows the comparison between the closest-fit normal distribution and our data set.

The same process can be applied to find the correct distribution for the disruption *modify number of classes*. Figure 4.7b shows the comparison between the closest-fit normal distribution and our data set.

4.5.4 Room Usage Optimization

In this section, we discuss the results of our different approaches to optimize room usage in IST.

Table 4.8: Comparison of greedy, ILP, handmade approaches in terms of global seated students. Result represents the best solution found by the algorithm for each data set. The optimal value was the one obtained by ILP. The CPU time for the decomposed problems corresponds to the sum of the CPU times of all sub-problems.

ILP							
		Decomposition	Time (sec)	Result (# students seated)	Difference (%)	Optimal	Total # students
Alameda	1^{st}	Week days	3 904.59	281 080	0	281 080	283 745
	2^{nd}	Week days	2 681.26	216 600	0	216 600	219 217
Taguspark	1^{st}	No	15	51 427	0	51 427	51 523
	2^{nd}	No	10	42 512	0	42 512	42 578
Greedy							
Alameda	1^{st}	No	1	277 422	1.3	281 080	283 745
	2^{nd}	No	1	213 731	1.32	216 600	219 217
Taguspark	1^{st}	No	1	50 698	1.41	51 427	51 523
	2^{nd}	No	1	42 024	1.14	42 512	42 578
GRASP							
Alameda	1^{st}	No	3 000	278 596	0.8	281 080	283 745
	2^{nd}	No	3 000	211 612	2.3	216 600	219 217
Alameda	1^{st}	No	6 000	278 596	0.8	281 080	515 23
	2^{nd}	No	6 000	214 050	1.1	216 600	42 578
Taguspark	1^{st}	No	2 387	51 213	0.6	51 427	51 523
	2^{nd}	No	1 439	42 341	0.4	42 512	42 578
handmade							
Alameda	1^{st}	N/A	N/A	268 668	4.41	281 080	283 745
	2^{nd}	N/A	N/A	210 958	2.6	216 600	219 217
Taguspark	1^{st}	N/A	N/A	51 006	0.81	51 427	51 523
	2^{nd}	N/A	N/A	42 286	0.53	42 512	42 578

Problem Decomposition

As expected, the application of two-stage integer linear programming allowed to reduce the total execution time (both stages), on average, by 15%. Thus, in the best case, we are able to remove almost a day of execution time. However, the results are not as good as some of the values reported in the literature [93]. This can be explained by the fact that our model does not change that much between stages. In other words, the constraints and variables are similar between stages in our approach than in some examples in the literature.

To reduce the size of the problem, one can separate the problem into sub-problems. In this case, the problem can be decomposed into five sub-problems (one per each day of the week). Decomposing the problem on weekdays does not remove any possible solutions since all classes are already scheduled. This separation is particularly interesting when the data set is large, and it is difficult to solve the data set as a whole. In this work, this technique was applied to solve the Alameda data sets when solving with the ILP approach. In the future, the solution can also be used to benefit from parallelization.

Number of Attending Students

Another issue to discuss is whether the constraint “the number of attending students must be smaller or equal to the capacity of the room” should be considered *hard* or *soft*.

Ideally, all students should be allowed to have a seat in the class in which they are enrolled. In practice, not all students will actually attend all the classes. Furthermore, it may be impossible (due to

space restrictions) to find rooms with sufficient capacity for all classes. In the literature, the constraint is usually considered *hard* [60, 83, 84] in particular when solving post-enrollment timetabling [60, 89] since the students choose the course they really want to attend. However, when solving curriculum-based timetabling [59, 67] it is common to consider the constraint as *soft*. At IST, as previously mentioned, both types of timetables exist. Undergrads usually have a specified curriculum to follow, but as they progress in the study, the more flexibility they have in the curriculum. Although ideally, this constraint should be considered as *hard*, independently of the type of timetable, in practice, this may be impossible. The ILP implementation shows that no feasible solution exists when considering the constraint as *hard*.

Table 4.8 shows the results for the different methods (greedy, ILP, GRASP and handmade) when maximizing the number of seated students.

The ILP implementation (4)-(9), maximizing the number of students seated, found the optimal value for all data sets. It is possible to see that it is inevitable to allocate some classes to rooms that do not have the required capacity. Table 4.8 shows in the column "Optimal", the optimal value in terms of the number of students seated for all data sets, and in column "Total # students" the total number of students that should be seated. With these results and the total number of students, it is possible to see that around 2% of the global number of students cannot be seated. This means that, with the current classes and rooms, it is impossible to seat all enrolled students for all classes. Note that to obtain the optimal values, it was necessary to use the decomposition discussed above (on weekdays) to avoid exceeding the memory limit for the Alameda data sets. The handmade timetables for this year leave more students without a seat. Table 4.8 shows the global number of students seated for the handmade solution (column Result) and distance to the optimal value in percentage (column Difference). The distance is computed based on the optimal value obtained by the ILP implementation.

The greedy algorithm using the cost function 4.15 was the only greedy algorithm that obtained a feasible solution. The CPU time of the algorithm for the complete data set is, on average, only 1 second (even for Alameda), making it unnecessary to apply the decomposition.

The solution obtained by the greedy algorithm has only 10% of the students without seats. The greedy algorithm is not able to find the optimal value. However, the solutions found are very similar as they are on average less than 2% from the optimal. The CPU time for the greedy algorithm is significantly shorter. This fact has particular importance when considering larger data sets. The greedy algorithm does not require any decomposition into sub-problems as it can solve the complete problem in less than 1 second. The greedy algorithm is two orders of magnitude faster than the ILP implementation (with the decomposition). This shows that ILP does not scale in terms of memory and time.

The GRASP algorithm lies in the middle of the results in terms of quality and CPU time. GRASP does not provide any type of quality assurance, unlike the greedy and ILP approaches. However, the quality of the solution does improve over time. The GRASP algorithm improves the quality of the solution but still falls short of the optimal. The randomness and the local search stage improves the

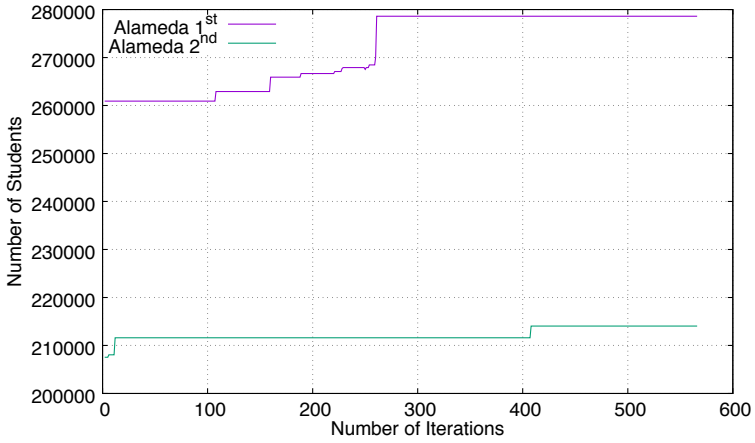


Figure 4.6: The evolution of the quality, in terms of students seated, of current best solution found by GRASP, for Alameda data sets

quality in terms of seating students.

When comparing with the simple greedy approach, GRASP improves only half percent in terms of the number of students seated. However, this improvement comes with a cost since the GRASP algorithm is worse in terms of CPU time. Note that both greedy algorithms optimize both objectives at the same time.

In the GRASP approach, most of the time is spent in the second stage of the algorithm. On average, the algorithm spends more than 80% of its CPU time on the local search stage. In all the Alameda instances, the time limit occurred before reaching the final number of iterations. However, in our case, the quality of the solution converges before the limit. Therefore, one can reduce the number of iterations without losing quality.

Figure 4.6 shows the evolution of the quality, in terms of students seated, of the current best solution for Alameda data sets as a function of the number of iterations. The values shown in Figure 4.6 represent the quality of the stored solution in a given iteration. The quality of Alameda 2^{nd} semester does not improve significantly over time. However, the quality of the original solution is closer to the optimal than for Alameda 1^{st} semester. This can be explained by the fact that this instance has a smaller number of courses and classes.

Furthermore, one can see, in Figure 4.6, that after 300 iterations, the solution quality is stable. Table 4.8 shows the quality of the results after 3 000 seconds and 6 000 seconds (time limit) for the Alameda data sets.

From these results, we can conclude that the constraint “the number of attending students must be smaller or equal to the capacity of the room” must be considered a *soft* constraint.

Slack on the Number of Attending Students

Maximizing the number of students seated may not be the best approach since all students are considered equally independently of the size and type of class. For example, it is worse not seating 3 students in a practical class with 20 students than not seating 3 students in a theoretical class with

Table 4.9: Minimal value of slack necessary to find a feasible solution for the Alameda and Taguspark data sets.

		Weekday				
		Monday (%)	Tuesday (%)	Wednesday (%)	Thursday (%)	Friday (%)
Alameda	1^{st}	35	27	27	27	27
	2^{nd}	35	35	35	35	12
Taguspark	1^{st}	33				
	2^{nd}	10				

100 students enrolled. For this reason, we added the Constraint 4.8 to the formulation to ensure a more even distribution of students which are above the ideal capacity.

Adding the Constraint 4.8 slack on the number of attending students does not compromise the CPU time of the first stage of the ILP implementation (number of seated students). Naturally, in some cases, the global quality of the solution may be worse (*i.e.* more students without seat). However, the distribution should be better since the percentage of students above the ideal capacity per room is smaller. Interestingly, only when solving the sub-problem for Monday on Alameda 1^{st} semester, the optimal was below the known optimal (only 1% worse).

α can never be below 27% in order to find feasible solutions in Alameda 1^{st} semester. However, in the 2^{nd} semester the value of α rises to a staggering 35%. For Taguspark, it is possible to impose a α of 10% in the 1^{st} semester. The results are summarized in Table 4.9. The lowest values of α were obtained by checking all the possible values until finding a feasible solution (starting with $\alpha = 0$). However, these results only show the lower value of α. A single overbooked class may cause the lower value.

Therefore, more detailed analyses of the results were performed. Figures 4.7a, 4.7b, 4.8a and 4.8b show the cumulative distribution of the number of slots with the number of students enrolled above the ideal capacity as a function of the percentage of students above the ideal capacity. From these figures, one can extract the number of time slots with at-most α of overbooking. It can be observed that as the percentage rises, the number of slots decreases. In other words, it is more common to have rooms with a small percentage of students above the ideal capacity. For example, in Figure 4.8a (Taguspark 1^{st} semester) when considering the ILP approach, there are only a small number of slots with the percentage of students above 30% of the ideal capacity. In fact, there are only 3 time slots (which represent only one class) in this situation. Ignoring this class, the α value could be lower than 5%. Interestingly, the number of slots with overbooking is higher in the 1^{st} semester in both campi. This result is expected since in both campi, in the 1^{st} semester, has a larger number of students enrolled.

To sum up, we have observed that high values of slack are caused by a small number of classes.

Overbooking

The results presented above clearly show that overbooking is a reality in these data sets. As such, the level of overbooking can be a metric to evaluate the quality of the solution. The comparison between the ILP and the handmade approaches are shown in Figures 4.7a, 4.7b, 4.8a and 4.8b for Alameda and Taguspark data sets. The handmade solution forces more rooms to be overbooked. For

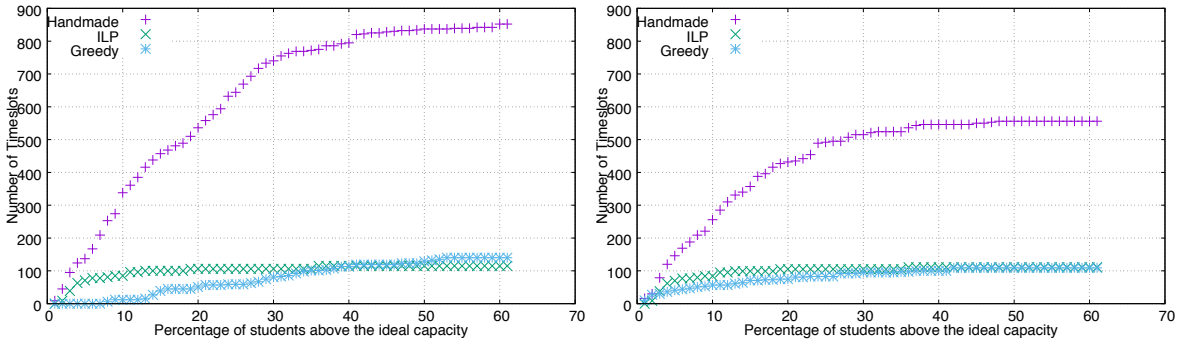


Figure 4.7: The cumulative distribution of slots with the number of students enrolled above the ideal capacity as a function of the percentage of students above the ideal capacity for: (a) Alameda 1^{nd} semester and (b) Alameda 2^{nd} semester.

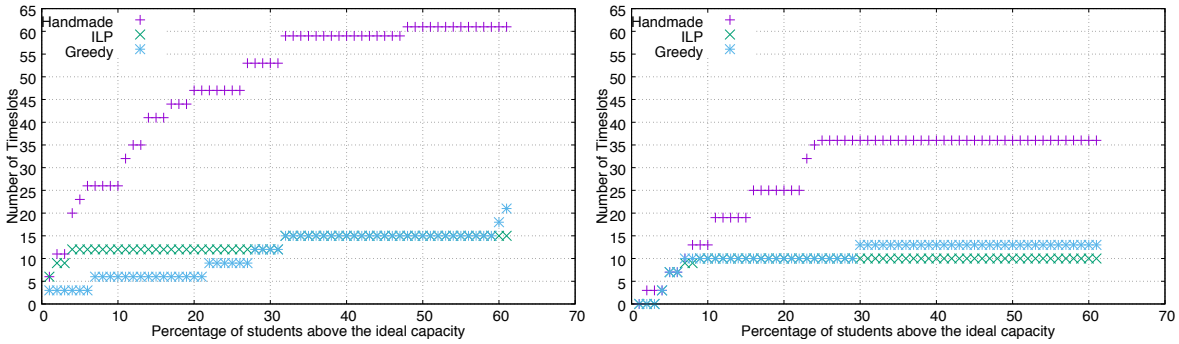


Figure 4.8: The cumulative distribution of slots with the number of students enrolled above the ideal capacity as a function of the percentage of students above the ideal capacity for: (a) Taguspark 1^{st} semester, (b) Taguspark 2^{st} semester.

example, in Figure 4.8a (Taguspark 1^{st} semester), there are 2 slots with 48% of students above the ideal capacity in the handmade solution. However, the solution found by the ILP implementation does not require any room to have these percentages. In the Alameda data sets, the ILP implementation found a solution with the highest percentage being 35% which is a clear improvement. The handmade solution has rooms with more than 50% of students above the ideal capacity. Furthermore, in Alameda 1^{st} semester, there are only 112 slots with overbooking in the ILP implementation, versus 852 slots with overbooking in the handmade solution (Figure 4.7a).

The greedy algorithm is, once again, a bit worse in terms of the quality of the results since it has rooms with a larger percentage of students above the ideal capacity. It is important to remember that the algorithm deals with this constraint as it deals with conflicts; it does not allow the assignment of classes to rooms above a certain threshold α. When comparing the results with the ones found by the ILP implementation, one can see that the threshold required to find a feasible solution is higher. However, the number of slots with overbooking is similar. When analyzing Figure 4.8b, corresponding to Taguspark 2^{nd} semester, one can see that the difference between Greedy and ILP approaches is small. In fact, the greedy approach has 13 time slots with overbooking versus only 10 in the ILP approach. This difference, in fact, corresponds to a single class. However, this class has 30% of

Table 4.10: The maximum number of seated students was obtained using the ILP approach, when pre-assigning the overbooked classes present in the handmade solution.

Campus	Working Day (Mon to Fri)	Semester			
		1^{st}		2^{nd}	
		Optimal	# Students	Optimal	# Students
Taguspark	All	49 038	49 555	40 231	40 511
Alameda	Monday	52 933	55 649	48 808	50 905
	Tuesday	55 509	59 228	42 609	44 457
	Wednesday	48 465	50 393	31 389	32 293
	Thursday	53 638	57 168	27 528	28 530
	Friday	40 657	42 957	44 978	47 339

students above the ideal capacity. On the other hand, in the data set of Alameda 2^{nd} semester (Figure 4.7b) the ILP implementation has 2 more time slots with overbooking than the greedy algorithm. This fact can be explained by an imposed lower value to α. The value of α for the ILP is 35%, which is smaller than the 42% used for the greedy algorithm.

When comparing the results of the greedy algorithm with the handmade solution, one can see that the improvement is significant in both the number of overbooked slots and the percentage of students who cannot be seated in those slots. For Alameda 1^{st} semester, the greedy algorithm only has 140 slots with overbooking versus 852 slots in the handmade solution. Moreover, the lower value of $alpha$ of the handmade solution is 60% versus 50% obtained by the greedy algorithm.

Some of the assignments that cause the highest number of students above the ideal capacity in the handmade solution could eventually be solved by simple reassignment of classes. However, it is possible that the assignment of certain classes to specific rooms without the proper capacity may be due to empirical knowledge; some teachers may justifiably prefer specific rooms, even though not all students can be seated. Furthermore, it could also mean that in practice, not all students will actually attend the class. Not considering this objective could cause students that actually attend classes not to be seated in order to seat students who may never attend. It should also be noted that the set of overbooked classes obtained by the ILP and greedy algorithms are actually a disjoint set of the set of overbooked classes found in the handmade solution. Thus, assuming there is a reason for assigning these rooms, we have forced these classes to be assigned to the preferred rooms. The addition of this objective does not change the CPU time. As expected, the overall number of unseated students is higher. When forcing these assignments, also as expected, the set of overbooked classes is the same for all algorithms. Even though they are now equal in terms of students seated, the ILP and greedy solutions are better in terms of compactness. Table 4.10 summarizes the number of seated students when allowing the ILP approach to keep the same room for the overbooked classes as in the handmade solution. These results are optimal, for each instance, and were obtained using the ILP approach.

Room Compactness Process

In terms of compactness, the greedy algorithm only performs the compaction procedure in tie-breaking scenarios. In other words, the compaction is only executed when allocating classes with

Table 4.11: Compaction results in terms of the number of transitions for the greedy algorithm, GRASP, ILP before and after the compactness optimization. The ILP finds the optimal solution to the Taguspark data sets within the time limit. The time spent just in compaction routine is also shown. The cells highlighted in gray represent values found through decomposition.

		handmade	Greedy	ILP			GRASP
		# Trans.	# Trans.	# Trans. before comp.	# Trans. after comp.	Extra Time (sec)	# Trans.
Alameda	1^{st}	2 148	1 242	2 068	937	8.4×10^5	1 118
	2^{nd}	1 945	1 030	2 001	685	4.2×10^5	944
Taguspark	1^{st}	383	279	369	217	1.2×10^2	238
	2^{nd}	330	241	307	193	3.8×10	210

Table 4.12: ILP compactness results in terms of number of transitions and CPU time (in days), for the Alameda data sets.

Weekday		Monday		Tuesday		Wednesday		Thursday		Friday	
Results		Time (d)	# Trans	Time (d)	# Trans	Time (d)	# Trans	Time (d)	# Trans	Time (d)	# Trans
Alameda	1	0.8	146	1.6	272	3.5	145	3	202	0.7	148
	2	0.5	155	0.7	148	2.5	123	1	122	0.4	137

the same gain in terms of seated students. Nevertheless, the results are an improvement when comparing them with the handmade solutions used in IST, even though the values found by the greedy algorithm are not optimal. Table 4.11 compares the values obtained by the ILP implementation, the greedy algorithm, and the handmade solution. The values represent the number of transitions from free to occupied slots and vice versa.

The ILP implementation finds the optimal solution for the Taguspark data sets although it requires significantly more CPU time. The number of transitions obtained is higher when using the greedy algorithm. Nevertheless, the results are close to the optimal found by the ILP implementation. The greedy algorithm finds a solution that is 1.2x worse than the optimal.

The ILP implementation was not able to compact all the allocations when considering the decomposed data sets from Alameda within a pre-defined time-frame. In the worst case, three days worth of CPU time were spent to find the optimal solution.

Overall, the number of transitions of the greedy implementation is 1.4x and 1.5x higher than the optimal for the Alameda 1^{st} semester and 2^{nd} semester, respectively. The number of days spent for each weekday for the Alameda data sets is shown in Table 4.12.

Once again, the quality of the solution found by the GRASP algorithm lies in the middle of the ILP and greedy approaches. Only the local search stage of GRASP optimizes the quality in terms of transitions. When comparing GRASP with the simple greedy algorithm, the improvement is more significant when considering the number of transitions. In this optimization objective, GRASP is able, on average, to reduce the distance to the optimal value by 12%. The summary of the results is shown in Table 4.11. Nevertheless, even with fewer iterations, the algorithm is considerably worse in terms of CPU time.

Figure 4.9 shows the evolution of the quality, in terms of transitions, of the current best solution for Alameda data sets as a function of the number of transitions. We conclude that the algorithm converges in more or less the same number of iterations for both optimization objectives (Figure 4.6). Considering this optimization objective, the first solution for the Alameda 2^{nd} semester has a larger distance to the optimal than when considering the number of students as an optimization objective.

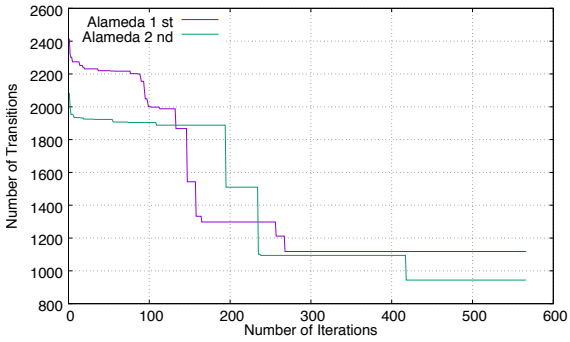


Figure 4.9: The evolution of the quality, in terms of transitions seated, of current best solution found by GRASP, for Alameda data sets

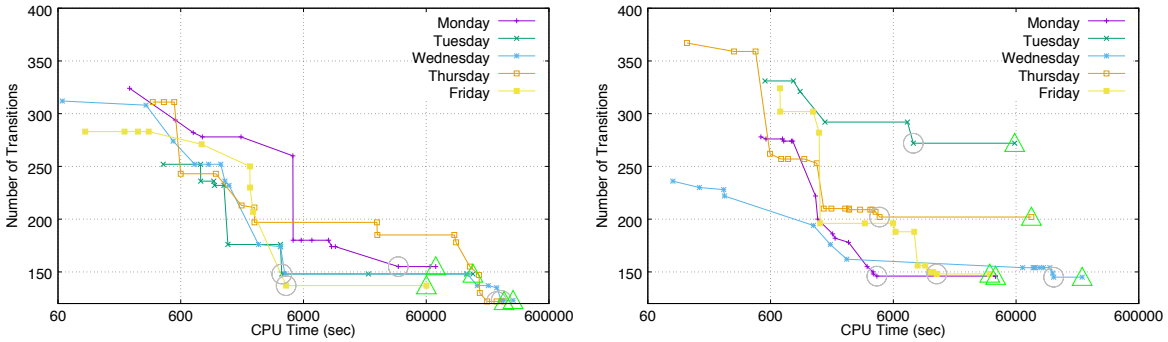


Figure 4.10: The evolution of the number of transitions over time in seconds (log scale), for the Alameda (a) 1^{st} and (b) 2^{nd} semester. The gray circle and the green triangle symbols mark the finding of an optimal value and the proving of optimality, respectively. The results were obtained by the execution of CPLEX with the default configurations.

Nevertheless, the Alameda 2^{nd} semester instance has a smaller number of transitions, which is normal since it has fewer classes.

Time limit for the ILP Room Compactness Process

The greatest weakness of the ILP implementation is the large CPU time required, in particular in the second stage. One approach to reducing the CPU time lies on warm-starting the second stage with the solution from the first stage. However, this process, on average, did not improve the CPU time. We believe that the improvement really depends on how close the solution found in the 1^{st} stage is to an optimal solution of the 2^{nd} stage. In this case, the solution, in most instances, for the 1^{st} stage is not close to an optimal solution for the 2^{nd} stage.

In the second stage, most of the time spent by the ILP solver is not improving the quality of the solution. The solver is actually proving the optimality. In practice, the trade-off between optimality and CPU time may sometimes lead to a shorter CPU time.

To this end, a study on the evolution of the quality of the solutions over time for the Alameda data sets (which are the most complex) was performed. Figures 4.10a and 4.10b show the evolution for each weekday for the 1^{st} and 2^{nd} semesters, respectively. Both graphs have a logarithmic scale and show the time in seconds. The large round gray symbols mark the reaching of the optimal solution

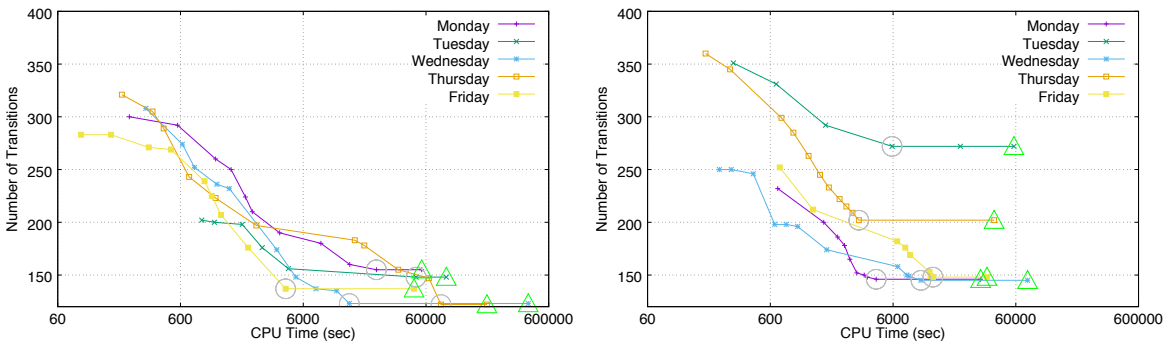


Figure 4.11: The evolution of the number of transitions over time in seconds (log scale), for the Alameda (a) 1^{st} and (b) 2^{nd} semester. The gray circle and the green triangle symbols mark the finding of an optimal value and the proving of optimality, respectively. The results were obtained by the execution of CPLEX configured to re-apply presolve with cuts and allow new root cuts.

(without proving it). The large green triangle symbols mark the time for proving the optimal solution. We conclude that, in most of the cases, the time spent after 6 000 seconds produces small changes in the quality of the solution. If we consider the 6 000 seconds as the time limit, we can see that for the 2^{nd} semester, most of the data sets already had their optimal value found, even though it was not yet proven.

As one can see, the evolution shown in Figures 4.10a and 4.10b is not smooth, but rather exhibits some *jumps*. These *jumps* correspond to a restart in the execution of the solver. Every time a *jump* occurs, the solution found is significantly improved. Therefore, one can try to produce more *jumps*. The ILP implementation was rerun with the `RepeatPresolve` flag with value 3 (re-presolve with cuts and allow new root cuts). The results for Alameda data sets are shown in Figures 4.11a and 4.11b. The optimal solution is always found faster with this configuration. Nevertheless, the solver still spends most of the CPU time proving the optimality. When solving most of the instances, the solver spends 50% or more time proving the optimality. In the worst case, for the data set of Thursday Alameda 2^{nd} semester, the CPU time spent on proving optimality is 92% of the total CPU time.

4.5.5 University Course Timetabling Problem

Table 4.13: Results for BOOLEAN, BOOLEAN' and MIXED models solving the university course timetabling problem with warm-start, considering general (Gen.) and quadratic (Qua.) constraints.

	$1^{st}semester$			$2^{nd}semester$		
	BOOLEAN	BOOLEAN'	MIXED	BOOLEAN	BOOLEAN'	MIXED
Time (s)	Time Out	Time Out	32.17	Time Out	Time Out	22.69
Optimal	Not Found	Not Found	Found	Not Found	Not Found	Found
# Variables	5 269 200	5 280 380	8 487 526	5 255 628	5 266 548	1 668 712
# Gen. Constraints	5 200 000	5 213 975	2 111 238	5 187 000	5 187 000	1 810 312
# Qua. Constraints	5 590	0	0	5 460	0	0
SCOM Handmade	2 148			1 945		
SCOM	937			685		

As one can see, the Taguspark instances are too simple. Therefore, from now on, we are going

to focus on the Alameda instances only. In the first approach, we compare the performance of the two different models when solving the original version of the problem, *i.e.* before disruptions occur. We considered the SCOM as the optimization objective since it is the one used at IST. Note that it is a different optimization objective from the ones used in the previous sections, and thus the execution time is not directly comparable. Furthermore, the solution found may not be optimal with this new objective. Nonetheless, we apply a warm-start procedure using this feasible solution to improve the performance of our approach. We obtained this solution with the room usage optimization procedure (see section 4.3). We also tested using greedy heuristics (the extension from section 4.3.4). However, they are worse in terms of quality and similar in terms of execution time. Therefore, we do not show the result.

Table 4.13 shows the results for the different models (BOOLEAN, BOOLEAN' and MIXED) when solving the university course timetabling problem with warm-start. Observe that the BOOLEAN model requires both general and quadratic constraints. The linearized version of the BOOLEAN model only requires general constraints. However, it adds two new auxiliary variables for each quadratic constraint. The BOOLEAN' model also adds two and half more constraints for each new variable. There is no clear advantage in the BOOLEAN model versus the BOOLEAN' model in this time limit. The MIXED model requires fewer constraints and variables. In terms of execution time, the BOOLEAN model is not able to find an optimal solution within the 600 seconds limit.

It is possible to reduce the size of the problem by reducing the number of students since not all students affect the solution. In this case, the students are used to identify the curricular path. It is natural that groups of students attend the same classes. Therefore, one does not need to generate constraints for all students. The constraints can be generated only for *distinct* students. In the data sets, around 15% of all students attend exactly the same classes. This number is relatively small but allows us to reduce the total number of constraints by 10% on average for each instance.

To study the impact of the warm-start strategy on the performance, we run the tests with and without this strategy for the MIXED model. The results from $1^{st}semester$ and $2^{nd}semester$ improve from 40 822.57 and 50 726.21 to 32.17 and 22.69, respectively. The version with warm-start is three orders of magnitude faster. The solution from the room usage optimization procedure is shown to be a good starting point. Recall that we are optimizing with a different optimization objective. The values without warm-start are similar to the results obtained with the room usage optimization procedure.

4.5.6 Minimal Perturbation Problem

The models proposed in this chapter were also tested in the presence of disruptions. For each disruption type, 50 different instances were created. In this chapter, we only show the results for the most relevant disruptions (*i.e.* the disruptions that are likely to occur). The instances were generated based on the data analyses explained above. We tested the models using all the disruptions found on the original data (see Table 4.7). In this section, we discuss the most relevant disruptions.

We warm-start the algorithm with a partial feasible solution extracted from the solution of the previous year. Once again and as expected, the MIXED model outperforms the BOOLEAN model. For

Table 4.14: Results for the most common disruptions using the MIXED model. δ_{HD} measures the number of perturbations, δ_{WHD} measures the number of students affected by the perturbations and δ_{SCOM} measures the change in the number of gaps in the student's timetable.

		Modify Enrollments		*Invalid Room*	
Opt.		$1^{st}semester$	$2^{nd}semester$	$1^{st}semester$	$2^{nd}semester$
SCOM	Avg. Time (s)	46.2	20.65	73.36	60.52
	Avg. δ_{SCOM}	-1	-1.5	0.5	0
	% Optimal	100	100	80	90
	Avg. δ_{HD}	510	399	372	340
	Avg. δ_{WHD}	27 303.9	27 287.1	27 000	27 602.5
HD	Avg. Time (s)	40.79	42.91	74.45	54.37
	Avg. δ_{SCOM}	0	0	4.4	0.87
	Avg. δ_{HD}	489	380	300	309.75
	% Optimal	100	100	80	90
	Avg. δ_{WHD}	26 803.5	26 990.2	26 660.83	27 334.7
WHD	Avg. Time (s)	39.97	43.6	69.17	60.28
	Avg. δ_{SCOM}	0	0	4.2	0.6
	Avg. δ_{HD}	508	399	490	489
	Avg. δ_{WHD}	25 908.5	26 512.9	26 442	27 468.12
	% Optimal	100	100	80	90
% of Feasible Solution		100	100	80	90

Table 4.15: Incremental approach to recover after disruptions of the type *invalid time* while optimizing δ_{SCOM}. δ_{HD} measures the number of perturbations, δ_{WHD} measures the number of students affected by the perturbations and δ_{SCOM} measure the change in the number of gaps in the student's timetable.

	$1^{st}semester$		$2^{nd}semester$	
	Stage 1	Stage 2	Stage 1	Stage 2
Avg Time (s)	239.02	68.67	210.68	58.4
Median Time (s)	282	68	231.24	58
Avg δ_{SCOM}	2	2	1	1
AVG δ_{HD}	416.67	416.67	400.34	400.34
AVG δ_{WHD}	26 726.34	26 726.34	27 473.34	27 473.34
% Feasible Solution	85		90	

this reason, only the results for the MIXED model are shown in Table 4.14, which shows the results for the most common disruptions for different optimization objectives. For each objective (SCOM, HD, WHD) we analyse the quality of the solution considering three objectives ($\delta_{SCOM}, \delta_{HD}, \delta_{WHD}$). Each objective - $\delta_{SCOM}, \delta_{HD}, \delta_{WHD}$ - measures the improvement in terms of the number of gaps in the student's timetable, the number of perturbations, and the number of students affected by those perturbations, respectively. Furthermore, the table shows the percentage of optimal solutions found for each optimization objective. Note that the experiments with *invalid time* disruption exceeded the time limit, and therefore, the results are not shown in this table. Naturally, the feasibility of an instance does not depend on the optimization objective. In order to improve performance, the warm-start strategy was modified only to add a partial solution (the variables affected by the disruption are not added). The unfeasible instances are unfeasible due to a subset of hard constraints. In the real world, these cases happen. The solution ends up by relaxing a few constraints. However, the relaxation is achieved through negotiation between all parties involved.

The $2^{nd}semester$ instance is easier to solve since it has a smaller number of classes to assign. This fact can be explained by the organization of the curricular plans.

When analyzing the performance, one can see that the *modify enrollment* disruption instances are the easiest to be solved. This can be explained by the fact the disruption adds, in general, only a small number of students. The use of slack in room capacity also contributes to better performance. This is the only disruption that is able to improve the value of the SCOM (when optimized for this objective). This disruption also allows reducing the number of students enrolled. *Modify enrollments* is the only disruption to have all instances with a feasible solution. The HD and WHD optimization objectives reduce the number of perturbations and the number of students affected while not improving the SCOM. For this disruption, the value of SCOM stays equal to the one found in the original solution.

The disruptions *invalid time assignment* and *invalid room assignment* cannot lead to a solution with an improved optimization value, given that these disruptions only add new constraints to the problem. The results show that the quality of the solution gets worse with these disruptions. Note that these disruptions may cause the instance to become infeasible. For all instances with feasible solutions, our approach finds optimal solutions. The *invalid time assignment* disruption is the only one to reach the time limit before finding a solution.

Globally, one can see that optimizing using SCOM causes more perturbations in the solution and affects more students. This result confirms the results obtained by [47], where it is said that performing more perturbations can improve the quality of the solution. Optimizing based on the WHD objectives also causes more perturbations but reduces the number of students affected by the change.

4.5.7 Incremental Approach for Recovery after Disruption

The simple recovery process takes too long for the *invalid time assignment* disruption. Therefore, we propose an incremental approach to reduce the search space by splitting the problem into two stages. A disruption of the type *invalid time assignment* clearly causes a change in the time slots of the class. However, it may not be necessary to modify the room assignment. For this reason, we divide the problem into: (i) the problem of assigning classes to time slots and (ii) the whole problem. In this case, the original solution for this sub-problem is added to the model as static. The second stage applies a warm-start with the results of the first stage, possibly improving its quality. This division does not exclude possible solutions since the second stage includes the whole problem again. This decomposition can be changed (*i.e.* start with the problem of assigning classes to rooms) depending on the disruption since we know beforehand which part of the problem must change.

Naturally, the proposed decomposition does not provide any guarantees about the optimality at the end of the first stage. Nevertheless, this decomposition ensures that a feasible solution to the first stage is a feasible solution to the whole problem. Conversely, one cannot conclude anything about the infeasibility of the instance based on the result from the first stage.

Table 4.15 shows the results for the *invalid time assignment* disruption, where one can see that the incremental process is much faster. In this case, the second stage does not improve the quality of the solution. In other words, the first stage not only found a feasible solution but also found an optimal solution to the complete problem.

Table 4.16: Comparison of our ILP approach with different methods from the state-of-the-art, in terms of the cost. The best values are highlighted in bold. Only 6 instances have no optimal solution found.

	MaxSAT [70]	ASP [177]	ASP - teaspoon [178]	Tabu [179]	Best known (Dec-19)	Ours
comp01	*TIMEOUT*	**5**	**5**	**5**	**5**	**5**
comp02	**24**	125	**24**	75	**24**	**24**
comp03	*TIMEOUT*	196	109	93	**64** (not optimal)	70
comp04	**35**	36	**35**	45	**35**	**35**
comp05	*TIMEOUT*	947	624	326	**284** (not optimal)	**284**
comp06	**27**	155	**27**	62	**27**	**27**
comp07	**6**	79	**6**	38	**6**	**6**
comp08	**37**	39	**37**	50	**37**	**37**
comp09	*TIMEOUT*	264	169	**119**	**96**	**96**
comp10	**4**	**4**	**4**	27	**4**	**4**
comp11	**0**	**0**	**0**	**0**	**0**	**0**
comp12	*TIMEOUT*	1114	456	358	**294** (not optimal)	302
comp13	62	112	**59**	77	**59**	**59**
comp14	**51**	52	**51**	59	**51**	**51**
comp15	*TIMEOUT*	196	109	87	**62** (not optimal)	70
comp16	**18**	28	**18**	47	**18**	**18**
comp17	**56**	171	**56**	86	**56**	**56**
comp18	*TIMEOUT*	184	81	**71**	**61**	**61**
comp19	58	91	**57**	74	**57**	**57**
comp20	**4**	80	**4**	54	**4**	**4**
comp21	**86**	232	124	117	**74**	**74**
DDS1	**48**	87	**48**	1024	**48**	**48**
DDS2	**0**	**0**	**0**	**0**	**0**	**0**
DDS3	**0**	**0**	**0**	**0**	**0**	**0**
DDS4	*TIMEOUT*	**26**	33	261	**17**	**17**
DDS5	**0**	**0**	**0**	**0**	**0**	**0**
DDS6	**0**	**0**	**0**	11	**0**	**0**
DDS7	**0**	**0**	**0**	**0**	**0**	**0**
test1	*TIMEOUT*	383	404	**234**	**224**	**224**
test2	**16**	31	**16**	17	**16**	**16**
test3	*TIMEOUT*	172	113	86	**67** (not optimal)	86
test4	*TIMEOUT*	232	156	132	**73** (not optimal)	90

4.5.8 International Timetabling Competition 2007

In order to validate the quality of the proposed approach, we also solve the benchmarks from the literature that are encoded in the format of ITC 2007 [59]. Table 4.16 shows a comparison of different state-of-the-art methods and our approach. One can see that our approach finds the optimal solution for 26 out of 32 instances. Our solution clearly outperforms, in terms of cost, most constraint programming-based approaches (SAT and ASP). However, it still falls short of the local search-based methods and the overall best-known results. Note that the best known result was obtained by multiple approaches. Furthermore, we extracted the best know value from the validation tool that was discontinued in 2019. The instances (6) for which our approach did not find the optimal solution within the time limit are the same instances for which there is no information about the best value.

4.5.9 Results Overview

In this chapter, we showed that integer programming is a good approach to solve timetabling problems with different characteristics and optimization functions.

Recall that one of the goals of this thesis is to re-use the search tree to reduce the execution time.

However, we showed that we could efficiently solve the MPP with ILP using only a good warm start function. The warm start also has a significant impact when solving the whole problem if we have a good initial solution. The impact is significant even if the warm-start solution was obtained with different optimization criteria. The warm start is able to reduce the execution time in three orders of magnitude.

To further improve the quality and performance of the ILP algorithm, we proposed an incremental algorithm. This algorithm aimed to decompose the problem and guide the changes to a specific part of the problem. This can be seen as a set of rules that are able to prune the search space. The concept of guiding the re-solving process can be improved by learning which courses are affected by the disruption. This concept is the key to the algorithm proposed in chapter 6 for the train scheduling problem.

The IST benchmark created in this thesis is considerably larger (*e.g.*, number of classes) and richer (in terms of the type of constraints). The ITC 2019 benchmark is even larger and richer (constraints) than IST. However, the fact that our approach is able to solve both the ITC 2007 and IST benchmark made us believe this approach could be a good starting point to tackle the International timetabling competition 2019.

4.6 Concluding Remarks

This chapter discusses the real-world problem of university course timetabling at IST. We propose different methods to optimize room usage and to recover after disruptions occur.

When focusing on the problem of room usage optimization for university timetables, we propose methods to optimize the room occupation by determining where to allocate events with pre-defined time slots. This optimization process must ensure that the rooms have enough capacity to seat all people participating in those events. First, we propose a two-stage ILP implementation to allocate classes to rooms while optimizing room usage. The first stage focuses on allocating classes to rooms in order to maximize the number of students seated. To ensure fairness, we added slack to the capacity of each room before maximizing the number of students seated. The second stage focuses on the minimization of the number of transitions from free to occupied (and vice versa) for each room. The ILP implementation finds the optimal solution for all data sets tested. However, it requires some decomposition in order to solve all data sets within the time and memory constraints. We have shown that it is possible to efficiently optimize the room usage and reduce the overbooking problem with integer programming.

When focusing on the problem of re-solving university course timetables after disruptions, we proposed two integer programming models that can find the *closest* new feasible solution. Our approaches were successfully evaluated with data sets from Instituto Superior Técnico. The models were tested using different optimization objectives, including the optimization objectives applied by the academic offices at Instituto Superior Técnico.

To improve the performance, we propose an incremental algorithm that divides the problem into

two stages: the problem of assignment of classes to rooms and the whole problem. The incremental algorithm can reduce the size and execution time without losing quality. Moreover, the experimental results also show that using warm-start techniques in this type of problems provides a significant advantage. We warm start the algorithm with a partial feasible solution extracted from the solution of the previous year.

5

International Timetabling Competition 2019

Contents

This chapter describes our different approaches used in the novel tool, *UniCorT*, to solve the International Timetabling Competition (ITC) 2019. The tool is accompanied by a 4-page long explanatory document, which was not peer reviewed [180]. *UniCorT* finished in the top 5, and the results were published in [17, 18].

The organization of this chapter is as follows. First, we formally describe the problem. Next, we describe the different components of *UniCorT*. Section 5.2.1 describes the different pre-processing techniques. Section 5.2.2 the different MaxSAT encodings proposed. Section 5.2.3 describes the local search method used to improve the quality of the solution. Section 5.3 describes the approach to solve MPP in *UniCorT*. Section 5.4 discusses the results obtained by the different configurations of *UniCorT*. Furthermore, we compare *UniCorT* with the state-of-the-art. The differences between IST case study and ITC are discussed. Finally, Section 4.6 concludes the chapter.

5.1 Problem Definition

In this section, we formally describe the ITC 2019 problem introduced in [58]. Let us consider a set of courses $Course$. A course ($course \in Course$) is composed by a set of classes C_{course}. These courses are characterized by configurations ($Config_{course}$) and organized in parts ($Parts^{course}_{config}$). A student must attend the classes in a single configuration. A student enrolled in the course $course$ and attending the configuration $config \in Config_{course}$ must attend *exactly one* class from each part $Parts^{course}_{config}$. The set of classes belonging to $part \in Parts^{course}_{config}$ is represented by C^{course}_{part}.

All classes C (from different courses) must have a schedule assigned to them. Each class $c \in C$ has a set of possible periods (P_c) to be scheduled. Class c has a hard limit on the number of students that can attend it (lim_c). A class c may have a set of possible rooms (R_c). Each room $r \in R_c$ has $capacity \geq lim_c$. Each class may also have a parent-child relationship with another class, *i.e.*, a student enrolled in class c must also be enrolled in the parent $parent_c$.

A time period p corresponds to a 4-tuple (W_p, D_p, h_p, len_p) denoting a set of weeks (W_p), a set of days (D_p), an hour (h_p), and its duration ($len_p > 1$). A set of weeks (days) is a bit-string $W_p = \{100\}$ ($D_p = \{100\}$) that represents that the period p consists of the first week (day) out of 4. A class can only be taught once per day. However, a class can be taught multiple times per week and in multiple weeks. $Weeks$ and $Days$ represent the total number of weeks, days, and hours, respectively. $Days^d$ ($Weeks^w$) corresponds to a particular weekday with $0 < d \leq |Days|$ ($0 < w \leq |Weeks|$).

Let us consider a set of rooms R where the classes can be scheduled. The travel time, in slots, between two rooms $r_1, r_2 \in R$ is represented by $travel^{r_1}_{r_2}$. Each room $r \in R$ has a set of unavailable periods P_r.

Given a set of students S, each student $s \in S$ is enrolled in a set of courses $Course_s$. Furthermore, UCTTP is subject to a set of constraints ($constraint_c$ is the set of constraints relating to class c) that can be divided into hard or soft constraints. The constraints are defined as follows:

- *RoomConflicts*: Two classes cannot be assigned to the same room when they overlap in time.

- *SameAttendees*: The classes cannot overlap in time and must ensure that attendees can travel between rooms with enough time to attend the classes.
- *SameStart*: The classes must start at the same time.
- *SameTime*: The classes must be taught at the same time. For classes with a different duration, the shorter class can start after the longer class as long as it ends before, the longer class ends. For classes with the same duration, it is the same as the *SameStart* constraint.
- *DifferentTime*: The classes must be taught at a different time of day.
- *SameDays*: The classes must be taught on the same days. For classes with a different number of days, the class with fewer meetings must meet on a subset of the days used by the class with more meetings.
- *DifferentDays*: The classes must be taught on different days.
- *SameWeeks/DifferentWeeks*: The same as *SameDays*/*DifferentDays* for weeks.
- *WorkDay*(V): There should not be more than V time slots between the start of the first class and the end of the last class on any given day.
- *Overlap (NotOverlap)*: The classes must (not) overlap in the time of day, the subset of days, and in weeks.
- *Precedence*: The classes must occur one after the other. For classes with multiple meetings in a week, we only enforce the constraint to the first meeting of the class.
- *MinGap*(V): Any two classes that are taught on the same day and week must be at least V slots apart.
- *SameRoom (DifferentRoom)*: The classes must be taught in the same (different) room.
- *MaxDays*(V): A class cannot spread over more than V days.
- *MaxDayLoad*(V): Classes must be spread over the days and weeks such that there are no more than V time slots on every day.
- *MaxBreaks*(V_1,V_2): There are at most V_1 breaks throughout a day. A break between two classes is a gap larger than V_2 time slots.
- *MaxBlock*(V_1,V_2): There are at most V_1 consecutive slots throughout a day. Two classes are considered consecutive if the gap between them is less than V_2 time slots.
- *StudentConflicts*: Two classes cannot overlap in time if the same student is enrolled in both.

There are four optimization objectives: (i) the cost of assigning a class to a room; (ii) the cost of assigning a class to a time slot; (iii) the number of student conflicts and (iv) a set of soft constraints. Each objective has its weights. We solve university course timetabling in two sequential MaxSAT runs.

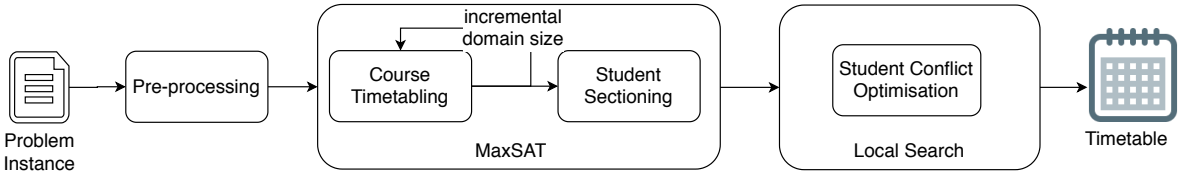


Figure 5.1: Overall schema of *UniCorT*.

First, we solve the course timetabling problem, and then we solve the student sectioning problem. The sequential runs may result in the loss of the global optimum (*i.e.* it may remove the optimal solution in terms of student conflicts). Moreover, it reduces the size of the global problem. Furthermore, this allows us to tackle the MPP using only the first MaxSAT model.

5.2 Introducing *UniCorT*

In this section, we describe *UniCorT*. Figure 5.1 describes the overall schema of the tool, which has three separate components: (i) pre-processing the UCTTP instance, (ii) using a MaxSAT solver to find a solution iteratively, and (iii) improving the quality of the solution with a local search procedure. The iterative algorithm is used to incrementally augment the search space by adding new assignment options (described later on). This novel addition to *UniCorT* is fundamental to solve all the instances of the ITC 2019.

5.2.1 Pre-processing

The pre-processing component relies on four techniques: (i) identifying independent sub-instances; (ii) merging students with exactly the same course enrollment plan; (iii) reducing the classes domain; and (iv) removing redundant constraints.

Identifying independent sub-instances

The first technique divides the problem instance into self-contained sub-instances. A set of sub-instances of the problem instance i (SI_i) is self-contained if and only if the following four constraints are upheld:

1. $\forall_{si_1,si_2 \in SI_i}\ Course_{si_1} \cap Course_{si_2} = \emptyset$;
2. $\forall_{si_1,si_2 \in SI_i}\ R_{si_1} \cap R_{si_2} = \emptyset$;
3. $\forall_{si_1,si_2 \in SI_i}\ S_{si_1} \cap S_{si_2} = \emptyset$;
4. $\forall_{si_1,si_2 \in SI_i}\ \forall_{c \in C_{si_1}} constraint_c \cap C_{si_2} = \emptyset$.

If these constraints are upheld then we can split the instances without removing any solutions.

Example 26. Let us consider a problem instance i with five courses $Course_i = \{co_1, \ldots, co_5\}$ and five rooms $R_i = \{r_1, \ldots, r_5\}$. The classes of the courses co_1 and co_2 can only be taught in two rooms r_1 and r_2. The classes of the courses co_3, co_4 and co_5 can only be taught in rooms r_3, r_4 and r_5,

respectively. Therefore, we can create four sub-instances *SI* $= \{si_1, \ldots, si_4\}$ such that $Course_{si_1} = \{co_1, co_2\}$, $Course_{si_2} = \{co_3\}$, $Course_{si_3} = \{co_4\}$ and $Course_{si_4} = \{co_5\}$.

Consider three students $S_i = \{s_1, s_2, s_3\}$ with the following enrollments: s_1 is enrolled in courses co_1, co_2; s_2 is enrolled in courses co_3, co_4; and finally s_3 is enrolled in co_5. The student enrollments reduce the number of sub-instances since sub-instances si_2 and si_3 violate Constraint 3. Hence, these two sub-instances must be solved together.

Consider a *no overlap* constraint between the classes of the courses co_4 and co_5. As a result, the sub-instances si_3 and si_4 violate Constraint 4. For this reason, the instance i can only be split into two self-contained sub-instances such that $Course_{si_1} = \{co_1, co_2\}$ and $Course_{si_2} = \{co_3, co_4, co_5\}$.

Merging students with exactly the same course enrollment plan

The goal of this technique is to reduce the number of variables and constraints by creating groups of students that share the same curricular plan [181, 182]. The following example illustrates the identification of groups of students with the same curricular plan.

Example 27. Let us consider three courses $Course = \{co_1, co_2, co_3\}$ and six students $S = \{s_1, \ldots, s_6\}$ that are enrolled in courses as follows: $s_1, \ldots, s_4$ are enrolled in the courses co_1 and co_2; and s_5, s_6 are enrolled in the courses co_2 and co_3. In this example, it is possible to generate two *perfect* clusters: clu_1 for all the students enrolled in courses co_1 and co_2; and clu_2 for all the students enrolled in courses co_2 and co_3.

However, this process may remove all the feasible solutions since each course's classes may have a different limitation on the number of students enrolled. Let us denote the greatest common divisor (GCD) between the numbers n_1 and n_2 as $GCD(n_1, n_2)$. Consider now an expansion of the previous example.

Example 28. Let us revisit Example 27 and consider that each course has two classes, and so $C_{co_1} = \{c_1, c_2\}, C_{co_2} = \{c_3, c_4\}$ and $C_{co_3} = \{c_5, c_6\}$. A student enrolled in the courses co_1, co_2, co_3 must attend *exactly one* class of each course. The limit on the number of students that can attend is, for each class, as follows: $lim_{c_1} = lim_{c_2} = 4$; $lim_{c_3} = lim_{c_4} = 3$; and $lim_{c_5} = lim_{c_6} = 2$. Figure 5.2 shows the clusters defined in Example 27 and a possible student sectioned to classes. One can see that the solution is infeasible. For this reason, we computed GCD between the total number of students enrolled in a course and the smallest capacity of the classes of that course. In this case, we obtain: $GCD(4, 4) = 4$ for course co_1; $GCD(6, 3) = 3$ for course co_2; and $GCD(2, 2) = 2$ for course co_3. This indicates that the cluster of students enrolled in co_2 needs to be smaller or equal to 3. Therefore, we need to split the cluster clu_1. One possibility is to divide it into two new clusters clu_3 and clu_4. This feasible solution is shown in Figure 5.3.

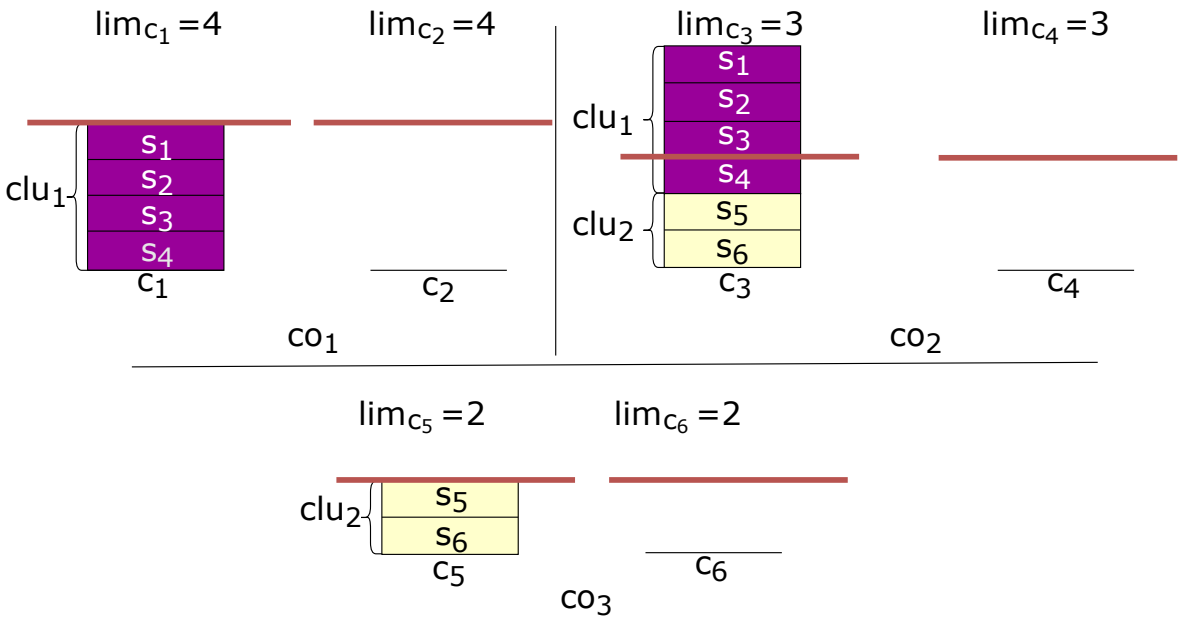


Figure 5.2: An infeasible assignment of students to classes based on the clusters defined in Example 27.

This process ensures that it is possible to find a feasible solution to a problem instance since it is possible to combine all groups of students into classes. However, we may remove the optimal solution by not allowing the assignment of a single student to a given class. The pros and cons of creating clusters are further discussed in Section 5.2.2. Note that the GCD can also be used to choose the number of sections of a course in order to reduce the number of conflicts *a priori* [182].

Reducing the classes domain

The goal is to reduce the domain of the classes by removing possible pairs of room and time assignments that are incompatible with the availability of the room. This technique was previously used in the work of Edon Gashi *et al.* [1].

Example 29. Consider the class c that can be taught in two rooms $R_c = \{r_1, r_2\}$. Furthermore, this class can be taught in three time periods $P_c = \{p_1, p_2, p_3\}$ in the same day of the same week. Room r_1 is unavailable in the periods p_1 and p_2. Room r_2 is only unavailable in the period p_2. Hence, we can remove *a priori* time period p_2 from P_c. Furthermore, we can also remove the pair $(r_1, p1)$.

Removing redundant constraints

The last technique removes redundant constraints referring to *SameAttendees* and to *StudentConflicts*. Both constraints ensure that two classes cannot overlap in time. In both cases, we consider that the two classes overlap in time if they share at least one week, one day of the week, and they overlap in time of a day or if they are consecutive in rooms that are too far apart. Note that *StudentConflicts* are always soft, but the *SameAttendees* may be hard or soft depending on the problem instance.

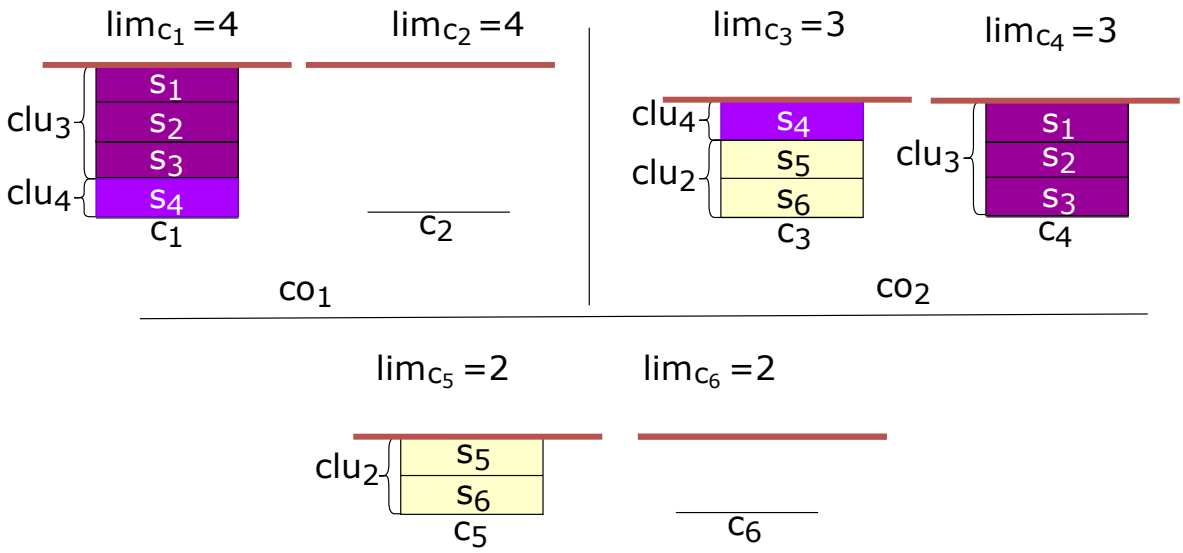


Figure 5.3: A feasible assignment of students to classes based on the clusters defined in Example 28.

5.2.2 MaxSAT

In this section, we formally describe two MaxSAT encodings for course timetabling. The two MaxSAT encodings for course timetabling are denoted as $\mathcal{D}$*irect* and $\mathcal{L}$*inked*.

The overall clauses needed in both encodings are summarized in Table 5.1. The ITC 2019 optimization objectives are encoded as soft constraints in both encodings. Furthermore, we also describe a MaxSAT encoding for student sectioning.

$\mathcal{D}$*irect* Course Timetabling

The $\mathcal{D}$*irect* encoding can be seen as an expansion of the *Boolean* encoding that we have proposed in the previous chapter. This encoding has only two sets of Boolean variables:

- t_c^{slot} represents the assignment of class c to the period $slot$,
  with $c \in C$ and $slot \in [0, \ldots, |P_c|]$;
- r_c^{room} represents the assignment of class c to the room $room$,
  with $c \in C$ and $room \in R_c$.

The existence of two sets of variables allows reducing the number of redundant variables. Using only one set of variables would increase the amount of memory allocation ($|R_c| \times |P_c|$), and most of these variables would be set to value 0.

In the $\mathcal{D}$*irect* encoding, we need two types of *exactly one* constraints. We need to ensure that each class is assigned *exactly one* slot ($\mathcal{D}1$) and when $R_c \neq \emptyset$ we need to ensure that each class is assigned *exactly one* room ($\mathcal{D}2$).

Example 30. Let us consider two classes, c_1 and c_2, with the following characteristics: $D_{c_1} = D_{c_2} = \{0101, 1010\}$; $W_{c_1} = W_{c_2} = \{11110, 01111\}$; $H_{c_1} = \{10, 11\}$; $H_{c_2} = \{11\}$; $P_{c_1} = \{1, \ldots, 12\}$; $P_{c_2} = \{1, \ldots, 8\}$; $R_{c_1} = \{1, 2\}$ and $R_{c_2} = \emptyset$. In this example, we generate the following *exactly*

Table 5.1: Constraints in the $\mathcal{D}$*irect* and $\mathcal{L}$*inked* encodings.

Direct		*Linked*	
$\mathcal{D}1$	$\sum_{slot \in [0,\ldots,\lvert P_c \rvert]} t_c^{slot} = 1$	$\sum_{hour} h_c^{hour} = 1$	$\mathcal{L}1$
		$\sum_{day} d_c^{day} = 1$	$\mathcal{L}2$
		$\sum_{week} w_c^{week} = 1$	$\mathcal{L}3$
$\mathcal{D}2$	$\sum_{room \in R_c} r_c^{room} = 1$		$\mathcal{L}4$
$\mathcal{D}3$	$\neg t_{c_i}^{p_i} \vee \neg t_{c_j}^{p_j} \vee \neg r^{room_{c_i}} \vee \neg r^{room_{c_j}}$	$\neg sd_{c_j}^{c_i} \vee \neg sw_{c_j}^{c_i} \vee \neg h_{c_i}^{hour_{p_i}} \vee \neg h_{c_j}^{hour_{p_j}} \vee \neg r_{c_i}^{room} \vee \neg r_{c_j}^{room}$	$\mathcal{L}5$
$\mathcal{D}4$	$\neg r_c^r \vee \neg t_c^p$		$\mathcal{L}6$
$\mathcal{D}5$	$\neg t_{c_i}^{p_i} \vee \neg t_{c_j}^{p_j}$	$\neg t_{c_i}^{p_i} \vee \neg t_{c_j}^{p_j}$	$\mathcal{L}7$
		$\neg h_{c_i}^{hour_{p_i}} \vee \neg h_{c_j}^{hour_{p_j}}$	$\mathcal{L}8$
		$\neg d_{c_i}^{Day_{p_i}} \vee \neg d_{c_j}^{Day_{p_j}}$	$\mathcal{L}9$
		$\neg w_{c_i}^{Week_{p_i}} \vee \neg w_{c_j}^{Week_{p_j}}$	$\mathcal{L}10$
$\mathcal{D}6$	$\neg r_{c_i}^{room_i} \vee \neg r_{c_j}^{room_j}$		$\mathcal{L}11$
$\mathcal{D}7$	$\sum_{c \in C} \sum_{p \in P_c} \sum_{d \in [1,\ldots,\lvert Day_p \rvert]} dayofweek_d^{const} \leq V$		$\mathcal{L}12$
$\mathcal{D}8$	$\forall_{d=1}^{7} \sum_{c \in C} \sum_{p \in P_c, Day_p^d = 1} t_c^p \times len_p \leq V$	$d_c^{Day_p} \times len_p \leq V$	$\mathcal{L}13$
$\mathcal{D}9$	$\neg t_{c_1}^{p_1} \vee \ldots \vee \neg t_{c_n}^{p_n}$		$\mathcal{L}14$
$\mathcal{D}10$	$\neg b_{h_1}^{const} \vee \ldots \vee \neg b_{h_n}^{const}$ $b_h^{const} \iff t_{c_i}^{p_0} \vee \ldots \vee t_{c_i}^{p_n} \vee t_{c_j}^{p_0} \vee \ldots \vee t_{c_j}^{p_m}$		$\mathcal{L}15$

one constraints: $\sum_{i=1}^{12} t_{c_1}^i = 1$,$\sum_{i=1}^{8} t_{c_2}^i = 1$ and $\sum_{i=1}^{2} r_{c_1}^i = 1$.

The additional constraints of the $\mathcal{D}$*irect* encoding are as follows:

- *RoomConflicts*: For each two classes, c_i and c_j, where $room \in R_{c_i}$, $room \in R_{c_j}$, and $p_i \in P_{c_i}$ overlaps in time with $p_j \in P_{c_j}$, add Constraint $\mathcal{D}3$.
- *SameAttendees*: For each two classes, c_i and c_j, where $room \in R_{c_i}$, $room \in R_{c_j}$, and $p_i \in P_{c_i}$ overlaps in time (or there is not enough time to travel between rooms) with $p_j \in P_{c_j}$, add Constraint $\mathcal{D}3$.
- *RoomUnavailability*: For each room $r \in R_c$ where $p \in P_c$ overlaps with *unavailability* slot of the room, we add $\mathcal{D}4$.
- *SameStart*: For each pair of classes, c_i and c_j, where $p_i \in P_{c_i}$ and $p_j \in P_{c_j}$ correspond to different starting times, add Constraint $\mathcal{D}5$.
- *DifferentTime*(*SameTime*): For each pair of classes, c_i and c_j, where $p_i \in P_{c_i}$ and $p_j \in P_{c_j}$ (not) overlap in time, add Constraint $\mathcal{D}5$.

- *DifferentDays* (*SameDays*): For each pair of classes, c_i and c_j, where $p_i \in P_{c_i}$ is a period with different sub-set (same) days than $p_j \in P_{c_j}$, add Constraint $\mathcal{D}5$.
- *DifferentWeeks* (*SameWeeks*): For each pair of classes, c_i and c_j, where $p_i \in P_{c_i}$ is an period for different (same) weeks than $p_j \in P_{c_j}$, add Constraint $\mathcal{D}5$.
- *WorkDay*(V): For each pair of classes, c_i and c_j, where more than *V* time slots occur between the start time of c_i and the end time of c_j, add Constraint $\mathcal{D}5$.
- *Overlap (NotOverlap)*: For each two classes, c_i and c_j, where $room \in R_{c_i}$, $room \in R_{c_j}$, and $p_i \in P_{c_i}$ (not) overlaps in time with $p_j \in P_{c_j}$, add Constraint $\mathcal{D}5$.
- *Precedence*: For each pair of classes, c_i and c_j, where $p_j \in P_{c_j}$ precedes $p_i \in P_{c_i}$, we add Constraint $\mathcal{D}5$.
- *MinGap*(V): For each pair of classes, c_i and c_j, where $p_j \in P_{c_j}$ and $p_i \in P_{c_i}$ are taught in the same week and day, and $hour_{p_i} + len_{p_i} + V \geq hour_{p_i}$, we add Constraint $\mathcal{D}5$.
- *DifferentRoom* (*SameRoom*): For each pair of rooms, $room_i \in R_{c_i}$ and $room_j \in R_{c_j}$, where $room_i = room_j$ ($room_i \neq room_j$), add Constraint $\mathcal{D}6$.
- *MaxDays*(V): Consider an auxiliary variable, $dayofweek_d^{const}$, where $const$ is the identifier of the constraint and $d \in \{1, \ldots, |Days|\}$. This variable corresponds to having at least one class involved in this constraint, assigned to weekday d. Hence, we only need to ensure that $\mathcal{D}7$.
- *MaxDayLoad*(V): For this constraint, we only need to add the pseudo-Boolean Constraint $\mathcal{D}8$. This constraint ensures that the sum of the classes' length taught on the same day does not exceed *V*.
- *MaxBlock*/*MaxBreaks*(V_1, V_2) To ensure that there are no more than V_1 consecutive slots (breaks) in a day between a set of classes, we need to generate all blocks. After computing all blocks, add the Constraint $\mathcal{D}9$ to ensure that at least one class of an invalid block has to be assigned to a different period. This constraint is added for every class, c_1 to c_n, assigned to a period ($p_1 \in P_{c_1}$ to $p_n \in P_{c_n}$) where the periods form a block of classes that violates one of these constraints.

Example 31. Let us consider two classes, c_1 and c_2, that are taught in the same day and cannot overlap in time. Furthermore, all classes are involved in the *MaxBreaks* constraint, which ensures that there are 0 breaks between them. For simplicity, let us consider that there are only three time slots per day, $t_1 < t_2 < t_3$, and all the classes have the same duration of 1 time slot. Figure 5.4 shows two assignments that violate the *MaxBreaks* constraint. To ensure that the constraint *MaxBreaks* holds, we add clauses $\neg t_{c_1}^{t_1} \vee \neg t_{c_2}^{t_3}$ and $\neg t_{c_2}^{t_1} \vee \neg t_{c_1}^{t_3}$.

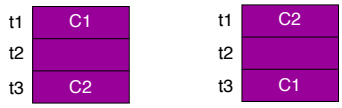


Figure 5.4: Two assignments that violate the *MaxBreaks* constraint between, c_1 and c_2, that are taught in the same day without breaks and cannot overlap in time.

Table 5.2: The relation between variables in $\mathcal{L}$*inked* encoding.

Variables	Relations
h and t	$h_c^{hour} \iff \bigwedge_n t_c^{slot_n}$
d and t	$d_c^{day} \iff \bigwedge_n t_c^{slot_n}$
w and t	$w_c^{week} \iff \bigwedge_n t_c^{slot_n}$
sd and d	$sd_{c_j}^{c_i} \iff (d_{c_i}^{Day_0} \vee \ldots \vee d_{c_i}^{Day_n}) \wedge (d_{c_j}^{Day_{n+1}} \vee \ldots \vee d_{c_j}^{Day_m})$
sw *and* w	$sw_{c_j}^{c_i} \iff (w_{c_i}^{Week_0} \vee \ldots \vee w_{c_i}^{Week_n}) \wedge (w_{c_j}^{Week_{n+1}} \vee \ldots \vee w_{c_j}^{Week_m})$

The method discussed above is impractical (Section 5.4.3) as it generates too many clauses. For this reason, we add an auxiliary variable b_h^{const} where $const$ is the identifier of the constraint and $h \in \{0, \ldots, |Hours|\}$. This variable corresponds to having at least one class involved in this constraint, assigned to an hour h of a single day. This auxiliary variable allows removing the symmetries present in the clause $\mathcal{D}9$. As a result, we add the clause $\mathcal{D}10$ for each block.

Example 32. Recall Example 31. To ensure that the constraint *MaxBreaks* holds, clauses $\neg t_{c_1}^{t_1} \vee \neg t_{c_2}^{t_3}$ and $\neg t_{c_2}^{t_1} \vee \neg t_{c_1}^{t_3}$ are added. However, the second clause is symmetric with respect to the first one. With the new auxiliary variable, we only need to add the clause $\neg b_{t_1}^{const} \vee \neg b_{t_3}^{const}$.

$\mathcal{L}$*inked* Course Timetabling

It is obvious that we do not always need to take into account the complete schedule information. For some constraints, we only need the information about the week, the day or the hour of a class, and not all the three. For this reason, the proposed $\mathcal{L}$*inked* course timetabling encoding has only *four* Boolean decision variables:

- $w_c^{Week_p}$ represents the assignment of class c to the set of weeks $Week_p$, with $c \in C$ and $p \in P_c$;
- $d_c^{Day_p}$ represents the assignment of class c to the set of days Day_p, with $c \in C$ and $p \in P_c$;
- $h_c^{hour_p}$ represents the assignment of class c to the hour $hour_p$, with $c \in C$ and $p \in P_c$;
- r_c^{room} represents the assignment of class c to the room $room$, with $c \in C$ and $room \in R_c$.

The possible schedules for a class are usually only a small part of the complete set. For this reason, we only define these variables for acceptable values in the class domain. Furthermore, the

$\mathcal{L}$*inked* encoding reduces the required number of constraints. For example, *SameStart* constraints (*i.e.*, forcing a set of classes to start at the same time) do not require information about the day or week of the class.

In contrast to the $\mathcal{D}$*irect* encoding, we can reduce the size of each *exactly one* constraint since we have separated the variables for the time allocation. Therefore, we have four *exactly one* constraints for each class (room, hour, day, and week). The reduction in the size of the *exactly one* constraint is important given that it allows avoiding a known bottleneck of timetabling encoding in CNF [68].

Example 33. Recall Example 30. The $\mathcal{L}$*inked* encoding for the same instance generates a much smaller number of *exactly one* constraints. In this example, we generate the following *exactly one* constraints: $\sum_{i=1}^{2} w_{c_1}^i = 1$, $\sum_{i=1}^{2} w_{c_2}^i = 1$, $\sum_{i=1}^{2} d_{c_1}^i = 1$, $\sum_{i=1}^{2} d_{c_2}^i = 1$, $\sum_{i=1}^{2} h_{c_1}^i = 1$, $h_{c_2}^{11} = 1$ and $\sum_{i=1}^{2} r_{c_1}^i = 1$.

The encoding further needs to ensure that each class is assigned *exactly one* hour ($\mathcal{L}$1), one set of days ($\mathcal{L}$2), one set of weeks ($\mathcal{L}$3), and in some cases that each class is assigned *exactly one* room ($\mathcal{L}$4). The last constraint is the same in both encodings.

The variables representing hours, days, and weeks need to have explicit relations since not all hours are available within all days and weeks. For this reason, we still use the period variable t. This variable allows generating binary clauses to encode more complex constraints. To further reduce the size of the clauses, we define the auxiliary variables sw and sd to represent classes that are taught on the same week and day, respectively. Table 5.2 summarizes the relation between variables in the $\mathcal{L}$*inked* encoding.

For some constraints, the $\mathcal{L}$*inked* encoding uses the same clauses as the $\mathcal{D}$*irect* encoding. For this reason, we only describe the clauses that are different, which are as follows:

- *RoomConflicts*: For each two classes, c_i and c_j, where $room \in R_{c_i}$ and $room \in R_{c_j}$, $hour_{p_i} + len_{p_i} > hour_{p_j}$ and $hour_{p_j} + len_{p_j} > hour_{p_i}$, with $p_i \in P_{c_i}$ and $p_j \in P_{c_j}$, add clause $\mathcal{L}$5.
- *SameAttendees*: For each two classes, c_i and c_j, where $hour_{p_i} + len_{p_i} + travel_{room_j}^{room_i} > hour_{p_j}$, with $p_i \in P_{c_i}$, $p_j \in P_{c_j}$, $room_i \in R_{c_i}$ and $room_j \in R_{c_j}$, add $\mathcal{L}$5.

The main difference between the two encodings for these two constraints is the use of two auxiliary variables: the same day and the same week.

- *SameStart*: For each pair of classes, c_i and c_j, where $hour_{p_i} \neq hour_{p_j}$, add clause $\mathcal{L}$8.
- *DifferentTime* (*SameTime*): For each pair of classes, c_i and c_j, where $hour_{p_i}$ and $hour_{p_j}$ do (not) overlap in time, add clause $\mathcal{L}$8.
- *DifferentDays* (*SameDays*): For each pair of classes, c_i and $, c_j$, where $Day_{p_i} \wedge Day_{p_j} = \emptyset$ ($Day_{p_j} \subseteq Day_{p_j}$), add clause $\mathcal{L}$9.

- *DifferentWeeks* (*SameWeeks*): For each pair of classes, c_i and c_j where $Week_{p_i} \wedge Week_{p_j} = \emptyset$ ($Week_{p_j} \subseteq Week_{p_j}$), add clause $\mathcal{L}$10.
- *MaxDayLoad*(V): We only need to add the Constraint $\mathcal{L}$13. The only difference here is the definition of the auxiliar variable. Now, we can use only d.

The remaining constraints are encoded in the same way for both encodings (see the previous section).

Student Sectioning

The usage of clusters requires defining the set $\mathcal{C}luster$ of clusters of students. The number of students merged in $clu \in \mathcal{C}luster$ is represented by $|clu|$.

In order to solve student sectioning, our encoding is extended with *one* decision variable s^c_{clu}, where $c \in C$ and $clu \in \mathcal{C}luster$. The advantage of the pre-processing step for creating clusters is to reduce the number of variables and constraints required to model students. Note that the ITC 2019 instances do not require student sectioning to be *balanced* as in [182]. Moreover, we cannot change the student choice of courses as in [183].

Example 34. Let us consider again Example 27. Recall that we have three courses $Course = \{co_1, co_2, co_3\}$ and six students $S = \{s_1, \ldots, s_6\}$. Students $s_1, \ldots, s_4$ are enrolled in courses co_1 and co_2 and students s_5, s_6 are enrolled in courses co_2 and co_3. Therefore, it is possible to generate two *perfect* clusters: clu_1 for all the students enrolled in courses co_1 and co_2; and clu_2 for all the students enrolled in courses co_2 and co_3. To sum up, the size of the clusters are as follows: $|C_{co_1}| = |C_{co_2}| = 1$ and $|C_{co_3}| = 6$.

The usage of two clusters reduces the number of variables from *22* (the number of students times the number of classes available for each student) to *9* (the number of clusters times the number of classes). Note that the reduction in the number of variables by using clusters depends not only on the number of students merged but also on the course composition. In this case, the smaller cluster (clu_2) achieves the largest reduction in the number of variables (7) since the respective courses have the largest number of classes.

In order to ensure that a student can only be assigned to a single course configuration, we define an auxiliary variable for each pair configuration-cluster of students. The variable is denoted as $conf^{config}_{clu}$, where $clu \in \mathcal{C}luster$, $config \in Config_{co}$ and $co \in Course$.

We need to add an *exactly one* constraint to ensure that each cluster of students clu is enrolled in *exactly one* configuration of each course.

$$\sum_{config \in Config_{co}} conf^{config}_{clu} = 1. \tag{5.1}$$

To ensure that the class capacity is not exceeded, we add the following constraint to each class c:

$$\sum_{clu \in \mathcal{C}luster} |clu| \times s^c_{clu} \leq lim_c. \tag{5.2}$$

In addition, to ensure that a cluster of students clu enrolled in a class c is also enrolled in its parent class $parent_c$, we add the following clause:

$$\neg s_{clu}^{c} \vee s_{clu}^{parent_c}. \tag{5.3}$$

Furthermore, we need to ensure that a cluster of students clu is enrolled in *exactly one* class of each part of a single configuration of the course co. Hence, for each cluster of students clu and for each pair of two classes, c_i and c_j, such that $c_i, c_j \in C_{part}^{co}$, where $part \in Parts_{config}^{co}$, we add:

$$\neg conf_{clu}^{config} \vee \neg s_{clu}^{c_i} \vee \neg s_{clu}^{c_j}. \tag{5.4}$$

In addition, for each cluster of students clu and for each $part \in Parts_{config}^{co}$ we add:

$$\neg conf_{clu}^{config} \vee s_{clu}^{c_i} \vee \ldots \vee s_{clu}^{c_{|C_{part}^{co}|}}. \tag{5.5}$$

The conflicting schedule of classes attended by the same cluster of students is represented by a set of weighted soft clauses. For each cluster of students clu enrolled in two classes, c_i and c_j, overlapping in time, we add:

$$\neg s_{clu}^{c_i} \vee \neg s_{clu}^{c_j} \vee \neg sw_{c_j}^{c_i} \vee \neg sd_{c_j}^{c_i} \vee \neg h_{c_i}^{hour_{c_i}} \vee \neg h_{c_j}^{hour_{c_j}}. \tag{5.6}$$

5.2.3 Local Search

The *goal* of this procedure is to improve the quality of the solution found without changing the schedule and room assignments of the courses. Neighborhood structures are the basis of this local search (LS) procedure. In this work, the neighborhood consists of small changes in the student sectioning. To create a new neighborhood, two operations can be performed: (i) allocating a cluster of students to a different class with empty seats and (ii) swapping two clusters of students between classes. Considering these moves, the procedure does not require the knowledge of course timetabling constraints. The LS procedure stops when the neighbors of the best solution cannot reduce the number of conflicts (*i.e.* the solution found has the best cost of its neighborhood).

Example 35. Let us consider again Example 28. Additionally, consider that the classes c_3 and c_5 are taught at the same time, on the same day, and in the same week. For this reason, the solution shown in Figure 5.3 has two conflicts (students s_5 and s_6 are sectioned into two classes that overlap in time). This solution has two neighbors with a better solution. These two neighbors are shown in Figure 5.5. The neighbor **M1** swaps students s_5 and s_6 with students s_2 and s_3 (from class c_3 to c_4). However, this move is not possible since the clusters do not allow separating students s_2 and s_3 from s_1. The neighbor **M2** results from just sectioning students s_5 and s_6 to class c_6 instead of c_5. This change does not require breaking any clusters and reduces the number of conflicts to zero.

5.3 Disruptions

Figure 5.6 illustrates the process of solving the university timetabling problem subject to disruptions. The process starts with a problem instance and a timetable, and ends when a new feasible

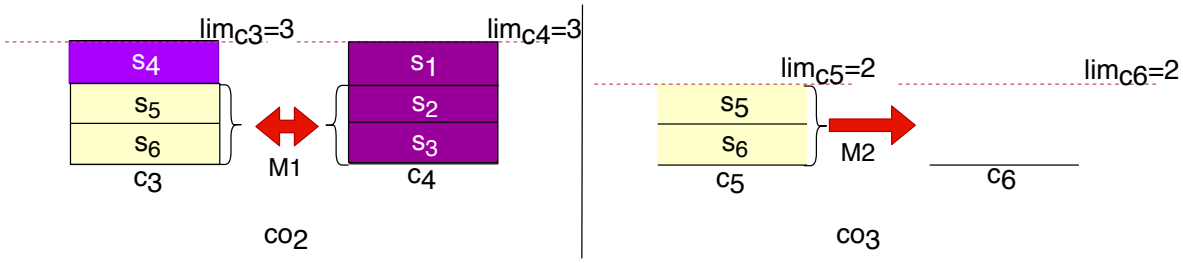


Figure 5.5: The two neighbors of the solution in Example 28. The neighbor on the left is invalid since it breaks a cluster.

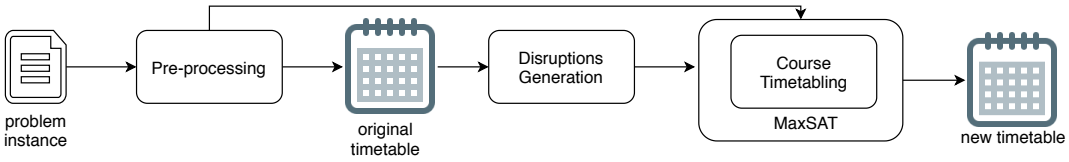


Figure 5.6: Algorithm schema to solve university timetabling problems subject to disruptions.

timetable is found. Recall that each problem instance is pre-processed before generating the encoding.

In this case study, we consider the following disruptions: *invalid time* and *invalid room*. These disruptions reduce the domain of a specific class c in terms of available time slots or rooms. Disruptions in the student enrollments would only cause changes in the student sectioning part. The problem definition has the underlining the assumption that all the rooms in the domain of class have enough capacity for the students attending. As our original solutions are sub-optimal we do not consider disruptions in the enrollments.

Invalid Time: The time slot t is no longer available for class c:

$$\neg t_c^t. \tag{5.7}$$

Invalid Room: The room r is no longer available for class c:

$$\neg r_c^r. \tag{5.8}$$

When recovering from disruptions, we apply lexicographic optimization with two objectives: (i) the HD and (ii) the overall quality of the solution (computed based on the four objectives defined above). This way, we can take advantage of the disruption to improve the quality of the solution.

5.4 Experimental Evaluation

In this section, we discuss the main computational results obtained. This section is organized as follows. Section 5.4.1 describes the experimental setup used to validate *UniCorT* is described. Section 5.4.3 discusses our results for UCTTP. Section 5.4.4 presents a summary of the results. Finally, Section 5.4.5 shows a comparison between *UniCorT* and the other solutions proposed in the context of the ITC 2019.

5.4.1 Experimental Setup

The experimental evaluation targets the performance of *UniCorT* on the benchmark from the ITC 2019 [58]. The benchmark is divided into three groups of instances (early, middle, and late). All results were verified by an online validation tool provided by the organizers [1].

The XML parser used to parse the ITC 2019 input file was *RAPIDXML*[2].

UniCorT was implemented in C++ and is available on github (`https://github.com/ADDALemos/MPPTimetables/`). *UniCorT* uses the MaxSAT solver *TT-Open-WBO-Inc* [33, 184][3] as a black box. *TT-Open-WBO-Inc* is a linked MaxSAT solver [32] that has different algorithms and encodings to solve a given problem. The results shown in this thesis correspond to the best configuration of the parameters of the solver. The solver was executed with the following parameters: `-algorithm=6` corresponding to the use of linear search with the clusters algorithm [34]; `-pb=2` corresponding to the use of the adder encoding [185] to convert the PB constraints to CNF; and `-amo=0` corresponding to the use of the ladder encoding [186] to convert *exactly one* constraints to CNF.

The linear search with the clusters algorithm uses a lexicographic optimization objective [187]. Recall that the ITC 2019 considers four optimization objectives:

- the cost of assigning a class to a room (*Room*);
- the cost of assigning a class to a time slot (*Time*);
- the number of student conflicts (*Students*);
- the weighted sum of violated soft constraints (*Distribution*).

Each instance has its own weight for each objective. We have computed the worst possible penalization for each objective and used the obtained values to configure the lexicographic optimization.

The experiments were performed on a computer with Fedora 14, with a 2.6 GHz CPU and 128 Gb of RAM.

All results were obtained when running the solver with a time limit of 6 000 seconds. It was observed that increasing the execution time would lead to spending more time in memory management than actually solving the problem. For example, no improvements in the solutions to solve the ITC 2007 benchmark were observed when using 10 000 seconds as the time limit as in Asín Achá and Nieuwenhuis [70].

5.4.2 Data characteristics

Table 5.3 shows the different characteristics of the instances. One can see that the instances are distinct from each other. Instances from *iku** are the largest in terms of classes. However, they do

[1] `https://www.itc2019.org/validator`, accessed in August 2020.

[2] *RAPIDXML* is available at `http://rapidxml.sourceforge.net/`, accessed in February 2019.

[3] *TT-Open-WBO-Inc* won both the Weighted Incomplete tracks at *MaxSAT Evaluation 2019*. The results are available at `https://maxsat-evaluations.github.io/2019/`.

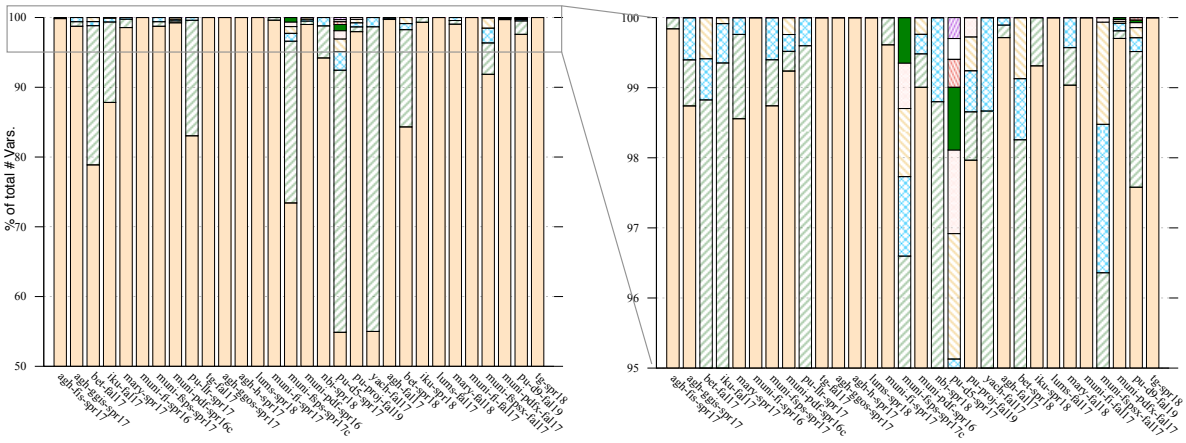


Figure 5.7: Distribution of variables for each sub-instance. Each pattern represents a different sub-instance.

not have students or *MaxBlock/MaxBreak*. They have one order of magnitude more variables than the next largest instance (despite not having students). The *muni-f** instances have a particular small search space in terms of possible rooms per class (only 4).

5.4.3 Computational Evaluation

In this section, we discuss the results of *UniCorT* using different configurations.

Pre-processing Techniques

Recall that we discussed four pre-processing techniques: (i) identification of independent sub-instances; (ii) merging students into clusters; (iii) reducing the classes domain; and (iv) removing redundant constraints.

Identification of independent sub-instances. The identification of independent sub-instances allows splitting the problem without losing solutions. In the end, it is just a question of combining all the solutions.

The distribution of variables for each sub-instance created is shown in Figure 5.7. The sub-instances are identified with different patterns. On average, we can split an instance into 3 sub-instances. In most cases, the instances have one large sub-instance and two smaller sub-instances (except for the instances *pu-d5-spr17* and *yach-fal17*). All instances have a sub-instance with at-least 50% of the total number of variables. Most sub-instances are small and have less than 15% of the variables of the original instance.

Figure 5.8 shows the execution time for each sub-instance for the best configuration of the solver. The best configuration is the same for all instances. The patterns identify the sub-instances. The patterns shown in Figure 5.7 and missing from Figure 5.8 are sub-instances with neglectable execution times. Note that small instances require a smaller amount of the total execution time of the instance. On average, 70% of the execution time is spent on the largest instance, and the rest is uniformly

Table 5.3: Data sets per university (instances sorted by # of variables).

		$\|C\|$	Avg. $\|R_c\|$	Avg. $\|P_c\|$	$\|S\|$ (k)	#MaxBreak	#MaxBlock	# Var. (k)
yach-fal17		417	4	43	1	0	0	19
nbi-spr18		782	4	38	2	0	0	35
tg*	Avg.	693	11	24	0	0	0	42
	Median	693	11	24	0	0	0	42
mun-f*	Avg.	743	4	44	1	0	3	45
	Med.	700	4	30	1	0	2.5	38
mary*	Avg.	916	14	12	4	0	0	47
	Med.	916	14	12	4	0	0	47
lums*	Avg.	494	26	43	0	0	0	82
	Med.	494	26	43	0	0	0	82
bet*	Avg.	1 033	25	23	3	24	19	140
	Med.	1 033	25	23	3	24	19	140
pu*	Avg.	3 418	12	33	28	16	0	196
	Med.	1 929	12	30	31	17	0	125
muni-pdf*	Avg.	2 586	15	53	4	0	13	374
	Med.	2 526	17	56	3	0	10	373
agh*	Avg.	1 955	34	89	3	15	0	380
	Med.	1 239	10	75	2	14	0	340
iku*	Avg.	2 711	25	34	0	0	0	1 050
	Med.	2 711	25	34	0	0	0	1 050

allocated to the smaller instances. However, this technique has a significant impact on the memory spent.

Globally, this technique allows reducing the execution time by 8% and the memory consumption by 18%. Note that we considered a sequential execution of all sub-instances. Hence, further gains could have been achieved with parallel execution.

Merging Students. Merging students with the same curricular plans allows reducing the number of variables and constraints on the student sectioning part of the problem. Figure 5.9 shows the percentage of the total number of variables required to model students in different clusters. The *clusters* represent the percentage of the total number of variables required to model students with different curricular plans. However, this type of clusters cannot be applied in practice since they would remove feasible solutions (see Example 5.3). Alternately, the *GCD clusters* represent the cluster divided using the GCD method discussed above. Recall that the number of variables needed to model students is influenced by the number of classes per enrolled course (see Example 34).

Most instances have a significant bottleneck in the creation of clusters caused by the hard limit on the number of students enrolled in a class. On average, the *GCD* clusters are 40 points worse than a normal cluster. On average, one can reduce the number of variables related to students up to 23%. Instances *nbi-spr18* and *yach-fal17* exhibit a larger reduction on the number of variables (around 50%). On the other hand, instances *pu** exhibit the smallest reduction (14%).

Reducing the Class Domain. Reducing the class domain implies reducing the number of possible time-room assignments for each class. Removing *a priori* time slots results in removing on average

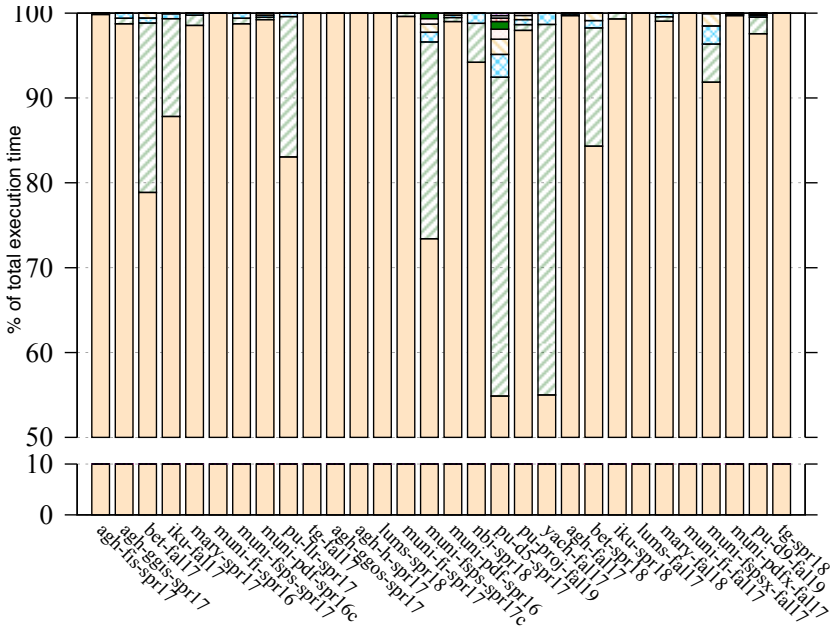


Figure 5.8: Allocation of execution time to each sub-instance. Each pattern represents a different sub-instance.

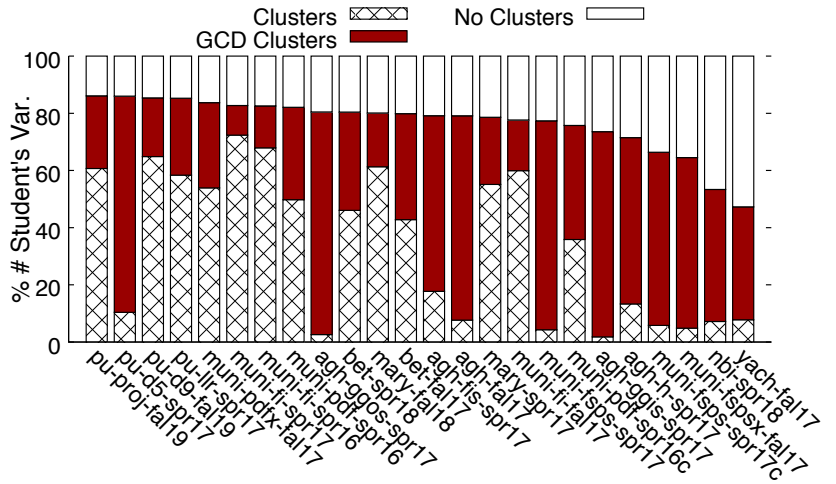


Figure 5.9: Number of variables required to model students sectioning with increasing clustering strategy.

10% of the number of possible assignments (*i.e.*, the number of variables).

If our models had only one type of decision variables, we could take full advantage of this pre-processing technique. However, as our models have multiple different types of decision variables, we can only remove the options that are unavailable for all rooms. Nevertheless, we can still remove on average 4% of the options. The possible gains from removing all possible pairs *a priori* is outweighed by the number of constraints needed when using only one decision variable.

Removing Redundant Constraints. The goal here is to remove redundant constraints. In our work, we remove only *StudentConflict* constraints. These constraints sometimes overlap with *SameAttendes constraints*. On average, we are able to remove 6% of the student conflicts. This step is important to reduce the memory consumption when the MaxSAT solver is handling student sectioning.

The structure of the local search procedure does not benefit directly from the application of this

technique. The local search only solves the student sectioning problem and thus never considers *SameAttendes constraints*. Nevertheless, the local search is performed in the neighborhood of the original solution. Therefore, the quality of the solution found by the MaxSAT solver is important.

MaxSAT versus ILP

In the previous chapter, we proposed novel ILP encodings to solve the specific course timetabling problem in IST. Therefore, our first approach to solve the ITC 2019 problem was a direct expansion of the MIXED model[4] proposed in the previous chapter.

The ILP encoding was able to solve only the smallest 5 instances. This value increased to 10 with the novel pre-processing techniques described in this chapter. These results can be explained by the combination of a large number of classes and time options. Recall that at IST we solved the timetables for a single week. This type of decomposition cannot be used here, as the classes have a diverse set of weeks to be taught. Clustering classes per week would make the problem infeasible.

The MaxSAT encoding is similar to the ILP encodings. The difference in performance can be explained by the way the solver deals with the different types of constraints in the ITC 2019 instances. The ITC 2019 constraints are of a different nature and thus create many more binary clauses than the IST case study. MaxSAT solvers perform better with binary clauses. This can be explained by the nature of the unit propagation procedure of the solvers [31, 188].

We experiment with different time limits. However, the major limitation of both approaches is memory. When we relax the time limit, the ILP solver will stop improving and start managing memory. Nevertheless, mixed-integer linear programming is a good approach to solve the ITC 2019. In fact, the winner[71] used an encoding similar to the one proposed in this paper. The main difference lies in the decomposition of the problem. Recall that this decomposition allows the algorithm to solve the problem in parallel with solid heuristics to guide the search. Furthermore, Holm *et al.*[71] required ten days of execution time to find the best solutions[5]. The impact of the parallelization was essential to remove the memory issues that we have experienced.

MaxSAT Solving

This section evaluates the MaxSAT part of *UniCorT* with different Weighted Conjunctive Normal Form (WCNF) encodings. Recall that we split the course timetabling from the student sectioning problem. For this reason, the $\mathcal{D}irect$ course timetabling encoding described in this thesis can be seen as an expansion of the work of Asín Achá *et al.* [70]. However, the addition of the concept of weeks and the new pseudo-Boolean constraint caused the problem to have worse performance than expected without string pre-processing and incremental algorithms. Furthermore, the notion of weeks and days in this competition can be exploited with $\mathcal{L}inked$ encoding.

Figure 5.10 compares the total number of hard clauses with the number of soft clauses generated by our encodings. It is clear that the number of soft clauses is considerably smaller for all instances.

[4]The model with the best results for the case study at IST.
[5]The comprehensive overview of their tool is available at: https://www.itc2019.org/papers/itc2019-holm-slides.pdf.

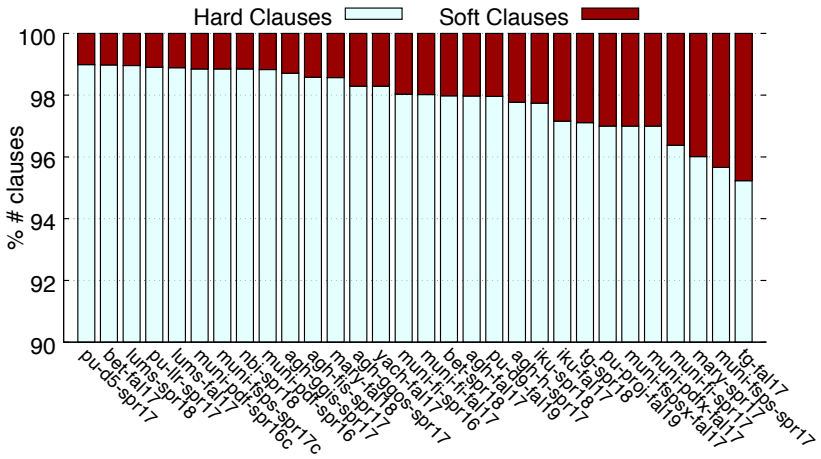


Figure 5.10: Percentage of soft clauses for each instance.

On average, the number of soft clauses is 2% of the global number of clauses. Most instances that timed out have a higher percentage of soft clauses, but these instances also have a larger overall number of soft clauses. With this difference in mind, we focused more on the hard constraints as they are dominant.

UniCorT is able to find a solution within the time limit for *all* instances using the best approach (see Table 5.5). However, the solver was not able to prove optimality for the competition instances, either due to time or memory limitations. We were only able to prove optimality in the four *test* instances of the ITC 2019. As these test instances do not enter in the competition and their characteristics are completely different (much simpler), the results are not shown in this thesis. Note that the solver used is incomplete. The solvers submitted to the incomplete track do not have to be complete, although some of them are if given enough time. The linear search and clustering algorithm do not guarantee optimality due to the creation of clusters. Nevertheless, in these instances, the algorithm is able to prove optimality since there is only one cluster created.

Figure 5.11 compares the number of hard clauses generated by the WCNF encoding and the CPU time needed to find the best solution for each instance considering two approaches to encode the problem ($\mathcal{D}$*irect* and $\mathcal{L}$*inked*). Recall that the number of soft clauses is, on average, 4% of the global number of clauses. In general, one can see that the instances with a larger number of hard constraints require more time to be solved. The $\mathcal{L}$*inked* encoding reduces the number of constraints needed per instance. In practice, 15 more instances are solved within the time limit.

The $\mathcal{D}$*irect* encoding requires, on average, 7×10^{10} more constraints than the $\mathcal{L}$*inked* encoding. This can be explained by the fact that most constraints are only related either to an hour, day, or week. There are only a few constraints that involve all weeks, days, and hours simultaneously. Furthermore, one can reduce the size of the constraints by the usage of auxiliary variables (*e.g.* sd). For this reason, the usage of only one variable for the time allocation problems creates redundant constraints.

In the case of the $\mathcal{L}$*inked* encoding, for most instances, the solver requires only a short amount of time to produce the best solution. Figure 5.12 shows a comparison between the normalized cost

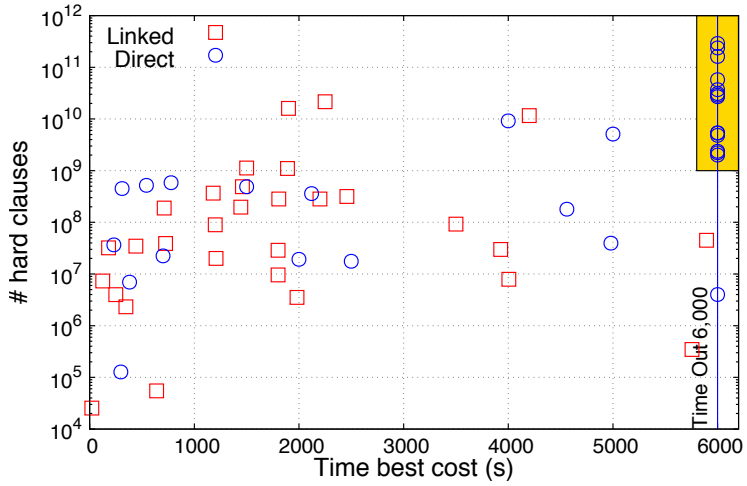


Figure 5.11: A comparison between the $\mathcal{L}inked$ and the $\mathcal{D}irect$ encodings in terms of the number of hard constraints (log scale) versus the time spent to find the best solution. Both encodings were executed after removing symmetries and with the iterative algorithm. The yellow square contains the instances that timed out. 0% and 50% of the instances are in the square for $\mathcal{L}inked$ and the $\mathcal{D}irect$ encodings, respectively.

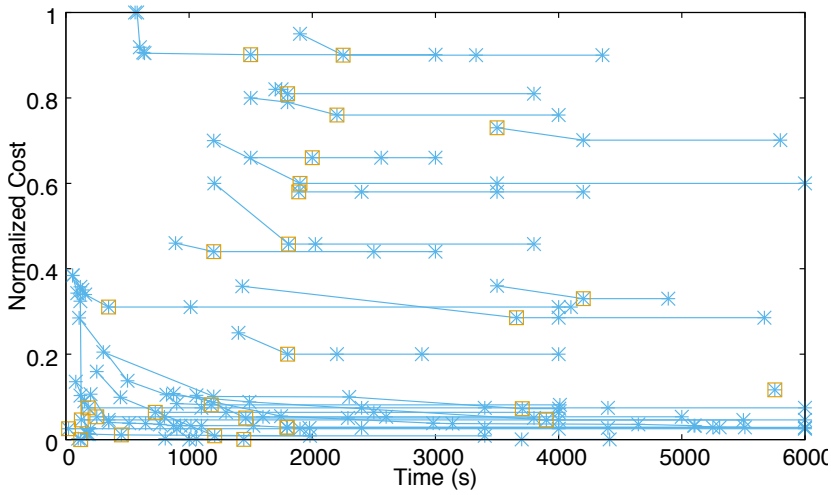


Figure 5.12: Normalized cost versus CPU time for each instance with $\mathcal{L}inked$ encoding. The yellow square represents the best cost found.

of the best solution found and the CPU time in seconds. The figure shows the normalized cost since each instance has its own weights on the optimization objective and therefore would be impossible to compare them in the same graph. One can see that the best solution for most instances is found early on (within 2 000 seconds). In fact, for only 7 out of 30 instances, the quality has been improved after 2 000 seconds.

***MaxBlocks* and *MaxBreaks*.** Figure 5.13 shows the percentage of clauses generated from *MaxBlocks* and *MaxBreaks* constraints for each instance. One can see that these constraints generate a significant number of additional clauses. In the worst case, we need to generate over 35% more clauses to deal with these constraints, thus making it impossible to solve even small instances.

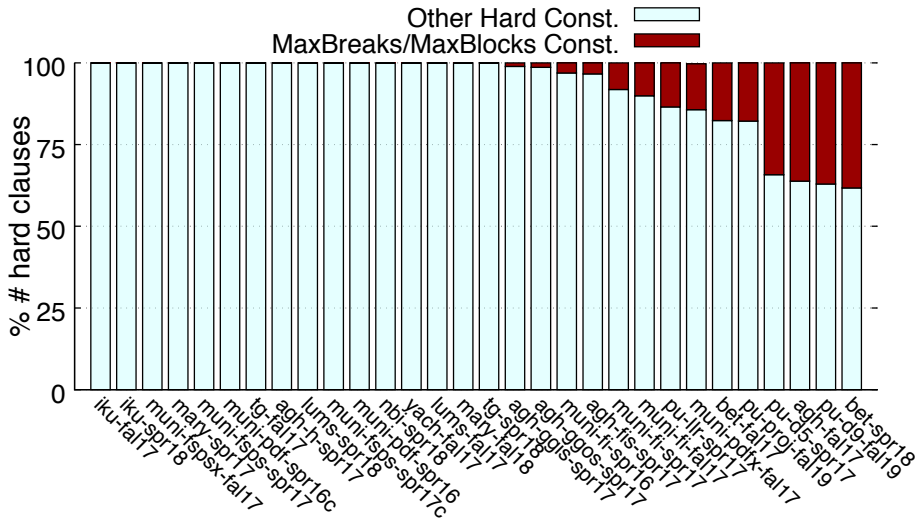


Figure 5.13: Percentage of clauses generated by *MaxBlocks* and *MaxBreaks* constraints.

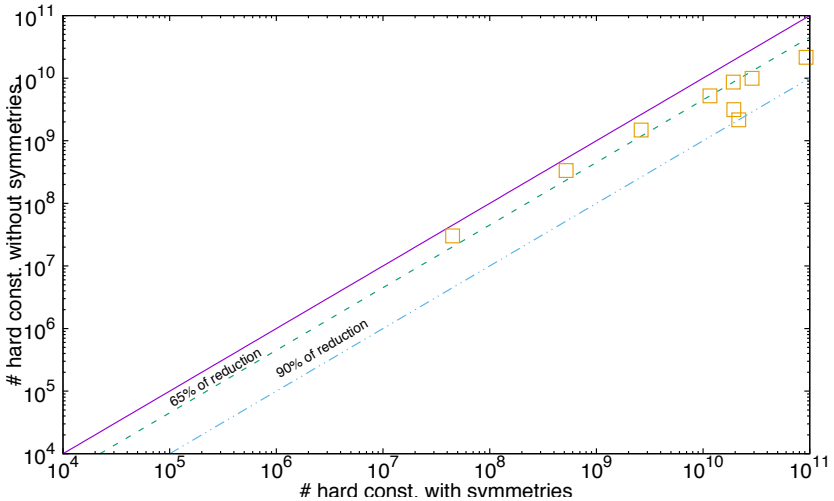


Figure 5.14: Percentage of clauses generated with and without removing symmetries for the $\mathcal{L}$*inked* encoding.

Symmetry breaking. Symmetry breaking procedures have been successfully applied to solve university course timetabling problems [146]. Symmetry breaking used in *MaxBlock* and *MaxBreak* constraints is an important step to solve more instances. With symmetry breaking, we are able to model the *MaxBlock* and *MaxBreak* constraints adding less than 1% of the new constraints.

Figure 5.14 shows the number of constraints generated with and without symmetries using the $\mathcal{L}$*inked* encoding for the instances with *MaxBlock* and *MaxBreak* constraints. As one can see, we reduce the number of clauses, on average, by 65% (dashed green line). In the best case, we can reduce the number of clauses from 10^{10} to 10^{9}.

Decomposing UCTTP. Our best approach decomposes the UCTTP into two sub-problems: (i) course timetabling and (ii) student sectioning. This decomposition may remove the optimal solution. However, it does not remove any feasible solution. The goal of the decomposition is to reduce the size of the problem, especially for instances with a large number of clusters of students. The

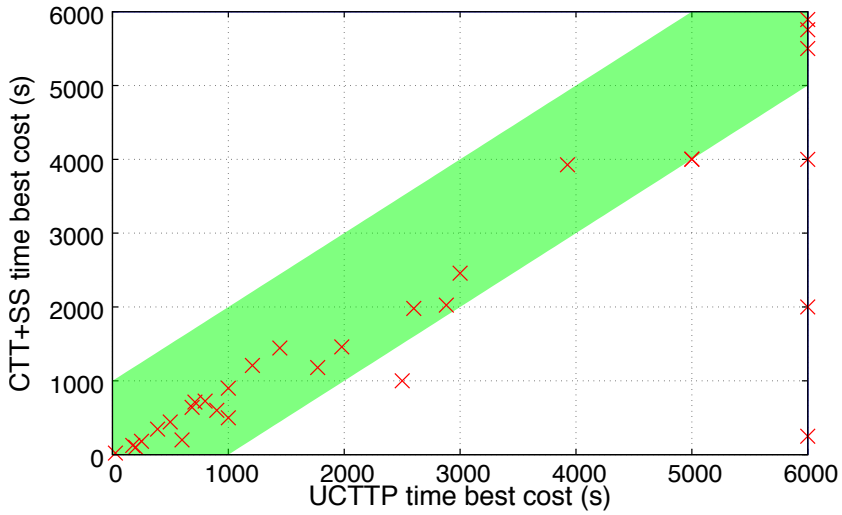


Figure 5.15: A comparison of the CPU time, in seconds, when solving the *CTT+SS* problems separated or the *UCTTP* as a whole.

decomposition allows us to solve 6 more instances. Figure 5.15 compares the performance of the solver before and after decomposing the problem in terms of CPU time.

Iterative approach. The largest instances (*e.g. iku**) make the proposed *exactly one* constraints to be impractical for the solver to handle. Recall that the encoding of *exactly one* constraints in CNF depends on the number of variables (nv) involved in the constraint. The experiments performed by Bittner et al. [68] showed that most instances with *exactly one* constraints with $nv > 26$ result in a memory overflow. In our case, nv depends on the number of possible time or room assignments for each class. One of the major bottlenecks relates to *exactly one* constraints for the time options of a class. The decomposition into two sub-problems: (i) course timetabling and (ii) student sectioning does not reduce nv.

Next, we propose an iterative algorithm to solve the aforementioned problem. The algorithm starts by adding only the time options that have a zero penalty. The algorithm checks in each iteration if a feasible solution is found. If that is the case, the MaxSAT solver starts optimizing the solution. In each subsequent iteration, we increase the value of the penalty, thus increasing the search space. A new iteration occurs every time one of two things happens: (i) the solution is infeasible or (ii) we found an optimal solution for this sub-problem. The algorithm ends when one of two things occurs: (i) the optimal solution is found for the whole problem or (ii) the time limit is reached.

Table 5.4 shows the number of iterations required to find a feasible solution and the total number of iterations performed before the time limit is reached. One can see that, on average, only one iteration occurs after the first feasible solution is found. This can be explained by the fact that iterations leading to feasible solutions have larger execution times than iterations leading to the infeasible solution since they require optimization. All iterations but the last one are only solving a decision problem with no feasible solution. For example, the first 32 iterations to solve *pu-d9-fal19* do not find a feasible solution and the solver only starts optimizing when a feasible solution is found (iteration 33). No more iterations

Table 5.4: Comparison between the number of iterations required to find a feasible (#SAT) solution and the number of iterations performed before the time limit is reached (#Run).

Inst.	agh-fis-spr17	agh-ggis-spr17	bet-fal17	iku-fal17	mary-spr17
#SAT	5	6	5	3	5
#Run	6	6	6	3	6
Inst.	muni-fi-spr16	muni-fsps-spr17	muni-pdf-spr16c	pu-llr-spr17	tg-fal17
#SAT	27	2	22	41	7
#Run	30	2	23	42	9
Inst.	agh-ggos-spr17	agh-h-spr17	lums-spr18	muni-fi-spr17	muni-fsps-spr17c
#SAT	5	3	2	2	2
#Run	7	6	4	6	2
Inst.	muni-pdf-spr16	nbi-spr18	pu-d5-spr17	pu-pro j-fal19	yach-fal17
#SAT	27	12	55	34	3
#Run	29	14	57	34	4
Inst.	agh-fal17	bet-spr18	iku-spr18	lums-fal17	mary-fal18
#SAT	5	3	2	3	5
#Run	5	5	2	4	7
Inst.	muni-fi-fal17	muni-fspsx-fal17	muni-pdfx-fal17	pu-d9-fal19	tg-spr18
#SAT	9	3	5	33	4
#Run	11	3	5	33	7

are performed given that in the meantime, the time limit is reached.

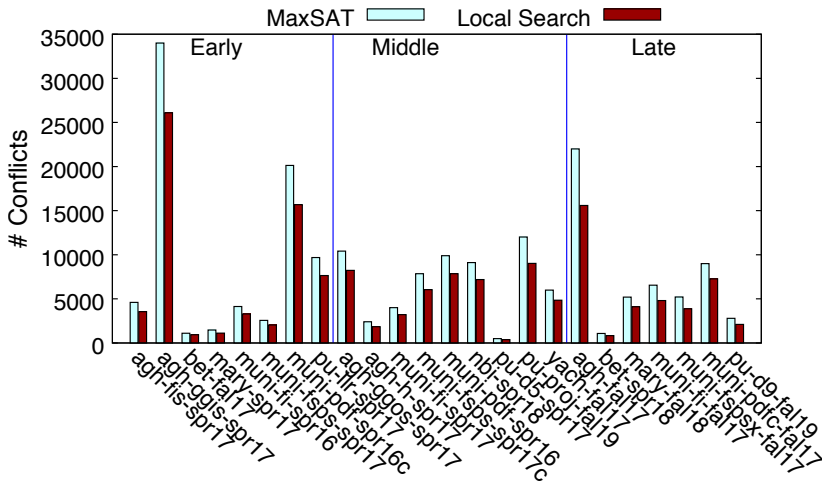


Figure 5.16: A comparison of the cost, in terms of students conflicts, before and after applying the LS procedure.

Local Search

Our straightforward implementation of this method allows improving the quality of the solution without adding significant overhead. On average, the method requires only 6% of the overall execution time. Figure 5.16 compares the number of conflicts before and after this procedure. On average, the procedure reduces the number of conflicts by 22%.

5.4.4 Final Results

Table 5.5 shows the best cost value found by *UniCorT*, in its best configuration, per optimization objectives and instances. Note that the penalties associated with the three optimization objectives (student conflicts, allocation penalty, and additional soft constraints) vary from instance to instance. Hence, it is difficult to make a fair comparison. Nevertheless, one can see that the student conflict objectives, overall, are the most costly, even with the LS method. The *muni** instances are on average the worst in terms of room allocation penalty. This can be explained by the structure of these instances since they have fewer room options (R_c) and a large penalty associated.

Table 5.5: The cost per optimization objectives and instance.

Instance	Cost	Students	Time	Room	Distribution
agh-fis-spr17	35 139	3 555	2 248	2 312	404
agh-ggis-spr17	161 118	25 558	2 474	6 692	1 116
bet-fal17	296 015	942	224	8 963	8 932
iku-fal17	29 929	0	22 444	5 805	56
mary-spr17	51 147	1 114	1 376	805	7 290
muni-fi-spr16	19 314	3 286	352	628	120
muni-fsps-spr17	211 142	2 040	58	292	360
muni-pdf-spr16c	567 900	15 678	58 316	27 600	4 361
pu-llr-spr17	68 003	7 642	1 169	2880	30
tg-fal17	6 774	0	1 792	30	158
agh-ggos-spr17	79 745	8 230	6 045	9 045	358
agh-h-spr17	55 887	1 848	1 442	1 039	2 656
lums-spr18	119	0	0	49	14
muni-fi-spr17	18 080	3 212	284	958	21
muni-fsps-spr17c	618 217	6048	411	1 027	141
muni-pdf-spr16	310 994	7 853	38 680	27 094	900
nbi-spr18	49 924	7 196	5 946	9 208	5
pu-d5-spr17	17 513	380	1130	391	712
pu-proj-fal19	126 568	9020	5750	11110	1628
yach-fal17	32 198	4 856	8	1 008	687
agh-fal17	142 687	15 594	3 993	3 880	2 991
bet-spr18	353 920	826	188	8 644	10 968
iku-fal18	45 537	0	36 838	6 299	80
lums-fal17	813	0	60	613	16
mary-fal18	44 097	4 107	596	665	234
muni-fi-fal17	19 683	3 810	86	289	9
muni-fspsx-fal17	401 155	3 878	323	1 215	271
muni-pdfx-fal17	228 560	7 263	13 045	23 045	1 760
pu-d9-fal19	71 903	2 101	3 666	7 453	410
tg-spr18	31900	0	1942	3996	1201

5.4.5 Comparison with the State-of-the-Art

The ITC 2019 continues to keep an active research branch with updated results. In total 23 participants evaluated their instances with ITC 2019 tool. However, only 13 participants submitted a

Table 5.6: The best results, in terms of points, on the ITC 2019 benchmark as May 30, 2021 (https://www.itc2019.org/score).

Position	Author	Early	Middle	Late	Total
1.	Holm et al. [71]	97	126	226	449
2.	UniTime	64	123	195	382
3.	Rappos et al. [72]	58	61	123	242
4.	Gashi and Sylejmani [1]	21	55	100	176
5.	Er-rhaimini [86]	18	45	96	159
6.	**Alexandre Lemos, Pedro T Monteiro and Inês Lynce**	11	30	111	152
7.	I Gusti Agung Premananda	5	25	54	84
8.	Jason C.H	2	10	45	57
9.	Marlúcio Alves Pires	7	17	27	51
10.	Georgia Ioanna Makraki	0	3	12	15
11.	Eduardo Flores	0	3	2	5
12.	Jerry Wang	0	2	0	2
13.	Quentin Peña	0	0	1	1

valid solution for at least one benchmark instance (not considering test instances). The results, in terms of points[6], on May 30 of 2021, are shown in Table 5.6. Recall that these results are obtained without any time and memory limits. Note that solutions with 0 points do not mean the approach did not solve any instances. The points are assigned F1-style, and thus if your solution has a low quality, you may have 0 points.

The simulated annealing approach proposed by Gashi and Sylejmani [1] solves only 16 out of 30 instances within 6 000 seconds. For the remaining 14 instances, the provided solutions are infeasible, violating hard constraints. On average, the provided solutions violate 12 hard constraints. Note that the simulated annealing approach, without a time limit, can continue to improve its quality. Given enough time, this approach is able to find solutions to all instances.

Figure 5.17a compares *UniCorT* with the simulated annealing tool [1] in terms of the execution time needed to find the first feasible solution. One can see that, for all instances, *UniCorT* finds a solution quickly. Note that the cases where simulated annealing takes 6 000 seconds mean no feasible solutions were found. Figure 5.17b compares the cost of the solution found by simulated annealing and by *UniCorT*. Solutions that are not feasible are shown in Figure 5.17b with a cost of 900×10^3. One can see that *UniCorT* finds a solution with better quality for 4 instances. On the other hand, simulated annealing is able to find 12 solutions with better quality. However, recall that *UniCorT* finds a solution on more than 14 instances within the time limit. The reason for the improved quality found by simulated annealing is the way this approach deals with student conflicts. The 4 instances for which *UniCorT* is able to find a better solution have in common the fact that they do not have students.

Figure 5.18 shows the quality of the solution found by the simulated annealing tool overtime with the time limit of 6 000 seconds. One can see that it is slowly but continuously improving the quality of the solution. The regular *jumps* where the quality of the solution gets significantly worse are justified by the restart strategy. This process is used to avoid local minima [189]. Figure 5.18 also shows that

[6]The points are calculated based on the rules described in [58].

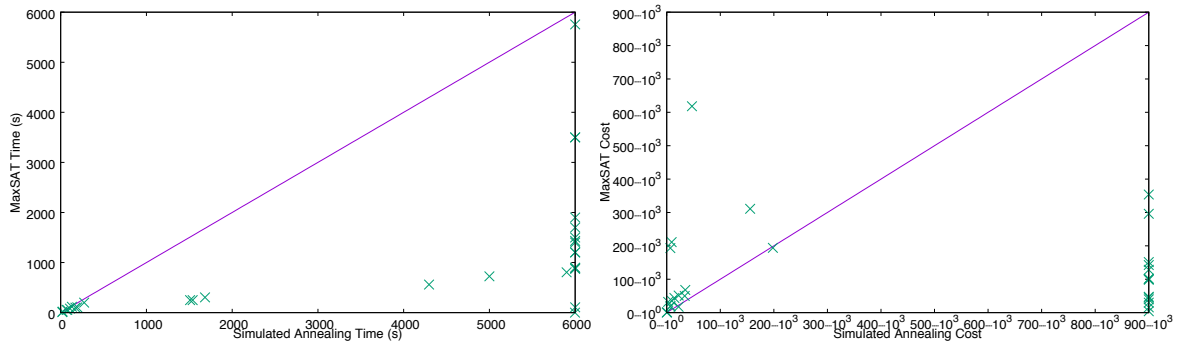


Figure 5.17: Comparison between the Gashi and Sylejmani [1] approach and *UniCorT* in terms of: (a) the execution time and (b) the cost of the best solution.

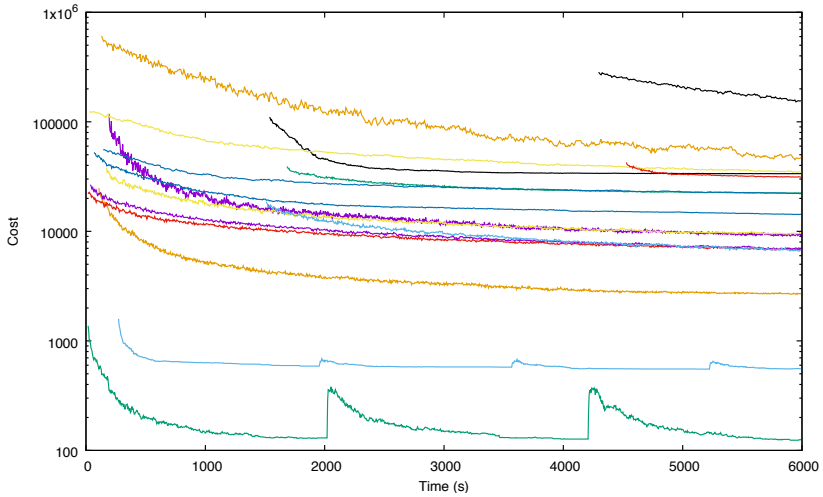


Figure 5.18: Quality of the solution found by Gashi and Sylejmani [1] approach over time.

the three instances only find their first feasible solution after 4 000 seconds. Note that *UniCorT* has only one instance for which the algorithm takes more than 4 000 seconds (see Figure 5.12).

To sum up, *UniCorT* is very effective at finding a solution that does not violate hard constraints. For this reason, *UniCorT* is able to find feasible solutions for all instances within the time limit. Furthermore, our approach is able to find a good solution early on. However, the simulated annealing tool is able to find solutions with better cost if one considers larger time limits as the solver continues to improve the quality of the solution with low memory expenditure. Indeed, we have observed that, given enough time, a feasible solution is found for all the instances. In the worst case, 357.437 seconds are needed to find the first feasible solution.

The organizers of the ITC 2019 made available their solution based on the *UniTime* solver. *UniTime* and *UniCorT* are both able to find a feasible solution within the time limit. Figure 5.19a shows the execution time required to find the best solution. One can see that *UniCorT* finds the best solution faster than *UniTime*. As a matter of fact, our solution is 1.4$\times$ faster than *UniTime*. However, *UniTime* is able to find a solution with better quality, in particular when considering student conflicts. Figure 5.19b compares the cost of the best solutions found by *UniTime* and *UniCorT*. The solution with values on or close to the line (*i.e.* solutions with similar values for both solvers) are solutions to

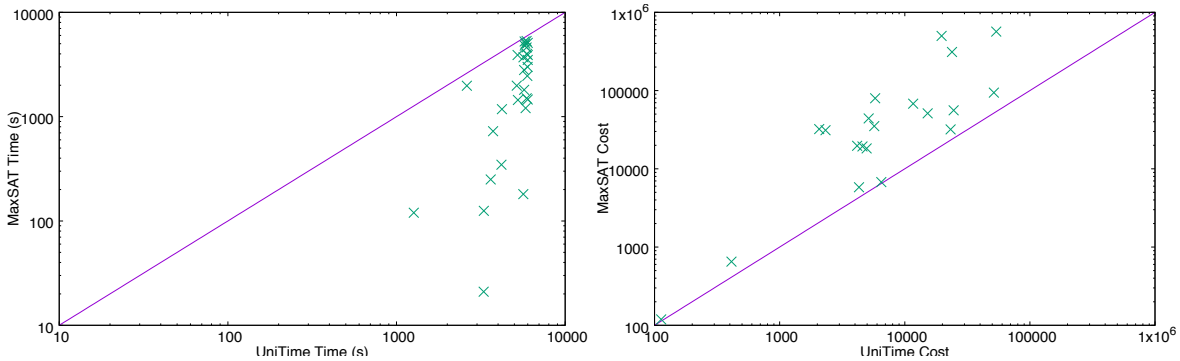


Figure 5.19: Comparison between *UniTime* and *UniCorT* in terms of (a) the execution time and (b) in terms of the cost of the best solution.

Table 5.7: Average distance to best-known solutions for the ITC 2019 benchmark organized by universities. Bold represents a objective where *UniCorT* finds a solution equal to or better than the best-known solution.

	Avg. distance to best know cost			
Family of Instances	**Students**	**Time**	**Room**	**Distribution**
*agh**	32 951	7 914	3 476	4 509
*bet**	5	**-7**	**0**	**0**
*iku**	**0**	**-8**	9	**0**
*lums**	**0**	**-87**	**-76**	1585
*mary**	24 410	666	469	11 877
*muni**	173 759	28 371	5 224	7 096
nbi-spr18	26 148	**-1 664**	95 758	**0**
*pu**	22 640	**-4 321**	**-26**	**-63**
*tg**	**0**	468	**-414**	4 420
yach-fal17	22 820	40	548	6 720
Benchmark Avg	173 759.1	28 371.7	95 758	11 877.5

instances without students.

Table 5.7 compares the distance between the cost of the solution found by *UniCorT* and the best-known values for that instance[7]. The values are separated by objectives. Note that each instance has its own weights. To improve readability, the table is organized by the university. One can see that *StudentConflicts* is clearly the main difficulty for *UniCorT*. The time allocation penalty is not that difficult. This is in part justified by the dynamics of the iterative approach of *UniCorT*. Note that the best known values are obtained in uncertain conditions (*i.e.* without any time and memory limitations).

The *muni** instances are the most difficult for *UniCorT*. This can be explained by the structure of the instances, namely, the number of time slots and the number of possible rooms. Those instances are also the ones that have a larger weight regarding the *StudentConflicts* objective (in the extreme, they have weight 100).

The *mary** instances are the ones with more violations of distribution constraints, even though they are neither the instances with the largest number of constraints nor with the largest weight in this objective. The main characteristic of these instances is the number of distribution constraints per class. In other words, the *mary** instances are among the most over-constrained instances.

[7]Obtained from `https://www.itc2019.org/`, accessed in August 2020.

Table 5.8: Results for the *Invalid Room* disruption. δ_{HD} measures the number of perturbations and δ_{cost} measures the change in the global quality of the solution.

		Invalid Room					
		Avg. Time (s)	Med. Time (s)	Avg. δ_{HD}	Med. δ_{HD}	Avg. δ_{cost}	Med. δ_{cost}
Early	agh-fis-spr17	1 460.4	1 612.7	22	29	42	39
	agh-ggis-spr17	2 321.2	2 210.8	11	8	0	1
	mary-spr17	231.5	253	25	29	54	55
	muni-fi-spr16	2 133.2	2 317.9	15	18	4	6
	muni-fsps-spr17	812	999.1	13	18	0	0
	muni-pdf-spr16c	4 114.8	4 101.2	42	38	26	21
	pu-llr-spr17	142.5	143	35	36	6	6
	tg-fal17	1 208.8	1 247	100	112	18	19
Middle	agh-ggos-spr17	3 212.6	3 212.9	40	40	640	639
	agh-h-spr17	679.9	699.9	19	20	57	60
	lums-spr18	913.9	921.8	18	18	0	0
	muni-fi-spr17	80.8	99	9	13	36	37
	muni-fsps-spr17c	888.4	977.3	39	44	20	20
	muni-pdf-spr16	1 354.5	1 444.1	89	94	1 335	1 336
	nbi-spr18	3 701.7	3 781	14	13	33	35
	yach-fal17	415.56	420	56	66	84	86
Late	lums-fal17	999.9	1 000.2	20	20	0	0
	mary-fal18	788.9	812.1	20	24	40	42
	tg-spr18	813.8	888	5	8	100	100
	muni-fi-fal17	248.9	250.1	9	10	36	30

5.4.6 Minimal Perturbation Problem

UniCorT was compared with an integer programming approach shown in the previous chapter. The ILP approach discussed in the previous section was extended with the constraints (5.7) and (5.8). The results showed that the integer programming approach is able to find the optimal solution for the MPP but only for a subset of instances compared to those solved by the MaxSAT approach. Furthermore, the MaxSAT approach is much faster.

There is no historical data available for the ITC 2019 benchmark. For this reason, we used the disruption profile obtained from the IST analysis shown in the previous chapter. Therefore, the probability of a disruption to occur is extracted from the IST case study. This fact, combined with the constraint format, caused most disruption to have no impact. For example, changing the number of students enrolled in a class does not make the solution infeasible in ITC 2019.

Our approach is able to find feasible solutions to all the disruptions tested. Moreover, the solver is able to find an optimal solution for all disrupted instances. Despite the fact that the disruptions only add new constraints, one can occasionally improve the cost of the solution. This can be explained by the fact that our original solutions are sub-optimal. Otherwise, the new solution could only, in the best case scenario, be as good as the original one.

The results for the disrupted instances with *invalid room* and *invalid time* are shown in Tables 5.8 and 5.9, respectively. The tables show the average and median required CPU time to find an optimal solution, as well as the distance between the two solutions (δ_{HD}) and the change in the global cost (δ_{cost}). It is important to take into consideration that the value of δ_{HD} is directly linked to the size of the instance due to the process of generating disrupted instances.

Table 5.9: Results for the *Invalid Time* disruption. δ_{HD} measures the number of perturbations and δ_{cost} measures the change in the global quality of the solution.

		Invalid Time					
		Avg. Time (s)	Med. Time (s)	Avg. δ_{HD}	Med. δ_{HD}	Avg. δ_{cost}	Med. δ_{cost}
Early	agh-fis-spr17	1 596.22	1 711.1	5 001	5 003	4	6
	agh-ggis-spr17	2 358.2	2 100.4	4	3	0	3
	mary-spr17	381.2	380.1	0	4	0	6
	muni-fi-spr16	1 784.2	1 794.2	16	18	0	0
	muni-fsps-spr17	212.4	218.4	45	46	0	0
	muni-pdf-spr16c	2 992.1	3 001.2	6	6	4	4
	pu-llr-spr17	342.6	356	122	126	10	10
	tg-fal17	1 408.7	1 484	2 021	2 070	25	25
Middle	agh-ggos-spr17	5 465.8	5 466.1	92	93	276	139
	agh-h-spr17	919.1	920.9	97	98	290	289
	lums-spr18	961	978.8	6 446	6 436	0	0
	muni-fi-spr17	40.12	39	144	140	433	423
	muni-fsps-spr17c	500.3	498.8	137	136	0	0
	muni-pdf-spr16	1 035.3	1 030	636	630	6 363	6 364
	nbi-spr18	3 803.8	3 991.1	164	186	3 284	3 289
	yach-fal17	112.56	111	100	100	0	4
Late	lums-fal17	1 085.58	1 100.1	6 777	6 787	0	0
	mary-fal18	800.12	812.1	269	270	807	900
	tg-spr18	933.2	934	568	559	1 704	1 705
	muni-fi-fal17	149.2	140.2	101	108	50	51

Figure 5.20 shows the CPU time per university for the instances with and without disruptions. In most cases, less time is needed to solve a problem instance subject to small disruptions than to solve the original problem instance. If the disruptions cause no perturbations in the original solution, then almost no time is needed (only parsing time). However, our disrupted instances were subjected to significant disruptions. In most cases, the solver is able to find the optimal solution taking around the same time it took to find the best solution without disruption. The time spent to find a solution increases with the number of perturbations required.

As one can see in Figure 5.20, the *invalid room* disruptions are, in most cases, easier to sort out

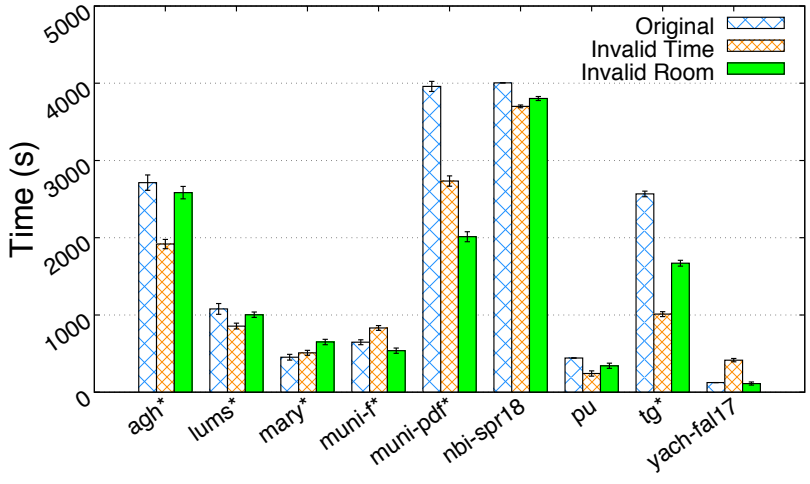


Figure 5.20: A comparison of the CPU time per disruption scenario and university.

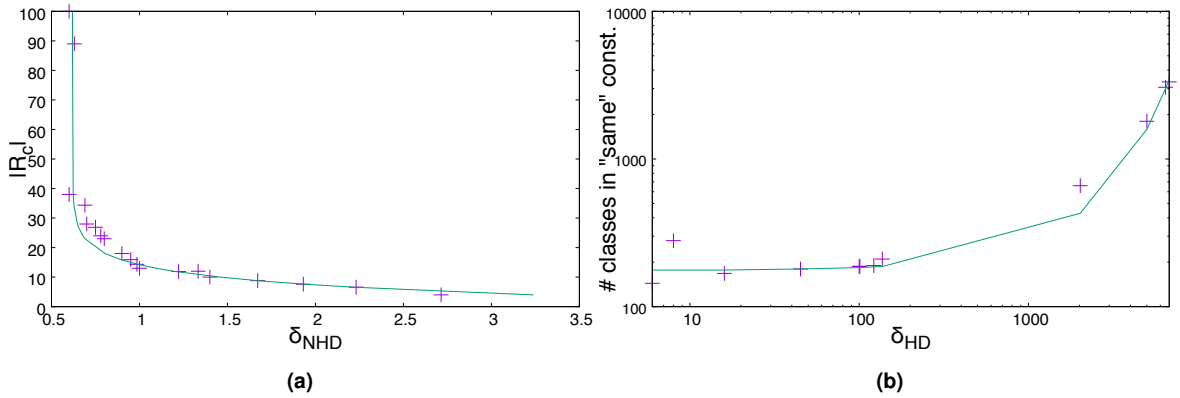


Figure 5.21: (a) Room domain size (R_c) versus the normalized number of perturbations (δ_{NHD}) for the room disruptions. (b) Number of classes involved on constraints of type *same* (log scale) versus the number of perturbations (HD) for the time disruptions (log scale). Data points represent the results and the line the best fit function.

than *invalid time* disruptions. The CPU time is shorter since fewer perturbations are needed. The reduction in time can also be explained by the fact that a smaller number of hard constraints are, in fact, related to rooms. The solutions found are usually closer to the original one. This can be explained by the fact that most instances have fewer rooms than the time slots available.

The *muni-f** instances are, in most cases, the most difficult instances to solve after *invalid room* disruptions. This can be explained by the fact that these instances are very *tight* in terms of room space. On average, these instances only have 4 possible rooms by class versus an average of 14 in the other instances.

In the previous chapter, we used a warm-start to guide the search when solving the MPP. The execution time did not require the use of more advanced methods. However, for the instances of the ITC 2019, the execution time is significant, and one can improve it.

As we are only adding new constraints, all discarded paths in the search tree continue to be discarded. For this reason, we can store the search tree and its decisions. When disruption occurs, we only need to add the new constraints, correct the bounds/ partial solution and continue the search.

This method has a significant advantage in execution time. On average, we reduce the execution time by 10%. We continue to spend more time when the original solution is far from the new optimal solution. This can be explained by the fact that we are taking less advantage of the original search (implying more backtracks).

The incremental method could be further improved using a core guided algorithm (similar to the one discussed in the next chapter). The application of the algorithm is too straightforward, and it has some disadvantages. This problem is discussed in the future work section.

To evaluate the quality of the fittings, the following metrics were defined. Root mean square error (RMSE) has a range from 0 to ∞, where the best fit model has a value closer to zero. Coefficient of determination (CD) has a range between 0 and 1, where the best fit model has a value closer to 1. To perform the fitting, we used the Microsoft Excel Solver [176].

Figure 5.21a shows the relation between the room domain size on δ_{HD}. The RMSE of the fit function is 0.04. The CD is 0.95. Note that, for fairness, we normalized the value of δ_{HD}. The

normalization simply takes into account the number of disruptions generated to the instance (δ_{NHD}).

The *lums** instances are the ones that have the largest δ_{HD} when tested subject to *invalid time* disruptions (see Table 5.9). This fact can be explained by the large number of constraints forcing the classes to be in the same allocation slot (*SameWeek*, *SameTime*, *SameDay* and *SameStart*). These constraints force a chain of perturbations for a single disruption. Figure 5.21b shows the relation between the number of classes involved in constraints of type *Same* and the δ_{HD}. The RMSE of the fit function is 131.8. The CD is 0.11.

We consider the HD as the distance metric for the ITC 2019 benchmark. All our variables are Boolean, and thus the Manhattan distance would have the same result as applying the HD. In the IST case study, we use a metric weighted on the number of students affected (WHD). Recall that this type of metric is essential when solving the mid-semester changes. In the ITC 2019 benchmark, the course timetabling is split from the student sectioning. Our approach solves this problem separately, making the weighted metric harder as we may not know the actual number of students enrolled. Furthermore, it adds a high cost. The quality of the solution decreases by 15%. Table 5.10 shows the comparison of results when considering WHD and HD as optimization criteria. Note that negative values represent results that are better when optimizing WHD. Naturally, optimizing WHD reduces the number of students affected in the instances with students. Optimizing the students involved has a negative impact on the execution time and on the cost. The only instances that are not affected are the ones that do not have students.

Table 5.10: Comparison of results when optimizing WHD or HD.

		Avg. Time Optimizing δ_{WHD} - Time Optimizing δ_{HD} (s)	Avg. δ_{WHD} Optimizing δ_{WHD} - δ_{WHD} Optimizing δ_{HD}	Avg. δ_{cost} Optimizing δ_{WHD} - δ_{cost} Optimizing δ_{HD}
Early	agh-fis-spr17	201	-8206641	800
	agh-ggis-spr17	348.1	-8.464	220
	mary-spr17	10	0	100
	muni-fi-spr16	312	0	4312
	muni-fsps-spr17	33	-164.97	1221
	muni-pdf-spr16c	421.5	-17628	40
	pu-llr-spr17	45	-3296196	5234
	tg-fal17	87	0	0
Middle	agh-ggos-spr17	431	-207.368	1260
	agh-h-spr17	200	-184.3	2910
	lums-spr18	0	0	0
	muni-fi-spr17	20	-216	8121
	muni-fsps-spr17c	121	-41100	12000
	muni-pdf-spr16	222	-1908000	5213
	nbi-spr18	50	-328000	123
	yach-fal17	0	-5000	100
Late	lums-fal17	0	0	0
	mary-fal18	12	-1076269	1230
	tg-spr18	78	0	0
	muni-fi-fal17	12	-2010	2001

5.5 Concluding Remarks

This chapter describes a solution tailored to solve ITC 2019 problem instances. The resulting tool *UniCorT* solves all benchmark instances from the ITC 2019 within the time limit of 6 000 seconds. A

simpler version of this tool was placed among the five finalists. *UniCorT* takes advantage of four pre-processing techniques that are built upon: (i) self-contained sub-instances; (ii) clusters of students; (iii) reducing the classes domain; and (iv) removing the redundant constraints. The first method splits, on average, an instance into 3 sub-instances. Clustering of students reduces the number of variables used, on average, by 23%. The techniques for reducing the class domain reduce in 10% or 4% depending on the number of domain options. Moreover, *UniCorT* is able to remove up to 5% of redundant *StudentConflict* constraints. Finally, the LS method reduces the number of conflicts by 22% without adding significant overhead. In addition, the proposed SAT encoding identifies symmetries when encoding the *MaxBlocks* and *MaxBreaks* constraints, thus reducing the search space.

In order to reduce the size of the problem, and consequently the execution time, *UniCorT* solves the course timetabling and the student sectioning problems separately. This decomposition does not remove any feasible solution. However, it may discard the optimal solution. Still, it allows solving more instances within the time limit. *UniCorT* solves the course timetabling problem iteratively, incrementally adding new time slots for each class, thus reducing the search space. This reduction does not affect the optimal solution.

The experimental evaluation compares *UniCorT* with existing state-of-the-art solutions to solve the ITC 2019. *UniCorT* is, in general, faster than other approaches to obtain a feasible solution. However, the quality of the feasible solution found by *UniCorT* is worse for instances where *StudentConflicts* are strongly penalized.

Moreover, the proposed solution is able to efficiently solve them after the occurrence of the most common disruptions reported in the literature.

6

Train Scheduling

Contents

This chapter describes our different approaches used to solve Train Scheduling Optimization Problems (TSOP) [8]. The TSOP problem can be informally described as the combination of two complementary problems: (i) a routing problem and (ii) a scheduling problem. The goal of the *routing problem* is to find the least cost route for all trains passing through pre-defined stations. The goal of the *scheduling problem* is to assign departure times for each train from each station while minimizing the delay, subject to time and route constraints. To find the optimal solution, one must solve these two problems together. This work was motivated by the 2019 Swiss Federal Railway (SBB) challenge [190].

This chapter is organized as follows. Section 6.1 formally describes TSOP and explains how it relates to PESP. Section 6.2 describes the proposed MaxSAT encoding, and Section 6.3 describes the proposed iterative algorithms. Section 6.4 analyses the evaluation of the different iterative algorithms for both benchmarks. Furthermore, we compare our approach with the current state-of-the-art solutions. Finally, Section 6.5 concludes the chapter.

6.1 Problem Definition

The transport sector is expected to continue to grow over the next three decades [191]. Moreover, the reduction in CO_2 emissions conceded in the Paris Agreement is expected to promote a shift towards the use of collective transport, notably railways. According to the UNIFE World Rail Market Study [192], the railway market growth worldwide will be on average 2.7% between 2021-2023. In particular, the European Union will invest in energy-efficient railways and will continue transforming the railway into a seamless European network[1]. Hence, the size of the railways and the number of trains are expected to grow in the coming years significantly. Now, more than ever, it is important to optimize the schedules of trains efficiently.

Here, we formally define Train Scheduling Optimization Problems (TSOP) and Periodic Event Scheduling Problems (PESP). The definition of the different train based scheduling problems and their relations have been addressed in the past [104, 105]. Therefore, we also address the differences between both problems and propose a mapping from PESP to TSOP. Such mapping allows for PESP and TSOP to be solved using the same approach.

6.1.1 Train Scheduling Optimization Problems

Next, we formalize TSOP based on the description provided by the SBB challenge [109]. A railway network $\mathcal{R}$ is characterized by a graph (V, E) where V is the set of nodes (representing stations, junctions) and E is the set of edges (representing sections of railway tracks with the same characteristics); the marker of a node $mar^v, v \in V$; the minimum travel time of any train in an edge m^e, $e \in E$ (an integer value in seconds); the cost of traveling in an edge pen^e, $e \in E$.

Example 36. Let us consider, the railway network shown in Figure 6.1. The set of nodes is $V = \{v_1, v_3, v_4, v_5\}$ and the set of edges is $E = \{(v_1, v_3), (v_3, v_4), (v_3, v_5)\}$. The markers are mar^{v_1}=*A*,

[1] More information about the European Railway Traffic Management System project is available at http://www.ertms.net/.

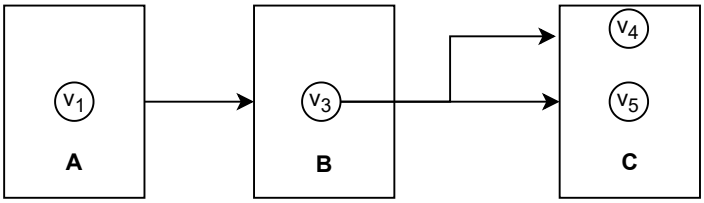


Figure 6.1: Part of the railway shown in Example 2.1, where one train has to go from *A* to *C*. Nodes and edges represent stations and sections of railway track, respectively. The squares represent a junction point.

mar^{v_3}=*B*, and mar^{v_4}=mar^{v_5}=*C*.

Consider a set of train lines T. Each train line $t \in T$ corresponds to the route a train must take. Each train line must be scheduled on the global railway $\mathcal{R}$. Hence, a train line t is characterized by: an acyclic subgraph (V_t, E_t) of (V, E) with all possible routes the train can take; a set of markers M_t representing the stops the train must do; the earliest (latest) time e_t^v (l_t^v) the train may arrive at node $v \in V_t$ (an integer value in seconds); the minimum stopping time of the train on node v is ms_t^v (an integer value in seconds); and the set of connections C_t between train t and other trains.

Considering the earliest/latest arrival time we can define, for each train and node, the entry time domain $\gamma_v^t = [e_t^v, l_t^v]$. Not all nodes have limits on entry time. Furthermore, the entry time limits are only guidelines and not mandatory (soft constraints). In the worst case, $\gamma_v^t = [e_t^{v_i}, l_t^{v_f}]$ where v_i and v_f are the first and last nodes of the train line, respectively.

A route for a train in a train line t is a sequence of connected nodes that pass through all the required nodes (M_t). $\mathcal{P}_t$ is the set of all the possible routes a train can take in the train line t. Each route is denoted by P_t^i, with $i \in [1, \ldots, |\mathcal{P}_t|]$ and $P_t^i \in \mathcal{P}_t$.

Example 37. Recall the train lines shown in Example 36. Let us consider a train line $t \in T$ with $V_t = V$ and $E_t = E$. A train must stop in *A*, *B*, and *C* as they belong to M_t. Hence, there are two possible routes the train can take: {(v_1, v_3),(v_3, v_4)} or {(v_1, v_3),(v_3, v_5)}.

A connection $c \in C_t$ can be of two types: *collision-free* or transfer of passengers/cargo. Let us consider the subset of *collision-free* ($CF_t \subset C_t$) connections. The usage of an acyclic subgraph for the train route removes the possibility of using the same train in the return trip (*e.g.* from *Sanshui S.* to *Bijiang*). For this reason, we define the concept of *collision-free* connection. In a collision-free connection, a train may share an edge/node. Furthermore, we can use these connections to encode the merge/split of trains. The second type is the subset of transfer $CT_t \subset C_t$ connections. Each transfer is characterized by: two train lines t_1 and t_2; two nodes $v_1 \in V_{t_1}$ and $v_2 \in V_{t_2}$ where the transfer will occur; and the minimum connection time is $mct_{t_1,t_2}^{(v_1,v_2)}$.

A **solution** to the TSOP has two components: (i) the assignment of all trains to the route they take through the railway ensuring that the train passes through a set of nodes (routing problem); and (ii) an assignment of entry times to each train at each node on the respective route (scheduling problem). In this work, we consider a complete discretization of time in seconds.

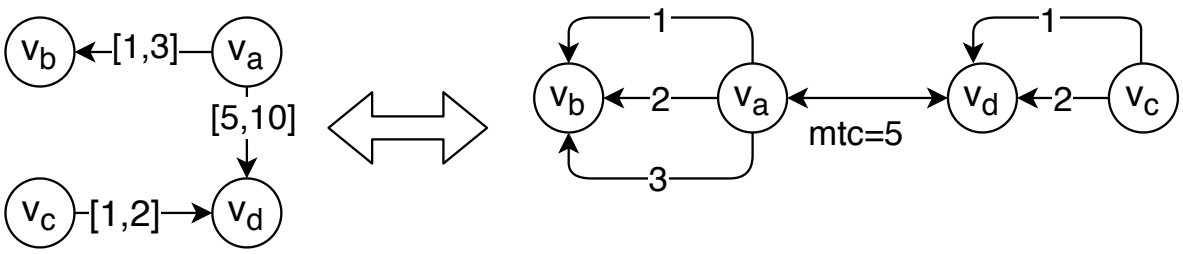


Figure 6.2: On the left a PESP network with 4 events, 3 constraints and $\omega = 40$ [2]. On the right, the PESP network converted into TSOP.

6.1.2 Periodic Event Scheduling Problems

Periodic Event Scheduling Problems (PESP) [40, 193, 194] can be formally defined as follows. Consider a directed graph (V', E'). Each $v \in V'$ corresponds to a periodic event. All events occur with periodicity ω. Each $e \in E'$ corresponds to a relation between events and is characterized by a feasible interval $[L_e, U_e]$, where $L_e(U_e)$ corresponds to a lower (upper) bound. Consider π_v with $v \in V'$ as the starting time of event v. The *goal* is to find $\pi_{v_i}, \pi_{v_j} \in [0, \omega)$ for every $e = (v_i, v_j) \in E$ such that $\pi_{v_j} - \pi_{v_i} modulo\ \omega \in [L_e, U_e]$. Furthermore, we want to minimize the cost given by $\sum_{(v_i,v_j)\in E'} \pi_{v_j} - \pi_{v_i} - L_e$.

Example 38. Figure 6.2 shows an example of a graph for a PESP instance. An optimal solution to this instance is: $\pi_{v_a} = 0, \pi_{v_b} = 1, \pi_{v_c} = 4$ and $\pi_{v_d} = 5$. This solution has a cost of 0, given by $(\pi_{v_b} - \pi_{v_a} - L_{(v_a,v_b)}) + (\pi_{v_d} - \pi_{v_a} - L_{(v_a,v_d)}) + (\pi_{v_d} - \pi_{v_c} - L_{(v_c,v_d)})$.

6.1.3 Converting PESP into TSOP

There are many possible approaches to encode PESP as TSOP. We propose a novel approach, which is described as follows. Different from TSOP, PESP is a cyclic problem. Ergo, we need to break the cycles. For example, the route $\{(a, b), (b, a)\}$ is split into two train lines for the same train, and thus we add a *collision-free* connection. The first train line is a to b and the second is b to a. The train in the second train line must only depart after the train arrives in the first train line. Furthermore, the two trains can be simultaneously in b (the same train).

The constraints of PESP can be divided into two types: (i) the traveling time of a train between two nodes, and (ii) the connection time between two trains. Yet, a PESP instance does not distinguish them (they are all edges). Hence, we must separate the edges in different train routes and connections. To keep all constraints of the original problem, we must ensure all nodes belong to a route. However, there are still multiple possible routes. Hence, we define that a route is the longest set of edges possible while ensuring that all nodes belong to exactly one route. The edges that finish outside the routes correspond to connections between routes.

Consider n trains in the PESP network. The route of a train tr is represented by E'_{tr} with $1 \leq tr \leq n$. V'_{tr} represents all the nodes in the route of the train tr. Furthermore, consider $E'_{con} \subset E'$ as the

set of connections in the network. Let us convert a PESP network (V', E') into a TSOP network (V, E). Recall that each train tr in the TSOP network has a corresponding sub-graph (V_{tr}, E_{tr}). After the conversion, the nodes remain the same ($V_{tr} = V'_{tr}$). The conversion of the edges takes into account the respective interval. Therefore, for each edge $e \in E'_{tr}$ we add e to E_{tr} for each value of $t \in [L_e, U_e]$. The edges are characterized by the minimal traveling time ($m^e = t$) and penalty ($pen^e = t - L_e$).

Now, the only part missing in the TSOP network are the connections. For each edge $(v_i, v_j) \in E'_{con}$ we add a new connection c to the set CT_{tr} where $v_j \in V_{tr}$. The connection c is characterized by two trains tr and tr' such that $v_j \in V_{tr}$ and $v_i \in V_{tr'}$. The lower bound of the interval corresponds to the minimum connection time (mct). The upper bound corresponds to the earliest arrival time in the next node for the first train depart.

Example 39. Let us consider once again the PESP graph shown in Figure 6.2. This graph represents two trains t_1 and t_2 that have a connection. Train t_1 departs from v_a and reaches v_b within the time interval [1,3]. Train t_2 departs from v_c and reaches v_d within the time interval [1,2]. Hence, we can encode the travel time of the trains as two railway graphs as shown in Figure 6.2. Each value of the time interval corresponds to a new edge with a different minimal traveling time (m^e). Finally, we must enforce the arrival of t_2 and the departure of t_1 are within the time interval [5,10]. We encode this constraint as a connection between the train on the node v_a and the train on the node v_d. The $mct_{t_1,t_2}^{(v_a,v_d)}$ is 5. As v_d is the final destination of t_2 we do not need to enforce the earliest arrival time for the next node. This is the only acceptable conversion from PESP into TSOP given that all others would leave independent nodes. The entry time intervals are $\gamma_a^{t_1} = [0]$ (as they have no constraints), $\gamma_b^{t_1} = [1, 3]$ (due to the traveling time), $\gamma_d^{t_2} = [5, 6]$ (due to the connection and the traveling time), and $\gamma_c^{t_2} = [3, 5]$ (due to the traveling time). These are the smallest entry time intervals one can compute without solving the problem.

6.1.4 Disruptions and Recovery

Example 40. Consider the railway shown in Figure 6.3. The traveling time of the train from v_1 to v_3 is 9 minutes, and from v_3 to v_4 / v_5 is 27 minutes. Assume the train cannot leave *B* before 9AM and it has a connection with another train at *C*. This connection requires the train to arrive before 9:42AM to ensure the passengers can switch to another train. Additionally, constraints require the train to stop 5 min in *C* and the minimal traveling time to *Z* is 14 min. The expected time of arrival in *Z* is 10AM. Finally, a disruption causes the edge (v_5, v_8) to be blocked for 4 min. There are three possible solutions to recover from this disruption: (i) wait for the edge to be free; (ii) change path from (v_5, v_8) to (v_4, v_7); and (iii) change path from (v_3, v_5), (v_5, v_8) to (v_3, v_4), (v_4, v_7). The simplest solution is the first one, *i.e.* to wait. However, it causes a delay of 4 min. The second solution requires one change to the train path. This change reduces the delay to 2 min. In this case, the delay is caused by the change of the track. The third solution causes no delay. However, it does require more changes to the train path. These changes can only occur if the train does not have to reach station *C* before the

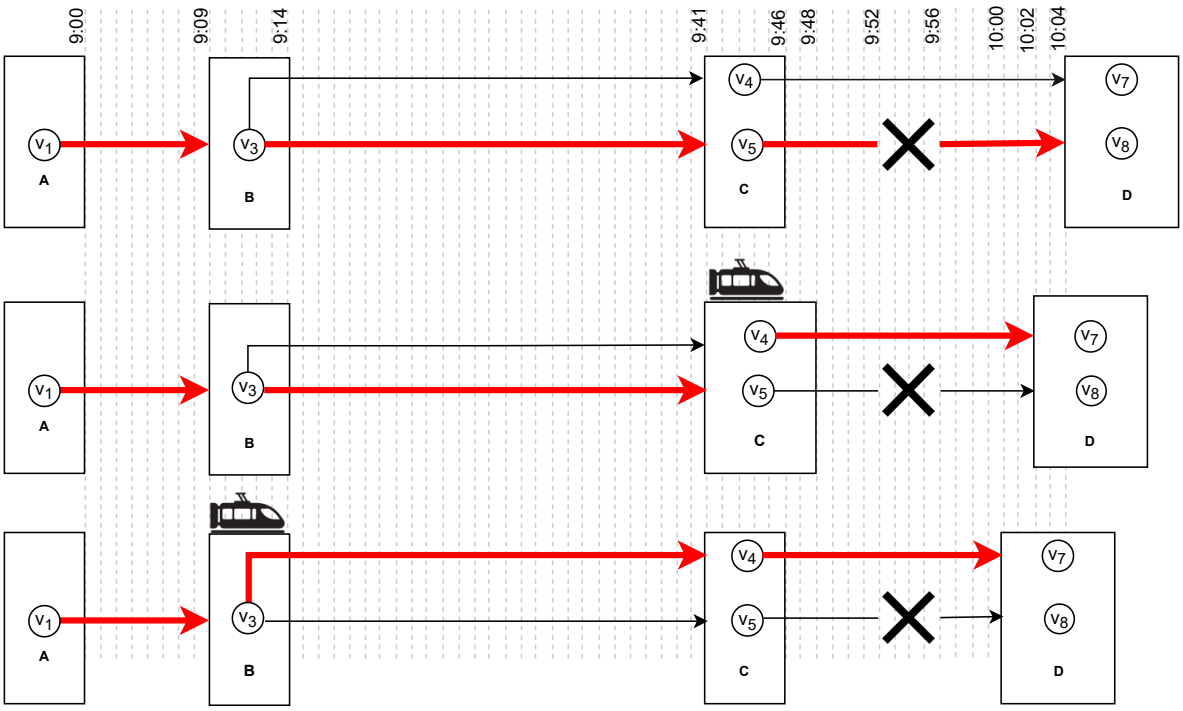


Figure 6.3: Schedule for a train on a railway network after a disruption occurs in the edge (v_5, v_8). The bold red arrows represent the possible solutions. The dashed gray lines represent the time. The size of the rectangle depends on the duration of the train stop in the corresponding station.

disruption is discovered.

Consider s_0 as the original solution to a TSOP problem instance, *i.e.* the route of each train and its schedule. s_0 is characterized by: the scheduled entry time (sen) of each train (t) in each node (v), and the scheduled passage (sp) of each train (t) by an edge (e).

In this work, a disruption can be of three types: slowdown, block track, and block train. Recall their definition in chapter 3.3.5. Each disruption τ, independently of its type, occurs at a specific time $time_\tau$ and has a duration dur_τ. If $time_\tau + dur_\tau \leq min_{v \in V_t, t \in T} sen_v^t$ then we have the freedom to change the whole schedule (we call these types of disruptions *before*). Otherwise, we can only change the schedule of trains for nodes with entry times after $time_\tau$ (*during*).

The *slowdown* disruptions are additionally characterized by the coefficient of velocity $speed_\tau^e$ for a set of edges $e \in E_\tau$. The coefficient represents the reduction of the train velocity in edge e.

Example 41. Consider a train line (a, b) and (b, c). The train departs from a at 9AM and it is expected to arrive in c at 3PM. The traveling time $m^{(a,b)} = m^{(b,c)} = 3hours$. The train is in station b when the weather conditions cause $m^{(b,c)}$ to change to 6 ($speed_\tau^{(b,c)} = 2$). Therefore, the train will arrive 3 hours late. However, there is another track from b to c that is longer but not affected by the weather. Assuming the traveling time is smaller we can re-route the train and reduce the delay.

The *block track* disruptions are additionally characterized by a set of edges $e \in E_{\tau}$ where no train can pass.

Example 42. Consider a train line (a, b) and (b, c). Furthermore, consider that there are two parallel edges between (b, c) referenced here as (b_1, c_1). The traveling time of a train on the track $m^{(b,c)} = 3$ is shorter than on the track $m^{(b_1,c_1)} = 4$. A fallen tree causes track (b, c) to be blocked for 3 hours. Re-routing the train to track (b_1, c_1) reduces the delay of the train.

The *block train* disruptions are additionally characterized by a set of trains $t \in T_{\tau}$ that cannot travel.

Finally, each $e \in E$ now has a cost $taxi_e$ associated with it. This cost represents the financial implication of subsidizing a taxi for passengers traveling on edge e. This value is only considered when a threshold δ of delay is exceeded.

Example 43. Consider a train line (a, b) and (b, c). The train departs from a at 9AM and it is expected to arrive in c at 3PM. The traveling time $m^{(a,b)} = m^{(b,c)} = 3hours$. A strike causes the train to stop for 6 hours. In this thesis, we do not consider the rolling stock problem, and thus we cannot assign a new train. We have neither the data for train capacity (number of passengers) nor the number of trains available (stock) and their location while not working. The only solution is to wait. The train is delayed by 6 hours.

In order to recover from the disruption, we cannot create unplanned stops outside the train path, cancel trains, and miss connections. Hence, the disruption on a train is propagated to all connected trains. We allow the train two stop in any stop along its train path for as long as it is needed. Adding stops outside the path would require a knowledge of the global railway that is not available in the SBB benchmark. The problem with canceling trips is choosing the correct weight. Otherwise, the algorithm will always choose to cancel trips. Nevertheless, in the real world, these are options that traffic controllers use.

6.2 MaxSAT Encoding

Our MaxSAT encoding to solve TSOP, has only two sets of Boolean decision variables:

- r_e^t represents the passage of train t in an edge e, with $t \in T$ and $e \in E_t$;
- $en_v^{t,ti}$ represents the entry of train t in the node v at time ti, with $t \in T$, $v \in V_t$ and $ti \in \gamma_v^t$.

The usage of two types of decision variables can be seen as redundant. However, it causes a reduction in the number and in the size of the clauses. Particularly, they allow us to define routes with fewer clauses. The penalization of routes using only $en_v^{t,ti}$ variables would require an unnecessarily high number of soft constraints. Finally, not all routes need $en_v^{t,ti}$ variables (discussed further on), but most of them need r_e^t variables to identify routes. All possible routes are precomputed along with the constraints used to propagate the train schedule.

6.2.1 Routing Constraints

In this section, we describe the routing constraints of TSOP, establishing that each train must pass through a set of nodes. To ensure that, we add the following exactly one constraint:

$$\sum_{(v_i,v_j)\in E_t, mar^{v_j}=m} r^t_{(v_i,v_j)} = 1 \quad \forall_{t\in T, m\in M_t} \tag{6.1}$$

Now, we only need to connect the dots, *i.e.* create a full route with edges obtained from the last constraint. For this purpose, we add the following clause:

$$\neg r^t_{(v_i,v_j)} \vee \bigvee_{v_k\in V_t,(v_k,v_i)\in E_t} r^t_{(v_k,v_i)} \forall_{t\in T,\,(v_i,v_j)\in E_t} \tag{6.2}$$

The only soft constraint in this part is the penalization of the chosen route. For each $t \in T, (v_i, v_j) \in E_t$ with $pen^{(v_i,v_j)} \neq 0$ we add the unit clause $r^t_{(v_i,v_j)}$ with weight equal to $pen^{(v_i,v_j)}$.

6.2.2 Time Constraints

In this section we describe the scheduling constraints of TSOP, establishing that each train must have exactly one entry time associated with each of the nodes it passes through. To ensure this, we need to define the set O. O is the set of nodes that overlap and thus cannot be visited by the same train (*e.g.* if $(v_i, v_j), (v_i, v_k) \in E_t$ then $v_j, v_k \in O$). $\mathcal{O}$ is a set of Os for the railway (V_t, E_t). Ergo for each $t \in T$ and $O \in \mathcal{O}$ we add:

$$\sum_{v_i\in O} \sum_{ti\in\gamma^{v_i}_t} en^{t,ti}_{v_i} = 1 \tag{6.3}$$

In fact, we do not need to have the entry time for all nodes a train passes through. We can reduce the number of variables by only considering the nodes for which time constraints exist. This change does not remove any solution. Furthermore, the value ti can be restricted to a subset of γ^v_t by checking the time constraints of the train. We can propagate the time constraints and minimal traveling times to reduce the subset (see Example 39). However, the time constraints are soft, and therefore, this method may cause the solution to be infeasible. For this reason, we use an iterative algorithm. At each step, γ^v_t is extended (discussed in the next section).

The relation between two consecutive entry times depends on the traveling time and thus on the route taken by the train. So, we define an auxiliary variable p^i_t that represents the passage of train t on the route i. The relation of this variable to the decision variables is given by:

$$p^i_t \leftrightarrow \bigwedge_{(v_i,v_j)\in P^i_t} r^t_{(v_i,v_j)} \quad \forall_{t\in T, i\in[1..|P_t|]} \tag{6.4}$$

One could use this variable to reduce the number of clauses necessary to encode the route penalty. However, using the decision variable r^t_e directly, even with a larger number of clauses, allows the solver to perform better. This can be explained by the nature of the unit propagation procedure [31, 188] of the solvers.

We still need to ensure that the entry time for two consecutive nodes is consistent. In other words, we need to ensure that the entry time of the second node is equal to the entry time of the first node

plus the travel time on the edge between these nodes and the minimal stopping time of the train. As a result, the following equivalence is added:

$$p_t^i \wedge en_{v_i}^{t,ti} \leftrightarrow \bigvee_{ti' \in [ti+m^{(v_i,v_j)}+ms_t^{v_j} \ldots \gamma_t^{v_j}]} en_{v_j}^{t,ti'} \; \forall_{t \in T,\, ti \in \gamma_t^{v_i},\, i \in [1 \ldots |P_t|],\, (v_i,v_j) \in P_t^i} \tag{6.5}$$

Additionally, another constraint is added to ensure that the two trains, t_1 and t_2, do not collide. Hence, clause 6.6 is added when $\forall t_1, t_2 \in T,\ v_1 = v_2, v_1 \in V_{t_1} \cap V_{t_2},\ (v_1, v_n) \in E_{t_1},\ ti_{t_1} \in \gamma_{t_1}^{v_1},\ ti'_{t_1} \in \gamma_{t_1}^{v_n},\ ti_{t_2} \in \gamma_{t_2}^{v_1},\ ti'_{t_1} - m^{(v_1,v_n)} > ti_{t_2} > ti_{t_1}$[2].

$$\neg en_{v_1}^{t_1,ti_1} \vee \neg en_{v_2}^{t_2,ti_2} \tag{6.6}$$

A conflict-free connection[3] does not require additional constraints. In this case, conflict constraints (two trains in the same place at the same time) are not added for trains with $CF \neq \emptyset$ on the nodes where the connection occurs. The transfer of passengers/cargo between two trains requires the addition of clause 6.6. In this case, we add a clause for every entry time for which the minimum connection time is not guaranteed. In this case, $v_1 \neq v_2$.

The earliest/latest entry time constraints for a node are all considered hard and are ensured by the previous constraints. These constraints are relaxed when needed. When the need arises, we add the following unit clause: $en_v^{t,ti}$ for all trains and nodes for which ti violates the earliest/latest entry constraints. The weight of this constraint is proportional to the delay.

6.2.3 Encoding Disruptions

There are three types of disruptions considered in this work. Next, we describe how these disruptions are encoded into WCNF.

The *train slowdown* disruption τ is encoded adding the equivalence 6.5 with a modified minimal traveling time for an edge ($speed_\tau^e \times m^e$).

The *block train* disruption τ is encoded adding a clause :

$$\neg en_v^{t,ti} \; \forall_{t \in T_\tau, ti \in [time_\tau \ldots time_\tau + dur_\tau], v \in V_\tau} \tag{6.7}$$

Finally, the *block track* disruption τ is encoded adding a clause:

$$\neg en_{v_2}^{t,ti} \vee \neg r_{(v_1,v_2)}^t \;\; \forall_{t \in T_\tau, ti \in [time_\tau \ldots time_\tau + dur_\tau], (v_1,v_2) \in E_\tau} \tag{6.8}$$

For readability, we write the cost function using pseudo-Boolean constraints. This constraints are easily encoded into SAT [35]. The cost function is as follows:

$$\sum_{t \in T} \Bigg[\alpha \times \sum_{v \in V_t, ti \in \gamma_t^v} \left[(|ti - sen_v^t|) \times en_v^{t,ti} \right] \tag{6.9}$$

$$+\beta \times \sum_{e \in E_t} \left[|r_e^t - sp_e^t| \right] + \sum_{(v_i,v_j) \in E_t} \gamma_{(v_i,v_j)} \Bigg]$$

$$\gamma_{(v_i,v_j)} = \begin{cases} 0 & \text{if} (|ti - sen_{v_j}^t|) < \delta \\ taxi_{(v_i,v_j)} & \text{otherwise} \end{cases} \tag{6.10}$$

[2] $ti'_{t_1} - m^{(v_1,v_n)}$ is the exit time of train t_1 at node v_1.

[3] Recall, that this type of connection allows split/merger of trains and changes in direction.

Algorithm 3: *learn* Algorithm

Input: An UNSAT core
Output: A relaxed problem instance

```
for clause ∈ UNSAT do
    for en_{v1}^{t1,ti1} ∈ clause do
        for en_{v2}^{t2,ti2} ∈ clause, ti2 ≤ ti1 do
            if isConnectionConstraint(clause) then
                γ_{t2}^{v2} ∪ ti1 + mct_{t1,t2}^{(v1,v2)};
            else
                if (v1, v2) ∈ E_{t_1} then
                    γ_{t2}^{v2} ∪ ti1 + m_{t1,t2}^{(v1,v2)};
                else
                    γ_{t2}^{v2} ∪ ti1 − m_{t1,t2}^{(v1,v2)};
                end
            end
        end
    end
end
```

Note that α, β are weights for the delay and changes of tracks and δ is the value the passenger is compensated for. We consider the values obtained from the Ministry of Land, Infrastructure, Transport and Tourism [195]. This cost function penalizes the delay locally (by station) and not globally since we know the schedule the train should take. Furthermore, we penalize the change of track by a train and the compensation of the passenger for delays larger than δ.

6.3 Iterative Learning

This section describes the novel iterative algorithms. We use a MaxSAT solver to solve routing and scheduling problems together. In the beginning, the only soft constraints in the problem are the penalization of the route. Observe that the only possible cause of the unsatisfiability are the earliest/latest entry time constraints since they are considered hard, although they may be soft. They are only considered soft after the first UNSAT call.

The most straightforward approach is to relax all time constraints. In other words, we expand the domain of the entry time variables in the instance. In the worst case, we end up with the full domain in all entry time variables. In each iteration, we add 30 new entry time variables (30 seconds) to each node. The number of iterations required in this approach is low, but we may have to deal with a large domain in each iteration. This approach is called *iterative*.

The performance of the algorithm can be improved by only expanding the domain of the variables that are the cause of unsatisfiability. This new approach is called *learn*.

6.3.1 Learning Algorithm

Given an unsatisfiable formula, a SAT solver provides a subset of still unsatisfiable clauses, named the unsatisfiable (UNSAT) core. The UNSAT core may be used to implement a relaxing scheme to

make the formula satisfiable. The different MaxSAT solvers tested use a SAT solver incrementally with assumptions [31]. For this reason, we can extract the UNSAT core from the underline SAT solver.

Learning algorithms have been successfully applied in the past to scheduling problems [4, 11, 196, 197]. In the area of train optimization, Matos *et al.* [11] followed a similar approach to relax the constraints and improve the lower bound estimation in binary search. Here, we are relaxing a problem instance by adding new variables (with an associated penalty) and constraints.

Algorithm 1 shows the pseudocode of the *learn* algorithm, where we start with an UNSAT core. The result is a relaxed instance that is the base for the next iteration. For each clause in the UNSAT core (line 1), we extract the entry time variables. For each two variables (lines 2-3), we compute the number of new variables needed for each node. This computation depends on the type of clause: connection (clause 6.6) or propagation (clause 6.5). If the clause relates to a connection, we update γ with the entry time of the following node plus the minimum connection time. If the clause relates to propagation, both variables relate to the same train, and the update of γ depends on the direction of the edge. We update γ with entry time of the following (previous) node plus (minus) the minimal travel time between them (lines 7-11). In the end, the new domain of the entry time variables in the affected nodes is enough to solve the cause of unsatisfiability. In addition, soft clauses are added to penalize the delay in the train departure time.

Example 44. Consider a train that must travel through two nodes that are connected by one edge (v_1, v_2). The earliest possible entry for the train on v_1 is 1PM. The latest entry time on v_2 is 3PM. The minimal traveling time in (v_1, v_2) is 1 hour. We only need two entry time variables for each node since the possible values for v_1 and v_2 are 1PM/2PM and 2PM/3PM, respectively.

Consider that v_1 is the spot for a connection and that the train must depart after 3PM to ensure the minimum connection time. The problem becomes unsatisfiable. The UNSAT core contains the clause for the connection. The first iteration extends the domain of the entry time for v_1 to accommodate 3PM as possible time (line 5). However, the answer is still unsatisfiable. The new core contains the clauses for connection and the time propagation constraint (v_1, v_2). The next iteration will grow the domain of entry time for v_2 (line 8).

This approach requires more iterations, but each iteration implies fewer changes to the domain. An alternative approach is to predict the next UNSAT core by propagating delays caused by the domain changes. This new approach is called *Learn+Propagation*.

6.3.2 Learning and Propagation Algorithm

The goal is to reduce the number of iterations by propagating the delay through the railway. This algorithm is similar to Algorithm 1, and thus, we are not going to show the pseudocode. At the end of Algorithm 1, we add a new cycle to propagate the changes. For each new variable created, we need to check the impact on the train possible routes. In other words, we need to propagate the delay through the railway. Therefore, new variables are created such that the train may continue to

be delayed on the next nodes. Whenever a new variable is added (corresponding to expanding the entry time domain in a node), the possible delay is propagated. Note that some nodes may already support the delay caused by this procedure.

Example 45. Recall Example 44. The *Learn+Propagation* approach solves the problem instance in just one iteration. The UNSAT core is the same. However, in this case, all delays are propagated. For this reason, we add 4PM as the domain value of the entry time for v_2 at the same time as we add 3PM to the domain of v_1. The solution has a delay of 1 hour since the train enters v_2 at, 4PM.

6.4 Experimental Evaluation

In this section, we discuss the computational results. First, the experimental setup is described. Next, the results of our approaches are discussed and compared with related work.

6.4.1 Experimental Setup

The evaluation was performed on a computer with Fedora 14, with a 2.6 GHz CPU and 128 Gb of RAM. The solver was executed with a time limit of 900 seconds.

Two benchmarks are considered: SBB [3, 109] and PESPlib [111]. The SBB contains 23 real-world instances divided into two data sets: the data set from CrowdAI challenge [109] and the data set from [3]. The main difference is that the first data set does not contain *conflict-free* connections, and therefore not all encodings from the challenge support it. The results were verified by external programs provided by SBB [198] and PESPlib [199].

6.4.2 Generating Disruptions

In the past, the impact of large disruptions on the overall public transportation network has been studied [200–202]. Marra *et al.* [201] used machine learning to identify patterns that impact the number and size of disruptions on the passenger's path. The machine learning method analyzed the real public transportation networks of Zurich. Anagnostopoulos *et al.* [200] studied the systemic influence and fragility of all Swiss train stations to disruptions. The goal was to restructure routes and stations in order to reduce the fragility of the schedule. Anagnostopoulos *et al.* [200] showed that the most influential stations (the ones used by more train routes) are less fragile than remote stations. In this work, we focus on finding the most common disruptions, their characteristics, and their causes, with the goal of creating a realistic benchmark to test our re-solving algorithm.

Figure 6.4 shows the percentage of disruptions per category during 2019 in the Dutch railway network. The disruptions in the Dutch railway network [203] have a direct impact on all railways across Europe. We can see that our model is able to encode 68% of all disruptions that occur in railway networks.

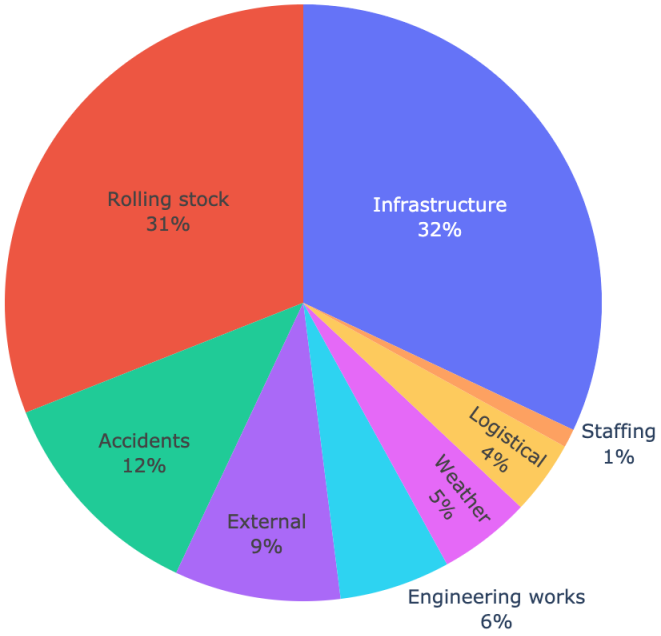


Figure 6.4: Percentage of disruptions per category during 2019 in Dutch railway network. The data was obtained from `rijdendetreinen.nl` (accessed November 2020).

Our data sets and disruption scenarios were obtained from the SBB open data [4]. Figure 6.5 shows the percentage of trains on-time (green), delayed (orange), and canceled (red) for the SBB trains in Switzerland during one year. Furthermore, Figure 6.5 shows the average cumulative delay (in minutes).

We can see that canceled trains are rare and thus were not considered in our solution. The largest delay in the network that occurred in 2019 was 153 minutes. However, we must bear in mind the fact that this delay is cumulative and thus depends on the path of the train. Moreover, the largest delay may be present in the section where the disruption did not occur. In this case, the largest value corresponds to the section where more trains end up (propagated delayed). For this reason, the goal is to find the cause of disruption to generate disruptions correctly.

We analyzed the schedule of all trains in Switzerland during 2019 to find the cause of the delay. Hence, we define the following Bayesian probabilities: $P_{station}$ is the probability of the train t getting delayed on the station v_2 knowing that the train t was on time at v_1 where (v_1, v_2) form an edge; and P_{time} is the probability of the train t getting delayed on the time t_i knowing that the train t was on time at $t_i - 1$.

Figures 6.6 and 6.7a show the values of $P_{station}$ for each station and P_{time} for each time of the day, respectively. The value of $P_{station}$ does not have a large variance for different stations. Nevertheless, we use these values to generate the place where the disruption occurs. On the other hand, one can see that there is a strong relationship between the time of the day and disruptions. Furthermore, we can see three peaks corresponding to the rush hours of early morning, lunch time, and late evening. These are times where most trains get delayed. The closest fit is a multimodal normal distribution, and

[4] The data was extracted from `https://data.sbb.ch/pages/home/`. (accessed November 2020)

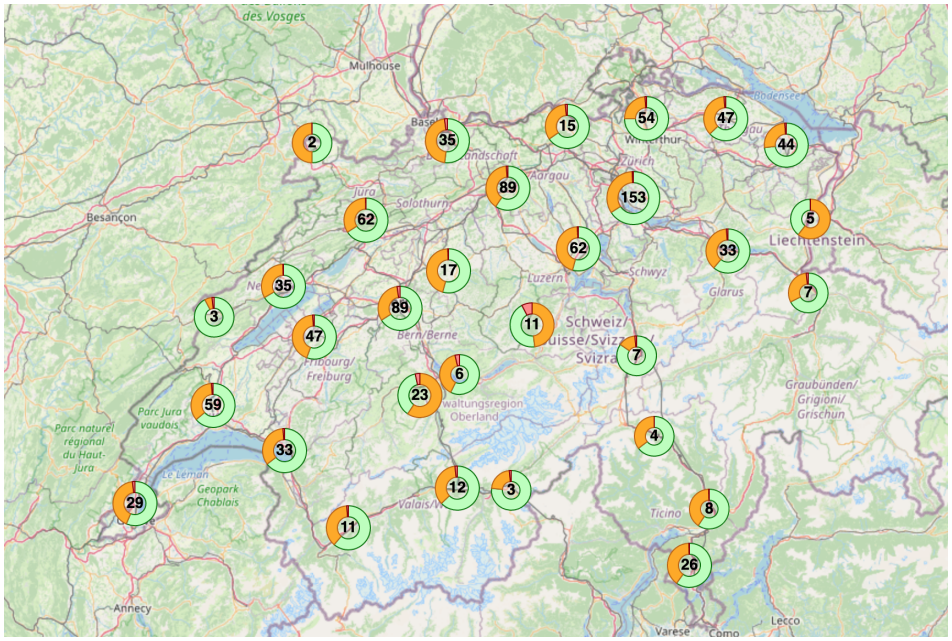


Figure 6.5: The percentage of trains on time (green), delayed (orange) and canceled (red) for the SBB train in Switzerland during 2019. For each section, we show the average cumulative delay (in minutes).

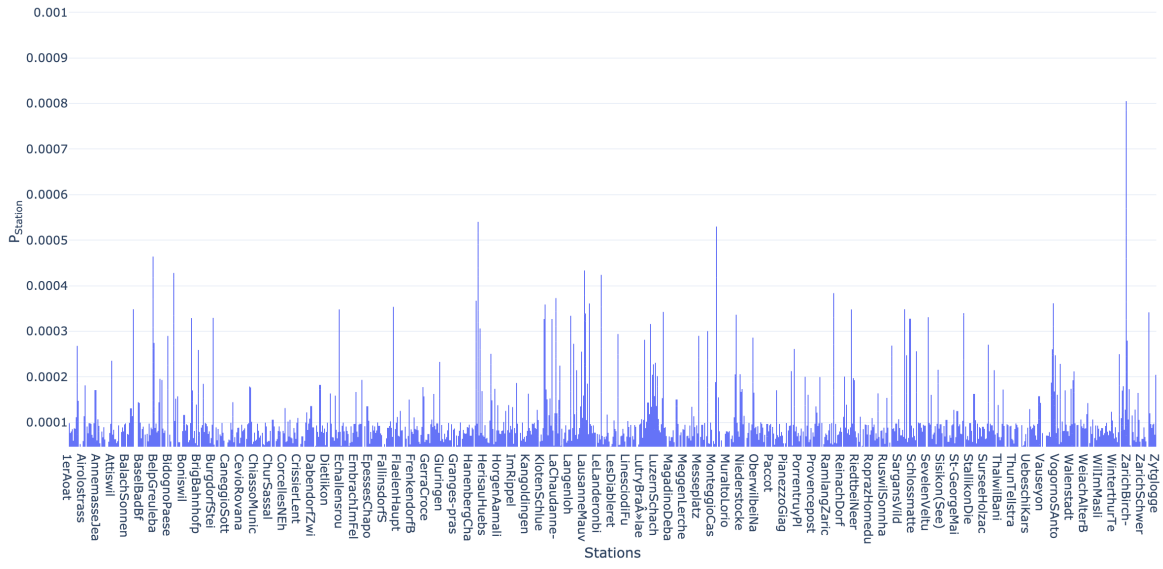
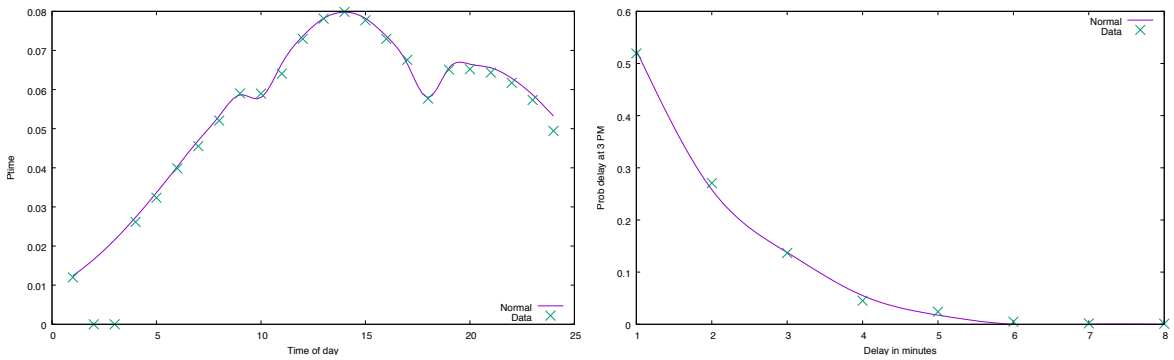


Figure 6.6: The probability of a train getting delayed at each station, knowing that the train was on time at the previous station.



(a) (b)

Figure 6.7: (a) The probability of a train getting delayed at each time of day, knowing that the train was on time at the previous hour. Line corresponds to the multimodal normal distribution that best fits the data sets. (b) Distribution of the duration of the delay disruption at 3PM, knowing that the train is delayed. Line corresponds to the Poisson distribution with the expected rate of occurrences of 1.4 that best fits the data sets of fluctuations delays in minutes.

the coefficient of determination is above 0.99. We used the Microsoft Excel Solver [176] to estimate the parameters.

This information is not enough for generating the full set of disruptions. We still need to generate the duration of the disruption. For this reason, we analyzed the duration of the disruptions that occur at a specific hour of the day. The closest fit is a Poisson distribution for each time of day and the coefficient of determination, on average, is above 0.99. Figure 6.7b shows the distribution of the duration of the delay disruption at 3PM, knowing that the train is delayed. Similar fits can be obtained for other times of the day.

To sum up, we use the probabilities described above to generate 50 different disrupted instances for each type of disruption. The disrupted instance is based on the instances from the SBB data set (23 instances). Therefore, our benchmark with disruptions is composed of two data sets depending on their type (*before*, *during*): the disruptions that occur before the train departure (2 300 instances) and the disruptions that occur after the train departure (3450 instances). Each data set is composed of *block track* disruptions (1150 instances), and *slowdown* disruptions (1150 instances). Additionally, the data set corresponding to disruptions that occur after the train departure has a set of instances with *block train* disruptions (1150 instances).

6.4.3 Computational Evaluation

This section discusses the results of our approach, including a comparison with related work.

Swiss Federal Railway (SBB) benchmark

The characteristics of each instance of the SBB benchmark are shown in Table 6.1. The benchmark with the conflict-free connection has more trains and a larger railway. However, we can see that in both data sets, the percentage of edges with time constraints (TE) is small (around 9%). Therefore, there is a clear advantage of only considering the entry time variables in those nodes.

Table 6.1: #T, #N, #TN stands for the number of trains, nodes, and nodes with time constraints.

SBB Benchmark without collision-free connections									
	P1	P2	P3	P4	P5	P6	P7	P8	P9
# T	4	58	143	148	149	365	467	133	287
# N	318	4 357	8 631	9 323	9 327	38 742	51 807	22 169	34 917
# TN	49	368	932	963	965	2 887	3 700	1 061	2 009

SBB Benchmark with collision-free connections									
	TS	TSE	TL1	TL1E	TL1EI	TL2	TL3	TL4	TL5
# T	131	132	447	448	448	448	448	448	448
# N	12 765	12 870	39 657	39 762	39 762	56 201	56 201	51 207	51 207
# TN	1 124	1 128	3 872	3 876	3 876	3 843	3 843	3 843	3 843
	TL6	TL7	TL8	TL9	TL10				
# T	448	448	448	448	451				
# N	41 156	41 156	64 983	64 983	57 068				
# TN	3 843	3 843	3 843	3 843	3 882				

Comparing MaxSAT solvers

All our iterative approaches rely on a MaxSAT solver. For this reason, the proposed solution is implemented with the top 5 MaxSAT solvers of both complete and incomplete weighted tracks of the 2020 competition [204]. The results show that there are no significant differences between the performance of these MaxSAT solvers. The difference between the worst and the best solver is only 3% more time. The best performing solver is TT-Open-WBO-Inc-20 [205], which is also the winner of the incomplete weight track of the 2020 MaxSAT competition.

TT-Open-WBO-Inc-20 is based on Open-WBO-Inc-complete [206], and actually, these are the two best solvers for these benchmarks.

SATLike-cw [207] uses SATLike [208] until it fails to improve the current solution in a given time limit. After that, uses TT-Open-WBO-inc-20 solver to improve the solutions further. In most instances, the TT-Open-WBO-inc-20 solver is called, and thus takes more time than the TT-Open-WBO-inc-20 natively. The difference between solvers is particularly noticeable for the largest instances (or if we do not reduce the size of the domain of entry time variables). We tried different values for the time limit for the local search procedure with no luck. In the future, we could try parameter tuning using dedicated tools [209].

MaxHS [39] extends the capabilities of SAT and MIP solvers by exploiting both technologies in a hybrid manner. The solver uses CPLEX [38] as a MIP solver and Glucose [210] as a SAT solver.

Note that we do not show the results for the RC2 [211] solver. RC2 is implemented in Python, and thus we implemented a wrapper to use the solver within our C++ implementation. This wrapper makes the comparison unfair as it adds unnecessary overhead to convert the object from one language to the other. Nevertheless, if one compares only the solver time in each iteration, RC2 is still slower than TT-Open-WBO-Inc-20.

There is no particular difference between the solvers of the complete and incomplete tracks. Note that solvers submitted to the incomplete track do not have to be complete, although some of them are if given enough time.

Inst.	TT-Open-WBO-Inc-20 [205]	Loandra [212–214]	Open-WBO-Inc-complete [206]	Open-WBO-Inc-satlike [216]	SATLike-cw [207]	UWrMaxSat [215]	MaxHS [39]
P1	0.14	0.14	0.14	0.14	0.15	**0.12**	0.13
P2	1.5	1.8	1.5	1.3	**1.4**	1.7	1.6
P3	**3**	4	**3**	**3**	**3**	3.8	4
P4	**10**	11	**10**	16	16	14	12
P5	**28**	**28**	**28**	36	36	**28**	**28**
P6	**42**	43	**42**	**42**	**42**	44	46
P7	70	78	70	**69**	70	71	70
P8	32	34	32	32	**31**	**31**	33
P9	**190**	205	194	**190**	**190**	200	**190**
TS	**29**	**29**	**29**	**29**	**29**	**29**	**29**
TSE	**36**	**36**	37	43	43	37	40
TL1	72	73	72	**70**	**70**	72	74
TL1E	**86**	**86**	**86**	99	99	92	95
TL1EI	68	66	68	74	74	67	**65**
TL2	**150**	158	152	158	162	156	153
TL3	149	152	149	145	155	149	**148**
TL4	**99**	102	101	109	110	105	110
TL5	90	92	91	**89**	91	90	**89**
TL6	**71**	**71**	**71**	75	75	75	77
TL7	**68**	**68**	**68**	**68**	69	**68**	**68**
TL8	244	269	245	243	**239**	248	245
TL9	230	260	233	228	**225**	250	255
TL10	**260**	295	262	288	299	262	275
Sum	**2 028.64**	2 161.94	2 039.64	2 107.44	2 129.55	2 093.62	2 107.73

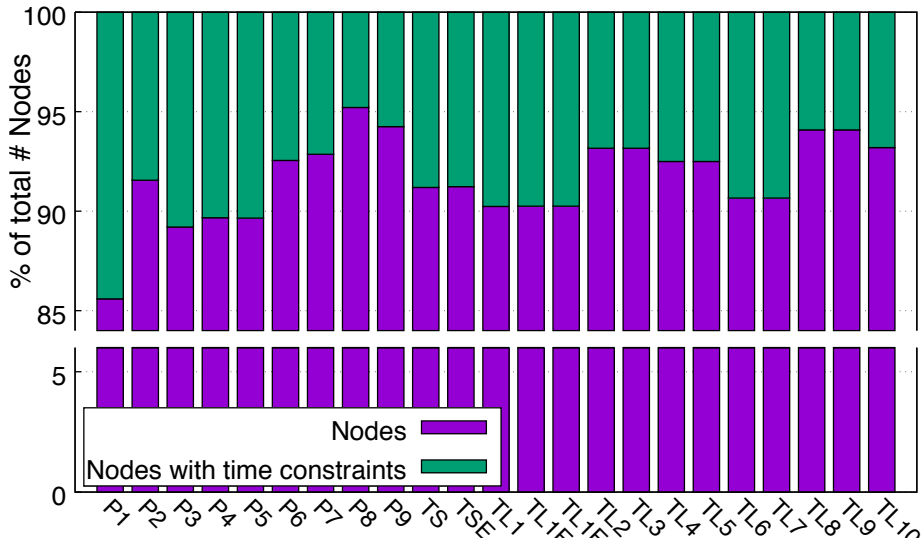


Figure 6.8: The percentage of edges with time constraints for each instance in the SBB benchmark.

Train Scheduling

Figure 6.8 shows the percentage of the total number of nodes that have time constraints. We can see that, on average, only 9% of the total number of nodes deal with time constraints. In the extreme case, concerning the smallest instance, 16% of the nodes have time constraints.

Hence, it is not surprising that the results are better when we only have entry time variables for nodes with time constraints. If we consider the variables in all nodes, we can only solve the instance P1. Still, this is not enough if we do not restrict the domain of the entry variables. Without restricting the domain of the entry variables, there are still 15 timeouts out of 23. The size of the instance causes the timeouts. A similar problem occurred with the *MaxBreak* and *MaxBlocks* constraints in the original version of *UniCorT* (as discussed in the previous chapter).

Restricting the domain of the entry variables allows for solving all instances. However, we need

Table 6.2: The running time in seconds for the different iterative approaches for the instances with optimal cost different from 0. The number of iterations is shown in parentheses.

	Iterative	Learn	Learning and Propagation
P5	116 (**3**)	56 (16)	**28** (4)
TSE	56 (**2**)	42 (5)	**36** (3)
TL1E	224 (**3**)	158 (17)	**86** (5)
TL1EI	171 (**2**)	96 (6)	**68** (**2**)
TL10	Time out (2)	380 (14)	**260** (**7**)

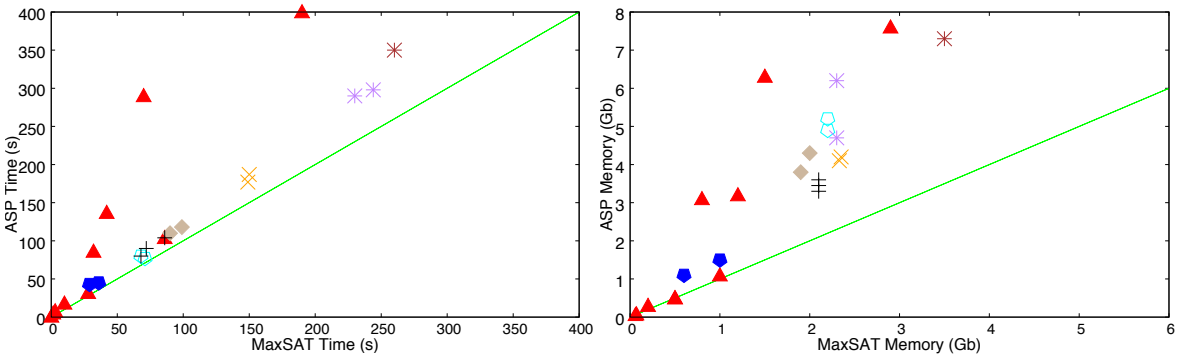


Figure 6.9: Comparison of (a) the running time (in seconds) and (b) the memory consumption (in Gb), between our best solution and the best ASP approach [3] for all SBB data sets. The same symbols/colors symbolize instances with the same overall characteristics. The only exception is the red triangles that represent the P instances and not characteristics.

to iterate to solve instances that have an optimal cost different from 0 (there are only 5). The running time can be improved.

Table 6.2 shows the results of the different iterative algorithms for the 5 instances. The iterative algorithm is the one with fewer iterations needed but expands the domain unnecessarily, which causes a time out in TL10. With the addition of the learning process, we can see clear progress in terms of running time. The best algorithm is the *Learn and Propagation*. The main reason is the fact that we avoid doing unnecessary iterations. Each call to the solver has an overhead, which may not be necessary if one can predict the problem. On average, the first call is really fast since it is easy to prove the unsatisfiability of the instance. However, the calls get slower each time the domains are increased.

Table 6.3 compares the performance of the proposed solution with the related work on the SBB Crowd Sourcing Challenge benchmark. As mentioned above, the ASP solution uses an approximation cost function to deal with the size of the problem. However, this process may remove the optimal solution. To make a fair comparison, we changed the ASP solution [3] to use an exact cost function. To this solution we call *ExactASP*. Table 6.4 compares the performance of the proposed solution with the related work on the SBB benchmark with collision-free connections. Recall that not all approaches allow collision-free connections.

ExactASP is slower than the approximated version but can prove the optimal solution for all instances but P9. The main difference is in memory consumption. The approximated version can solve all instances and find the optimal solution for most of them. The difference between the approximated

Table 6.3: The results for the SBB benchmark without conflict-free connections. T, M, C stand for running time in seconds (s), memory in gigabytes (Gb), and cost. ExactASP was adapted from [3] to have an exact cost function.

	ExactASP			ASP [3]			Greedy [217]			MaxSAT			ILP [218]		
	T(s)	M(Gb)	C	T(s)	M(Gb)	C	T(s)	M(Gb)	C	T(s)	M(Gb)	C	T(s)	M(Gb)	C
P1	1	0.2	**0**	1	**0.06**	**0**	1	0.1	**0**	**0.14**	**0.06**	**0**	0.7	**0.06**	**0**
P2	8	0.2	**0**	5	0.08	0	4	0.15	**0**	**1.5**	**0.07**	**0**	43	0.5	**0**
P3	18	0.7	**0**	8	0.3	**0**	8	0.72	**0**	**3**	**0.2**	**0**	94	2.2	**0**
P4	38	1	**0.1**	18	0.5	**0.1**	13	0.8	**0.1**	**10**	**0.5**	**0.1**	141	3	0.8
P5	64	1	**33**	32	0.5	**33**	501	1.75	37.3	**28**	**0.5**	**33**	671	5.1	237.6
P6	317	5.1	**0**	137	3.2	**0**	44	1.56	**0**	**42**	**1.2**	**0**	661	7.4	**0**
P7	580	14	**0**	290	6.31	**0**	91	1.8	**0**	**70**	**1.5**	**0**	899	9.24	**0**
P8	142	4.9	**0**	86	3.1	**0**	**31**	1.3	**0**	32	**0.8**	**0**	250	6.8	1.7
P9	TO	TO	TO	400	7.6	**0**	360	**2**	**0**	**190**	**2**	**0**	TO	TO	TO

and the real optimal is a few seconds of delay for most instances. However, this approach cannot solve the 4 largest instances of the SBB benchmark with collision-free connections.

The greedy [109, 217] approach is one of the fastest (the winner of the challenge). It is characterized by keeping the memory low (despite being implemented in Java) and yet finding a solution with a good cost. The shortcoming lies in the backtracking procedure. Instances P5 and P9 are the only instances that require backtracking and, therefore, more time.

ILP [109, 218] can prove optimality for all instances but P9. However, the decomposition removes the actual optimal solution. The choice of routes in the first stage reduces the search space but it also removes the actual optimal. Choosing the route has a direct impact on the overall cost of the solution.

Our approach can solve faster and with less memory than any other solution. The main difference lies in the iterative approach with the smallest domain at the beginning. Furthermore, it is the only approach that solves the exact problem.

Figure 6.9a compares the running time (in seconds) of our best solution and the best ASP approach [3] for the SBB benchmark. We can see that the MaxSAT approach is faster than the ASP counterpart for all instances. The MaxSAT approach is on average twice as fast as the ASP approach even with the approximation.

Figure 6.9b compares the memory consumption between our best solution and the best ASP approach [3] for SBB data sets. We can see that the great advantage obtained from the iterative nature of MaxSAT is memory handling. Notice that the line on 2Gb is full of instances with the same number of nodes, resources, and trains. The only difference lies on the size of the domain of the entry time variables. The conclusion is that the iterative approach enables memory reduction.

Periodic Event Scheduling

The characteristics of each instance of the PESP benchmark are shown in Table 6.5. Furthermore, we compare the number of variables and constraints of our approach with and without the pre-processing step with Matos *et al.* [4]. One can clearly see that our approach requires fewer variables and constraints (discussed later on).

To the best of our knowledge, there is no state-of-the-art tool publicly available able to solve PESP. Even though there are many SAT approaches in the literature, we choose to compare Matos *et al.* [4] for the following reasons. First, the proposed approach is self-contained. In other words, the approach

Table 6.4: The results for the SBB benchmark with conflict-free connections. The execution time is in seconds (s), and the memory consumption is in gigabytes (Gb). ExactASP is adapted to have an exact cost function.

	ExactASP			ASP			MaxSAT		
	T (s)	M (Gb)	C	T (s)	M (Gb)	C	T (s)	M (Gb)	C
TS	108	2,2	**0**	43	1.1	**0**	**29**	**0.6**	**0**
TSE	113	2,2	**6**	45	1.5	**6**	**36**	**1**	**6**
TL1	225	6,6	**0**	90	3.3	**0**	**72**	**2.1**	**0**
TL1E	260	6,6	**11**	104	3.45	**11**	**86**	**2.1**	**11**
TL1EI	200	6,6	**11**	80	3.6	**11**	**68**	**2.1**	**11**
TL2	468	10	**0.1**	187	4.9	**0.1**	**150**	**2.2**	**0.1**
TL3	443	10	**0**	177	5.2	**0**	**149**	**2.2**	**0**
TL4	275	10	**0.5**	110	4.3	1	**99**	**2**	**0.5**
TL5	278	10	**0**	111	3.8	**0**	**90**	**1.9**	**0**
TL6	193	10	**0.2**	77	4.2	**0.2**	**71**	**2.3**	**0.2**
TL7	-			80	4.1	**0**	**68**	**2.3**	**0**
TL8	-			298	4.7	**0**	**244**	**2.35**	**0**
TL9	-			290	6.2	**0**	**230**	**2.35**	**0**
TL10	-			350	7.6	10.75	**260**	**2.9**	**10.13**

does not require the implementation of customizing heuristics and pre-processing methods. Second, the approach only uses SAT and does combine multiple tools. Third, the description of the SAT encoding is precise enough to be replicated and implemented.

We solve the PESP benchmark after converting it to the TSOP format. Furthermore, the size of the network is reduced by removing unnecessary nodes *a priori*. A similar approach was proposed by Borndörfer *et al.* [40]. The proposed approach is able to reduce, in 35% and 12%, the size of the networks of the R and the BL instances, respectively. This pre-processing step improves the quality of the solution by 9% for the R instances.

Figure 6.10 compares the cost of the solution found by both MaxSAT approaches and the current best-known values for each instance. None of these values are known to be optimal. Note that we do not know the time and memory limits for which these values were found. Furthermore, not all values were found by the same tool. The best solution thus far is produced by a concurrent tool specifically designed to solve PESP [40]. They reduce the problem with pre-processing. The solving process is split into three phases: a SAT solver, ILP solver and specific heuristics to guide the overall search. The SAT solver is only used to warm-start the ILP solver. We are not able to find the optimal solution within the time limit. Our solution matched the current best-known cost for BL instances, which have specific characteristics. Regarding the rest of the benchmark, we are within 30% of the current best value.

The best SAT-based approach was proposed by Matos *et al.* [4, 11]. The characteristics of this approach were already discussed. Figure 6.11 compares the running time required to find the best solution using our MaxSAT approach with Matos *et al.* approach.

We are able to improve the quality of the solution by 22% and still reduce the running time, on average, by 214 seconds. This can be explained by the smaller size of our encoding. Matos *et al.* [4] encoding requires, on average, 1.4$\times$ more variables and 1.2$\times$ more constraints. This is due to the way Matos *et al.* [4] encodes cycles and constraints. They use $q_{x,i}$ variables meaning that event x

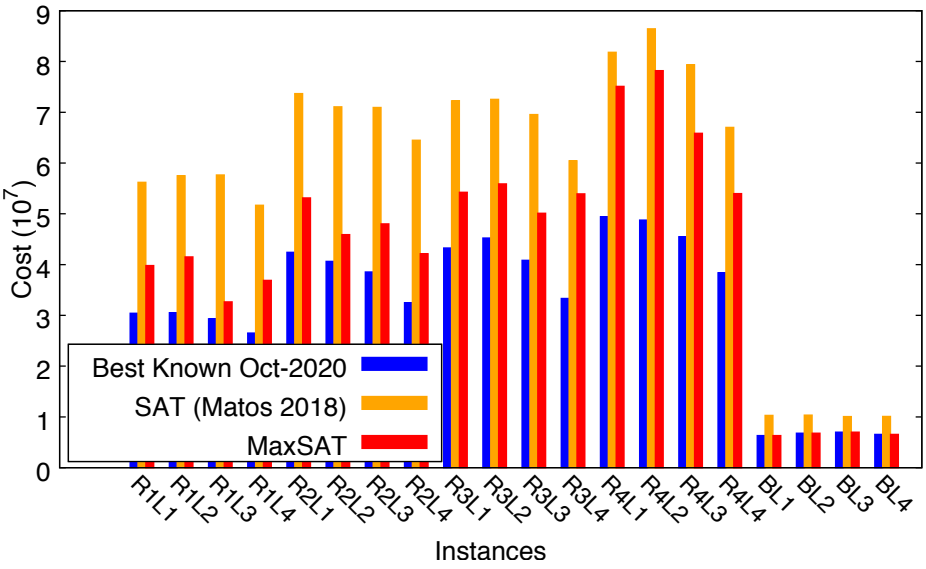


Figure 6.10: Comparison of the cost found by both MaxSAT approaches and the current best-known values for each instance.

starts no later than the time i. However, the usage of this decision variable requires the clauses to have a larger arity. The arity of the clauses is known to have a significant impact on SAT solver performance [188].

Recovering from Disruptions

In this section, we discuss the performance of our algorithm to recover from three different disruptions. Our algorithms can solve all disrupted instances with an optimal solution in less than 400 seconds. Recall that the disruption cannot lead to a solution with a better cost than the original one.

Slowdown When considering the disruption (*e.g.* bad weather) that causes a train to slow down, it is difficult to recover from the delay as the disruptions normally affect most edges. The only way to recover is to route through a faster path after or before the affected edges. However, this is rare since the original timetable is already fine-tuned and not robust. Recall that most nodes had an interval where the delay was acceptable. The goal was to reduce the entry time ti ensuring $ti \in \gamma_v^t$ (*i.e.*, without arriving too early) for each train t in each node n. Recall that the maximum entry time of the node in normal conditions is l_t^v. We can recover from this disruption without exceeding the maximum delay (l_t^v) in 90% instances. When considering the *during* disruptions, we can recover by spending, on average, only more 16 seconds to solve the original problem. When considering the *before* disruption, the algorithm takes an additional 120 seconds, on average. This can be explained by the fact that there is no need to solve the whole problem when solving the disruption while the train is traveling. On average, the number of variables is reduced by 30%.

Block track This is the only disruption that we can really improve by re-routing since there are other options. Naturally, the capacity of re-routing depends on the number of parallel routes. Figure 6.12

Table 6.5: #Nodes, #Edges, #Var, #Const stands for the number of nodes, edges, variables, and constraints. The direct and compact encodings are versions of the same model without and without the pre-processing step.

Instance			Matos et al. [4]		This Paper			
					Direct		Compact	
Name	# Nodes	# Edges	# Var (k)	# Const. (k)	# Var (k)	# Const. (k)	# Var (k)	# Const. (k)
R1L1	3 664	6 386	508	17 268	366	12 700	366	11 557
R1L2	3 668	6 544	530	18 163	385	12 607	385	11 346
R1L3	4 184	7 032	539	18 202	387	13 618	387	12 256
R1L4	4 760	8 529	688	23 749	504	19 440	504	17 496
R2L1	4 156	7 362	585	20 203	429	15 825	429	14 242
R2L2	4 204	7 564	606	21 096	449	15 892	449	14 303
R2L3	5 048	8 287	615	20 795	444	16 987	444	15 288
R2L4	7 660	13 174	990	34 762	744	29 868	744	26 881
R3L1	4 516	9 146	785	28 410	605	24 032	605	21 629
R3L2	4 452	9 252	808	29 356	625	24 978	625	21 481
R3L3	5 724	11 170	933	33 646	717	29 267	717	23 999
R3L4	8 180	15 658	1 284	46 172	986	41 793	986	37 614
R4L1	4 932	10 263	888	32 663	696	28 284	696	22 627
R4L2	5 048	10 755	940	34 763	741	30 384	741	24 307
R4L3	6 368	13 239	1 135	42 079	898	37 700	898	30 160
R4L4	8 384	17 755	1 534	57 005	1 218	52 627	1 218	45 259
BL1	2 688	7 988	536	10 702	329	6 323	329	6 260
BL2	2 606	7 488	504	10 201	304	5 822	304	5 764
BL3	3 044	9 311	603	11 855	389	7 477	389	7 402
BL4	3 816	13 502	764	13 807	595	9 429	595	9 335

shows the execution time (in seconds) for the best algorithm to solve original instances and to recover from disruptions of the *block track* type that occur before and during the travel of train. For the smaller instances, there is no difference between solving the original instances or the disrupted instances. In general, we need more iterations than in the original search to recover since there will always be a larger delay. This is the main reason for the execution time of disrupted instances to be worse. When considering the *during* disruptions, the number of iterations is compensated by the reduction in the size of the instance. This can be explained by the fact that we significantly reduce the size of the problem since we cannot change it anymore. Naturally, this depends on the location of the train when the disruption occurs. This explains the size of the error bars for the *during* disruption.

Block train This disruption can be used to partially model crew problems, as we can block the train due to insufficient staff. This disruption causes a delay, which is quite difficult to recover from. Similar to the train *slowdown* disruption, re-routing the train does not improve the delay in most cases. This is the disruption that requires more iterations of the *learning and propagation* algorithm to recover. For this reason, this is also the disruption that takes longer to solve. Figure 6.13 shows the time spent to recover after different types of disruptions occur. *Block track* is the fastest disruption to recover as fewer iterations are needed and there is more freedom to change the path of the train. These changes allow reducing the number of iterations.

In the worst case, the recovery procedure takes more than 62% of the total execution time. However, we can reduce this value solving the problem incrementally.

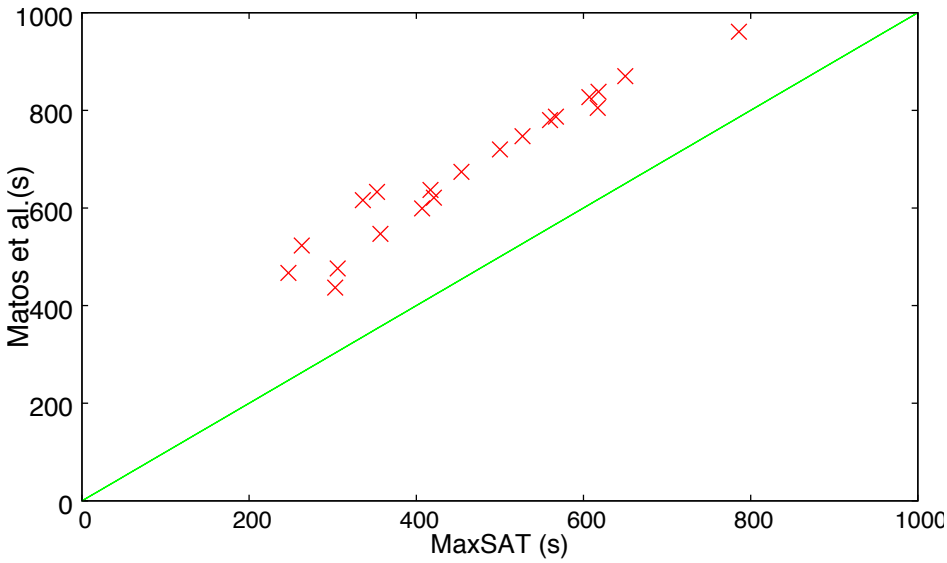


Figure 6.11: Comparison of the running time to find the best solution between our MaxSAT approach and Matos *et al.* [4].

Incremental versus "from scratch" The recovery algorithm spends most of its time solving the original problem again. For this reason, we can reduce this time with two techniques: (i) setting the polarity of the variables; and (ii) solving the problem incrementally.

Setting the polarity [219] of the variables is similar to the warm-start technique used in ILP. Its impact depends on the similarity of the original solution and the new optimal solution. This technique has a larger impact when no re-routing is needed.

Solving the problem incrementally requires saving the state of the search and then restarting the search when the disruption occurs. We only add new constraints, and therefore all the constraints learned are still valid. However, we need to change the cost function and update all the lower and upper bounds used in the search. Consequently, the algorithm may have explored a path in the search tree that is no longer relevant. This *per se* is not a problem but may have a small impact in the incremental search.

Figure 6.14 shows the number of instances solved with the respective execution time (in seconds) for each method. We can see that setting the polarity of the variables has a small impact. This impact is only effective for smaller instances with a shorter execution time. Also, the incremental solving has more impact than setting the polarity of the variables. Nevertheless, the impact is almost null for instances that require more iterations in the *learning and propagation* algorithm and more changes to the path. On average, the gain of using incremental solving is of 25%.

The performance improvements do not affect the quality of the found solution. The solution found has exactly the same cost as before. The procedure stops only when the optimal solution is found.

6.4.4 Results Overview

Our incremental algorithm is able to solve all data sets optimally for SBB. The algorithm allows solving the exact problem with no approximation. Furthermore, it will avoid memory problems by only

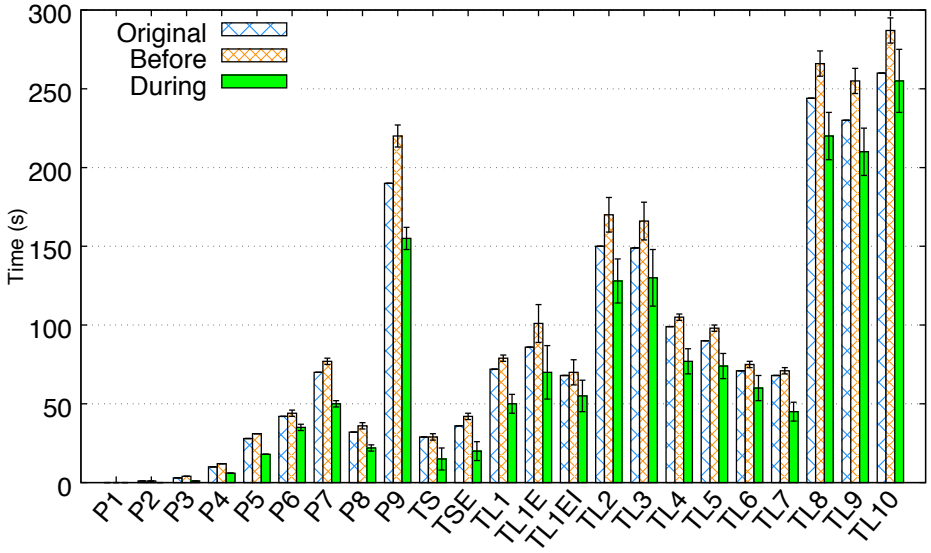


Figure 6.12: Comparison between execution times to find the original solution and to recover from *before* and *during* disruptions of *block track* type.

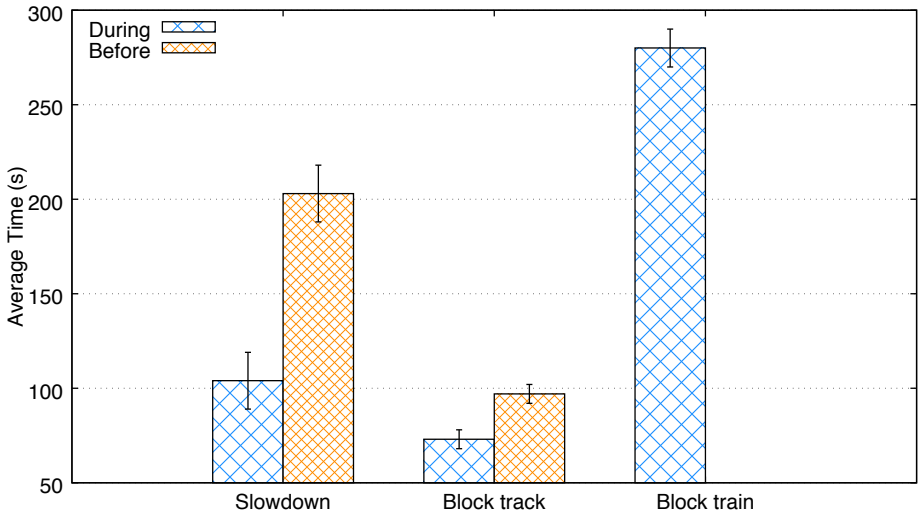


Figure 6.13: The average recovery time by type of disruptions.

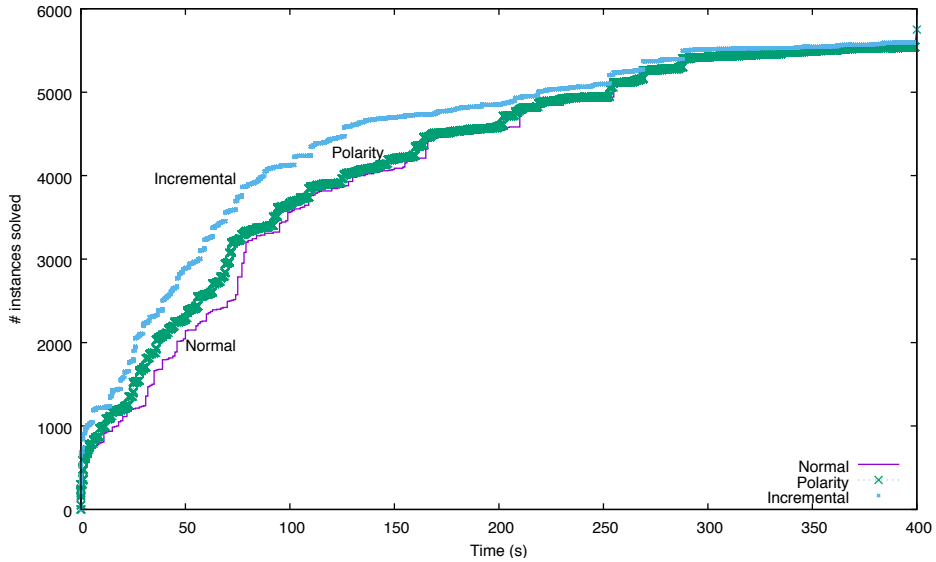


Figure 6.14: The number of disrupted instances solved as a function of the execution time (in seconds) for each method.

Table 6.6: Overall comparison of the best approaches to solve TSOP.

	ASP	This thesis
Proves Optimiality	NO	YES
Average Distance to Optimal Solution	1.12	0
Average Time (sec)	145.8	118
Max Time (sec)	350	260
Average Memory (Gb)	45	1.9
Max Memory (Gb)	7.3	2

using the smallest domain.

The algorithm also re-solves the most common disruption that occurs in train schedules. Note that we could easily add the disruptions missing if enough data were available (*e.g.* rolling stock). The incremental version of the algorithm, on average, reduces the execution time by 25%.

The novel conversion process allows solving PESP instances efficiently, in particular to the subset instances of BL. For these instances, we are able to match the best-known value. However, we cannot prove optimality. For all other PESP instances, we are within 30% of the current best value.

6.5 Concluding Remarks

This paper proposes a novel iterative MaxSAT encoding to solve the train scheduling optimization problem that takes advantage of the relaxing of the problem. We also propose a conversion between PESP and TSOP. The proposed approach was validated using the SBB and the PESPLib benchmarks.

The experiments show that the optimal solution is found for all SBB instances within 260 seconds, being on average twice as fast as the ASP counterpart, while avoiding the exponential growth of the memory requirements. Moreover, all PESPLib instances are solved within 786 seconds. These results are a considerable improvement when compared with current MaxSAT solutions. The quality

of the solution is improved by 22%, and the running time is reduced, on average, by 32%. Still, the quality of the solution is, on average, 30% away from the current best-known solution for the instance.

Furthermore, our algorithm is able to recover from disruptive scenarios. We encode into CNF 68% for all different disruptions that occur in railways. The algorithm was tested using randomly based disruptions generated based on the real-life disruptions of 2019 that occur in SBB.

7

Conclusions

Contents

Solving scheduling problems under disruptions [46, 131, 135, 136] has real-world applications that range from universities [44, 45, 47, 87, 132] to transport [49, 111, 151, 154, 157, 161]. There are two main approaches to solve scheduling problems under disruptions: (i) creating robust solutions; and (ii) solving the MPP. Robust solutions are usually imperfect as they are sub-optimal because they have to support disruptions that may never occur. Regardless of the solution's robustness, we may need to solve a MPP. This can guide us to the second approach. However, in the literature, the approaches to solve the MPP perform worse than when solving the original problem. The algorithms proposed in this thesis can be easily adapted to different scheduling problems.

This work advances the state-of-the-art on solving scheduling problems, in particular on solving MPP in the context of university course timetabling and train scheduling. The proposed work was evaluated and compared with state-of-the-art approaches. The evaluation was performed with benchmarks from different competitions of university timetabling and train scheduling. In order to validate the application of the proposed methods, in a real setting, we used real-world data sets and disruptive scenarios. The results show that we can improve the performance of the re-solving procedure by either solving the problem incrementally or only re-solving a small part of the problem (where the disruption occurs). Furthermore, we compare the impact of different encodings and pre-processing techniques on the quality and performance of exact methods. This chapter is organized as follows.

In this thesis, we used different types of distance metrics. When considering university course timetabling it is common to consider only the Hamming distance. This distance metric has the distinct advantage of being easy to encode and it has small impact on the performance of the tool.

After talks with the academic offices of IST, we defined metrics of real-world usage (number of students and compactness). With these metrics, the proposed algorithm can solve problem instances to optimality. However, these type of metrics (particularly Hamming distance weighted with the number of students) do not result well for the ITC 2019 benchmark. Note that, at IST, we already know the students enrolled before generating the timetable. In ITC 2019, we have to section students at the same time. This detail causes the WHD metric to have a significant impact on the performance. Furthermore, reducing the the value of WHD has a considerable impact on the overall quality of the solution. For this reason, the solution found could end up being worse for the students/teacher rather than optimizing only the HD.

When considering the train scheduling state of the art, the type of distance metrics is richer. We proposed a distance metric that is a composition of all these metrics. The overall metric is complex and adds a significant overhead to the encoding. Nevertheless, our incremental algorithm is able to efficiently solve the problem and to reduce the overhead caused by the distance metric being used.

We showed the importance of decomposition and pre-processing methods to solve large problem instances using exact methods such as SAT and ILP. Unfortunately, these methods are domain-dependent and therefore difficult to extrapolate to different scenarios.

We showed considerable advantages when considering the MPP by using warm start methods in both ILP and MaxSAT (variable polarity). The application of ILP and MaxSAT depends on the structure of the problem. Even though, in this thesis, SAT proved more efficiently in practice, this cannot be

generalized. Empirical evidence shows that a SAT solver is stronger when we have binary clauses and implications, while ILP is stronger when solving Pseudo-Boolean and linear constraints. It would be interesting to see if the new hybrid solver could exploit this.

Finally, we showed that solving incrementally the MPP is quite efficient. The storage of the search tree, variable order, and the previous decision are important to reduce the execution time. Naturally, this process is less efficient when the solution is further away from the initial solution. Not only the distances affect the performance, but also the type of changes affects the performance. For example, it is easier to solve when we only add new constraints, as we do not need to revisit previous UNSAT branches. In the context of the train scheduling problem, we showed that we could further improve the performance by guiding this incremental process as a result of learning from the inconsistencies in the solution.

Section 7.1 summarizes the principal contributions of this thesis. Section 7.2 describes the possible directions for future work.

7.1 Contributions

First, we focused on solving the university course timetabling problem at our university, IST. In order to tackle the problem, we started by profiling and cleaning the data. The result of this process was supplied to IST services to improve the data quality of the stored data [165]. As we suspected, one of the major problems in IST is the lack of room space and as the saying goes “space, like time, is money”. Therefore, we started by implementing three different algorithms to improve the current handmade timetable. The ILP formulation and the greedy algorithms with provable performance guarantees were published in [15].

Next, this work proposed to change the formulation to solve the whole timetabling problem of IST. Afterward, we also considered disruptions. To validate our approach, we studied the disruptions that occurred in the last 5 years in IST. With these results, we developed an algorithm to solve MPP efficiently. This algorithm divided the re-solving problem into two sub-problems: the assignment of classes to rooms and the assignment of classes to time slots. Depending on the type of disruption, we started by considering only one of these problems. In the end, we could find the optimal solution faster than conventional methods. This work was published in [16].

The next step was to generalize this approach to all universities. For this reason, we proposed a new tool, *UniCorT*, to solve the International Timetabling Competition (ITC) 2019. The solution finished in the top 5. *UniCorT* used different pre-processing techniques to improve quality and performance. The early results of *UniCorT*, at the time the competition ended, were published in [17]. Afterward, *UniCorT* continued to improve due to new incremental algorithms, and those results are currently under submission. Finally, the application of *UniCorT* to the MPP was published in [18].

The last contribution was a MaxSAT based encoding to solve and re-solve train scheduling problems after disruptions occur. To improve the performance on the SBB data sets, we employed an iterative learning algorithm that only relaxes the arrival time of the train when needed. Furthermore,

we created an incremental algorithm to re-solve the problem after disruptions occur. This incremental algorithm resulted in a better performance than traditional algorithms.

7.2 Future Work

This section describes possible future research work directions. The goal is to improve the performance of the re-solving methods by new pre-processing techniques and incremental algorithms. The idea of solving the problem covered in this thesis as a multi-criteria optimization problem would be interesting as it could bring a more balanced and fair result. In order to further detail the future work avenues, we need to consider the context of each application studied in this thesis. Therefore, this section is organized as follows. Sections 7.2.1 and 7.2.2 discuss possible avenues of research in terms of solving university course timetabling problems and train scheduling problems, respectively.

7.2.1 University Course Timetabling

UniCorT can be improved by creating a new encoding to represent student conflicts with the goal of reducing the memory consumption and thus improving the performance. This can be achieved through graph representation, for example. Besides, one can try to exploit symmetries to reduce the search space for the student sectioning problem.

The iterative algorithm used to solve the instances of ITC 2019 can also be improved. We can apply a similar procedure to the one used for train scheduling (core guided). However, this problem is harder to learn with the UNSAT core. There is no precise mapping between the constraint and the root of the problem (room assignment or time assignment). Nevertheless, we could learn from the cores to guide the search. We would only know which classes actually need to have new time slots. However, it would add a significant overhead to predict how many time slots would be required. Note that the available time slots for each class are not sequential and have different weights.

The algorithm to solve MPP in *UniCorT* can also be improved. We can improve the incremental algorithm with the benefits from the core-guide approach. When solving the MPP we could tune to take advantage of the decomposition already applied. The decomposition allows solving only a portion of the problem where the disruption occurred. We can guide the search of the incremental solver only to change a part of the problem.

7.2.2 Train Scheduling

The algorithms used to solve train scheduling can also be improved. For example, we can add new pre-processing techniques to improve the performance of the iterative algorithms. Indeed, the pre-processing techniques are the base of the success of *UniCorT*. Exploring more pre-processing techniques would be particularly interesting to improve the quality of results obtained in the PESP benchmark. In addition, we could relax the number of possible parallel routes and add them on demand. In other words, in each iteration, we would consider longer routes.

The algorithms to solve train scheduling problems under disruptions can also be improved. We can try to decompose the problem and only solve the portion of the problem where the disruption occurs.

Furthermore, our approach only considers 68% of the possible disruptions. We can extend the encoding and the algorithm in order to solve staffing [164] and rolling stock problems.

Bibliography

[1] E. Gashi and K. Sylejmani, “Simulated annealing with penalization for university course timetabling,” in *Proceedings of the International Timetabling Competition 2019*, 2020.

[2] P. Grossmann, S. Holldobler, N. Manthey, K. Nachtigall, J. Opitz, and P. Steinke, “Solving periodic event scheduling problems with SAT,” pp. 166–175, 2012.

[3] D. Abels, J. Jordi, M. Ostrowski, T. Schaub, A. Toletti, and P. Wanko, “Train scheduling with hybrid answer set programming,” p. 1–31, 2020.

[4] G. P. Matos, “Optimisation of periodic train timetables,” Master’s thesis, Instituto Superior Técnico, Portugal, 2018.

[5] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer, 2012.

[6] J. W. Herrmann, *A History of Production Scheduling*. Springer, 2006, pp. 1–22.

[7] M. Boddy and D. Johnson, “Automated finite capacity scheduler,” Mar. 17 2009, uS Patent 7,505,827. [Online]. Available: https://www.google.com/patents/US7505827

[8] R. Borndörfer, T. Klug, L. Lamorgese, C. Mannino, M. Reuther, and T. Schlechte, Eds., *Handbook of Optimization in the Railway Industry*. Springer, 2018.

[9] A. Bonutti, F. D. Cesco, L. D. 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. [Online]. Available: https://doi.org/10.1007/s10479-010-0707-0

[10] T. Müller, “Constraint-based timetabling,” PhD dissertation, Charles University in Prague Faculty of Mathematics and Physics, 2005.

[11] G. P. Matos, L. M. Albino, R. L. Saldanha, and E. M. Morgado, “Solving periodic timetabling problems with SAT and machine learning,” 2020.

[12] E. Merhej, S. Schockaert, and M. D. Cock, “Repairing inconsistent answer set programs using rules of thumb: A gene regulatory networks case study,” *International Journal of Approximate Reasoning*, vol. 83, pp. 243–264, 2017.

[13] A. Lemos, I. Lynce, and P. T. Monteiro, “Repairing boolean logical models from time-series data using answer set programming,” *Algorithms for Molecular Biology*, vol. 14, no. 1, Mar. 2019.

[14] F. Gouveia, I. Lynce, and P. T. Monteiro, “Semi-automatic model revision of boolean regulatory networks: confronting time-series observations with (a)synchronous dynamics,” *bioRxiv*, 2020.

[15] A. Lemos, F. S. Melo, P. T. Monteiro, and I. Lynce, “Room usage optimization in timetabling: A case study at Universidade de Lisboa,” *Operations Research Perspectives*, vol. 6, p. 100092, 2019.

[16] A. Lemos, P. T. Monteiro, and I. Lynce, “Disruptions in Timetables: A Case Study at Universidade de Lisboa,” *Journal of Scheduling*, 2020.

[17] A. Lemos, P. T. Monteiro, and I. Lynce, “ITC 2019: University Course Timetabling with MaxSAT,” in *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling (PATAT) 2021: Volume I*, 2020, pp. 129 – 146.

[18] A. Lemos, P. T. Monteiro, and I. Lynce, “Minimal perturbation in university timetabling with maximum satisfiability,” in *Proceedings of 17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, vol. 12296, 2020, pp. 317–333.

[19] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence. Elsevier, 2006, vol. 2.

[20] M. Ehrgott, *Multicriteria Optimization (2. ed.)*. Springer, 2005. [Online]. Available: https://doi.org/10.1007/3-540-27659-9

[21] C.-L. Hwang and A. S. M. Masud, *Multiple objective decision making—methods and applications: a state-of-the-art survey*. Springer Science & Business Media, 2012, vol. 164.

[22] H. Greenberg, *Integer programming*. Academic Press, 1971.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2009.

[24] T. Achterberg, T. Berthold, T. Koch, and K. Wolter, “Constraint integer programming: A new approach to integrate cp and mip,” in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, L. Perron and M. A. Trick, Eds. Springer, 2008, pp. 6–20.

[25] R. M. Karp, *Reducibility among Combinatorial Problems*. Springer, 1972, pp. 85–103. [Online]. Available: https://doi.org/10.1007/978-1-4684-2001-2_9

[26] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[27] H. D. Sherali and W. P. Adams, “Reformulation-linearization techniques for discrete optimization problems,” in *Handbook of combinatorial optimization*. Springer, 1998, pp. 479–532.

[28] J. P. Marques-Silva and K. A. Sakallah, “GRASP - a new search algorithm for satisfiability,” in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, R. A. Rutenbar and R. H. J. M. Otten, Eds. IEEE Computer Society / ACM, 1996, pp. 220–227.

[29] R. J. Bayardo Jr. and R. Schrag, “Using CSP look-back techniques to solve real-world SAT instances,” in *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI) and Ninth Innovative Applications of Artificial Intelligence Conference (IAAI)*, B. Kuipers and B. L. Webber, Eds. AAAI Press / The MIT Press, 1997, pp. 203–208.

[30] M. Davis and H. Putnam, “A computing procedure for quantification theory,” *J. ACM*, vol. 7, no. 3, pp. 201–215, 1960.

[31] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*. IOS press, 2009, vol. 185.

[32] R. Martins, V. M. Manquinho, and I. Lynce, “Open-WBO: A modular MaxSAT solver„” in *Theory and Applications of Satisfiability Testing (SAT) - 17th*, 2014, pp. 438–445.

[33] A. Nadel, “TT-Open-WBO-Inc: Tuning polarity and variable selection for anytime SAT-based optimization,” in *Proceedings of the MaxSAT Evaluations*, 2019.

[34] S. Joshi, P. Kumar, R. Martins, and S. Rao, “Approximation strategies for incomplete MaxSAT,” in *Principles and Practice of Constraint Programming (CP)*, 2018, pp. 219–228.

[35] N. Eén and N. Sörensson, “Translating pseudo-boolean constraints into SAT,” *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, no. 1-4, pp. 1–26, 2006.

[36] C. Ansótegui and J. Gabàs, “Solving (weighted) partial maxsat with ILP,” in *Proceedings of 10th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, ser. Lecture Notes in Computer Science, C. P. Gomes and M. Sellmann, Eds., vol. 7874. Springer, 2013, pp. 403–409.

[37] J. Davies and F. Bacchus, “Exploiting the power of mip solvers in maxsat,” in *Proceedings of 16th International Conference on the Theory and Applications of Satisfiability Testing (SAT)*, ser. Lecture Notes in Computer Science, M. Järvisalo and A. V. Gelder, Eds., vol. 7962. Springer, 2013, pp. 166–181.

[38] IBM ILOG, “Optimization studio CPLEX user’ s manual, version 12 release 8,” 2017.

[39] R. Hickey and F. Bacchus, “Speeding up assumption-based SAT,” in *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, ser. Lecture Notes in Computer Science, vol. 11628. Springer, 2019, pp. 164–182.

[40] R. Borndörfer, N. Lindner, and S. Roth, “A concurrent approach to the periodic event scheduling problem,” p. 100175, 2019.

[41] A. Kasirzadeh, M. Saddoune, and F. Soumis, “Airline crew scheduling: models, algorithms, and data sets,” *EURO Journal on Transportation and Logistics*, vol. 6, no. 2, pp. 111–137, Feb. 2015.

[42] B. McCollum, “University timetabling: Bridging the gap between research and practice,” in *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. Springer, 2006, pp. 15–35.

[43] A. Elkhyari, C. Guéret, and N. Jussien, “Solving dynamic resource constraint project scheduling problems using new constraint programming tools,” in *Proceeding of 4th International Conference of the Practice and Theory of Automated Timetabling (PATAT)*, 2002, pp. 39–62.

[44] A. E. Phillips, C. G. Walker, M. Ehrgott, and D. M. Ryan, “Integer programming for minimal perturbation problems in university course timetabling,” *Annals of Operations Research*, vol. 252, no. 2, pp. 283–304, 2017.

[45] T. Müller, H. Rudová, and R. Barták, “Minimal perturbation problem in course timetabling,” in *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, 2004, pp. 126–146.

[46] H. El Sakkout, T. Richards, and M. Wallace, “Minimal perturbation in dynamic scheduling,” in *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*. John Wiley & Sons, 1998.

[47] M. Lindahl, T. Stidsen, and M. Sørensen, “Quality recovering of university timetables,” *European Journal of Operational Research*, vol. 276, no. 2, pp. 422 – 435, 2019.

[48] R. M. Lusby, J. Larsen, and S. Bull, “A survey on robustness in railway planning,” *European Journal of Operational Research*, vol. 266, no. 1, pp. 1 – 15, 2018.

[49] W. Fang, S. Yang, and X. Yao, “A survey on problem models and solution approaches to rescheduling in railway networks,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 2997–3016, 2015.

[50] E. V. Andersson, “Assessment of robustness in railway traffic timetables,” Ph.D. dissertation, Linköping University Electronic Press, 2014.

[51] A. L. Lovelace, “On the complexity of scheduling university courses,” Master’s thesis, California Polytechnic State University, San Luis Obispo, 2010.

[52] B. Herres and H. Schmitz, “Decomposition of university course timetabling,” *Annals of Operations Research*, 2019.

[53] S. Even, A. Itai, and A. Shamir, “On the complexity of timetable and multicommodity flow problems,” *Society for Industrial and Applied Mathematics Journal on Computing*, vol. 5, no. 4, pp. 691–703, 1976.

[54] T. Müller, “ITC-2007 solver description: a hybrid approach,” *Annals of Operations Research*, vol. 172, no. 1, p. 429, 2009.

[55] S. Casey and J. Thompson, “Grasping the examination scheduling problem,” in *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. Springer, 2002, pp. 232–244.

[56] C. Gogos, P. Alefragis, and E. Housos, “An improved multi-staged algorithmic process for the solution of the examination timetabling problem,” *Annals of Operations Research*, vol. 194, no. 1, pp. 203–221, 2012.

[57] A. D’Ariano, M. Pranzo, and I. A. Hansen, “Conflict resolution and train speed coordination for solving real-time timetable perturbations,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 208–222, 2007.

[58] T. Müller, H. Rudová, and Z. Müllerová, “University course timetabling and International Timetabling Competition 2019,” in *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, E. K. Burke, L. Di Gaspero, B. McCollum, N. Musliu, and E. Özcan, Eds., 2018, pp. 5–31.

[59] L. Di Gaspero, A. Schaerf, and B. McCollum, “The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3),” Queen’s University, Tech. Rep., 2007.

[60] R. Lewis, B. Paechter, and B. McCollum, “Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition,” Cardiff Business School, Cardiff Working Papers in Accounting and Finance A2007/3, 2007.

[61] R. A. O. Vrielink, E. A. Jansen, E. W. Hans, and J. van Hillegersberg, “Practices in timetabling in higher education institutions: a systematic review,” *Annals of Operations Research*, vol. 275, no. 1, pp. 145–160, 2019.

[62] L. T. G. Merlot, N. Boland, B. D. Hughes, and P. J. Stuckey, “A hybrid algorithm for the examination timetabling problem,” in *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, E. Burke and P. De Causmaecker, Eds. Springer, 2003, pp. 207–231.

[63] H.-J. Goltz and D. Matzke, “University timetabling using constraint logic programming,” in *PADL*. Springer, 1999, pp. 320–334.

[64] T. Müller, “Itc 2019: Preliminary results using the unitime solver,” in *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling (PATAT) 2021: Volume 1*, 2020.

[65] M. Atsuta, K. Nonobe, and T. Ibaraki, “ITC-2007 track 2: an approach using a general CSP solver,” in *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, 2008, pp. 19–22.

[66] 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*, vol. 2. IEEE, 2005, pp. 55–60.

[67] M. Banbara, K. Inoue, B. Kaufmann, T. Okimoto, T. Schaub, T. Soh, N. Tamura, and P. Wanko, “$teaspoon$: Solving the curriculum-based course timetabling problems with Answer Set Programming,” *Annals of Operations Research*, vol. 275, no. 1, pp. 3–37, 2019.

[68] P. M. Bittner, T. Thum, and I. Schaefer, “SAT encodings of the at-most-k constraint - A case study on configuring university courses,” in *Proceedings of the Software Engineering and Formal Methods (SEFM)*, 2019, pp. 127–144.

[69] E. Demirović and N. Musliu, “MaxSAT-based large neighborhood search for high school timetabling,” *Computers & Operations Research*, vol. 78, pp. 172–180, 2017.

[70] R. J. Asín Achá and R. Nieuwenhuis, “Curriculum-based course timetabling with SAT and MaxSAT,” *Annals of Operations Research*, vol. 218, no. 1, pp. 71–91, 2014.

[71] D. S. Holm, R. Ø. Mikkelsen, M. Sørensen, and T. R. Stidsen, “A mip based approach for international timetabling competation 2019,” in *Proceedings of the International Timetabling Competition 2019*, 2020.

[72] E. Rappos, E. Thiémard, S. Robert, and J.-F. Hêche, “International timetabling competition 2019: A mixed integer programming approach for solving university timetabling problems,” in *Proceedings of the International Timetabling Competition 2019*, 2020.

[73] G. Lach and M. E. Lübbecke, *Optimal University Course Timetables and the Partial Transversal Polytope*. Springer, 2008, pp. 235–248.

[74] X. Feng, Y. Lee, and I. Moon, “An integer program and a hybrid genetic algorithm for the university timetabling problem,” *Optimization Methods and Software*, vol. 32, no. 3, pp. 625–649, 2017.

[75] A. E. Phillips, H. Waterer, M. Ehrgott, and D. M. Ryan, “Integer programming methods for large-scale practical classroom assignment problems,” *Computers & Operations Research*, vol. 53, pp. 42 – 53, 2015.

[76] V. Cacchiani, A. Caprara, R. Roberti, and P. Toth, “A new lower bound for curriculum-based course timetabling,” *Computers & Operations Research*, vol. 40, no. 10, pp. 2466–2477, 2013.

[77] M. Aschinger, S. Applebee, A. Bucur, H. Edmonds, P. Hungerländer, and K. Maier, *New Constraints and Features for the University Course Timetabling Problem*. Springer, 2018, pp. 95–101.

[78] H. Vermuyten, S. Lemmens, I. Marques, and J. Beliën, "Developing compact course timetables with optimized student flows," *European Journal of Operational Research*, vol. 251, no. 2, pp. 651 – 661, 2016.

[79] E. K. Burke, J. Marecek, A. J. Parkes, and H. Rudová, "Decomposition, reformulation, and diving in university course timetabling," *Computers & Operations Research*, vol. 37, no. 3, pp. 582–597, 2010.

[80] Y. Sun, X. Luo, and X. Liu, "Optimization of a university timetable considering building energy efficiency: An approach based on the building controls virtual test bed platform using a genetic algorithm," *Journal of Building Engineering*, p. 102095, 2020.

[81] A. Rezaeipanah, S. S. Matoori, and G. Ahmadi, "A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search," *Applied Intelligence*, Aug. 2020.

[82] E. Burke, D. Elliman, and R. Weare, "A genetic algorithm based university timetabling system," in *Proceedings of the 2nd east-west international conference on computer technologies in education*, vol. 1, 1994, pp. 35–40.

[83] H. E. Nouri and O. B. Driss, "Distributed model for university course timetabling problem," in *IEEE International Conference on Computer Applications Technology (ICCAT)*, 2013, pp. 1–6.

[84] H. E. Nouri and O. B. Driss, "MATP: A multi-agent model for the university timetabling problem," in *Software Engineering Perspectives and Application in Intelligent Systems - Proceedings of the 5th Computer Science On-line Conference (CSOC2016), Vol 2*, 2016, pp. 11–22.

[85] E. Özcan, M. Misir, G. Ochoa, and E. K. Burke, "A reinforcement learning - great-deluge hyper-heuristic for examination timetabling," *International Journal of Applied Metaheuristic Computing*, vol. 1, no. 1, pp. 39–59, 2010.

[86] K. Er-rhaimini, "Forest growth optimization for solving timetabling problems," in *Proceedings of the International Timetabling Competition 2019*, 2020.

[87] A. Gülcü and C. Akkan, "Robust university course timetabling problem subject to single and multiple disruptions," *European Journal of Operational Research*, vol. 283, no. 2, pp. 630 – 646, 2020.

[88] P. Fernandes, C. S. Pereira, and A. Barbosa, "A decision support approach to automatic timetabling in higher education institutions," *Journal of Scheduling*, vol. 19, no. 3, pp. 335–348, 2016.

[89] T. Song, S. Liu, X. Tang, X. Peng, and M. Chen, “An iterated local search algorithm for the university course timetabling problem,” *Applied Software Computing*, vol. 68, pp. 597–608, 2018.

[90] E. H. Kampke, W. de Souza Rocha, M. C. S. Boeres, and M. C. Rangel, “A GRASP algorithm with path relinking for the university courses timetabling problem,” *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, vol. 3, no. 2, pp. 1–7, 2015.

[91] W. de Souza Rocha, M. Claudia, S. Boeres, and M. C. Rangel, “A GRASP algorithm for the university timetabling problem,” in *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, 2012, pp. 404–406.

[92] A. V. Moura and R. A. Scaraficci, “A GRASP strategy for a more constrained school timetabling problem,” *International Journal of Operational Research*, vol. 7, no. 2, p. 152, 2010.

[93] H. Vermuyten, S. Lemmens, I. Marques, and J. Beliën, “Developing compact course timetables with optimized student flows,” *European Journal of Operational Research*, vol. 251, no. 2, pp. 651–661, 2016.

[94] G. Lach and M. E. Lübbecke, “Optimal university course timetables and the partial transversal polytope,” in *Experimental Algorithms, 7th International Workshop*, 2008, pp. 235–248.

[95] C. Beyrouthy, E. K. Burke, D. Landa-Silva, B. McCollum, P. McMullan, and A. J. Parkes, “Towards improving the utilization of university teaching space,” *Journal of the Operational Research Society*, vol. 60, no. 1, pp. 130–143, 2009. [Online]. Available: https://doi.org/10.1057/palgrave.jors.2602523

[96] C. Beyrouthy, E. K. Burke, D. L. Silva, B. McCollum, P. McMullan, and A. J. Parkes, “The teaching space allocation problem with splitting,” in *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, 2006, pp. 228–247.

[97] M. Lindahl, A. J. Mason, T. R. Stidsen, and M. Sørensen, “A strategic view of university timetabling,” *European Journal of Operational Research*, vol. 266, no. 1, pp. 35–45, 2018.

[98] 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.

[99] D. Holm, R. Mikkelsen, M. Sørensen, and T. Stidsen, “A mip formulation of the international timetabling competition 2019 problem,” Technical University of Denmark, Tech. Rep., 2020.

[100] H. Rudová, T. Müller, and K. S. Murray, “Complex university course timetabling,” *J. Sched.*, vol. 14, no. 2, pp. 187–207, 2011.

[101] G. Post, L. D. Gaspero, J. H. Kingston, B. McCollum, and A. Schaerf, “The third international timetabling competition,” *Ann. Oper. Res.*, vol. 239, no. 1, pp. 69–75, 2016.

[102] E. Demirović and N. Musliu, “Maxsat-based large neighborhood search for high school timetabling,” *Comput. Oper. Res.*, vol. 78, pp. 172–180, 2017.

[103] A. Caprara, M. Fischetti, and P. Toth, “Modeling and solving the train timetabling problem,” *Operations Research*, vol. 50, no. 5, pp. 851–861, 2002.

[104] G. Caimi, L. G. Kroon, and C. Liebchen, “Models for railway timetable optimization: Applicability and applications in practice,” *Journal of Rail Transport Planning & Management*, vol. 6, no. 4, pp. 285–312, 2017.

[105] C. Liebchen and R. H. Möhring, “The modeling power of the periodic event scheduling problem: Railway timetables — and beyond,” in *Algorithmic Methods for Railway Optimization*. Springer, 2007, pp. 3–40.

[106] P. Sels, T. Dewilde, D. Cattrysse, and P. Vansteenwegen, “Reducing the passenger travel time in practice by the automated construction of a robust railway timetable,” *Transportation Research Part B: Methodological*, vol. 84, pp. 124 – 156, 2016.

[107] M. Fischetti and M. Monaci, “Using a general-purpose mixed-integer linear programming solver for the practical solution of real-time train rescheduling,” *European Journal of Operational Research*, vol. 263, no. 1, pp. 258 – 264, 2017.

[108] C. Artigues, E. Bourreau, V. Jost, S. Kedad-Sidhoum, and F. Ramond, “Trains do not vanish: the ROADEF/EURO challenge 2014,” *Annals OR*, vol. 271, no. 2, pp. 1091–1105, 2018.

[109] J. Jordi, A. Toletti, G. Caimi, and K. Schupbach, “Applied timetabling for railways: Experiences with several solution approaches,” in *Proceedings of 8th International Conference on Railway Operations Modelling and Analysis (ICROMA)*, ser. RailNorrköping. Linköping Electronic Conference, 2019, pp. 1–9.

[110] M. Buljubašić, M. Vasquez, and H. Gavranović, “Two-phase heuristic for SNCF rolling stock problem,” *Annals of Operations Research*, vol. 271, no. 2, pp. 1107–1129, Jun. 2017.

[111] M. Goerigk, M. Schachtebeck, and A. Schöbel, “Evaluating line concepts using travel times and robustness,” *Public Transport*, vol. 5, no. 3, pp. 267–284, 2013.

[112] M. Bofill, D. Busquets, and M. Villaret, “Reformulation based MaxSAT robustness - (extended abstract),” in *Principles and Practice of Constraint Programming - 20th International Conference*, 2014, pp. 908–912.

[113] L. Climent, R. J. Wallace, M. A. Salido, and F. Barber, “Robustness and stability in constraint programming under dynamism and uncertainty,” *Journal of Artificial Intelligence Research*, vol. 49, pp. 49–78, 2014. [Online]. Available: https://doi.org/10.1613/jair.4126

[114] T. Petit and A. C. Trapp, “Finding diverse solutions of high quality to constraint optimization problems,” in *Proceedings of the 24th International Joint Conference on Artificial Intelligence IJCAI*, 2015, pp. 260–267.

[115] E. Hebrard, B. Hnich, B. O'Sullivan, and T. Walsh, "Finding diverse and similar solutions in constraint programming," in *20th National Conference on Artificial Intelligence and 17th Innovative Applications of Artificial Intelligence*, 2005, pp. 372–377.

[116] O. Bailleux and P. Marquis, "DISTANCE-SAT: complexity and algorithms," in *Proceedings of the 16th National Conference on Artificial Intelligence and 11th Conference on Innovative Applications of Artificial Intelligence*, 1999, pp. 642–647.

[117] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[118] P. Ross and D. Corne, "Comparing genetic algorithms, simulated annealing, and stochastic hill climbing on timetabling problems," in *Evolutionary Computing, AISB Workshop*, 1995, pp. 94–102.

[119] E. F. Krause, *Taxicab geometry: An adventure in non-Euclidean geometry*. Courier Corporation, 1975.

[120] D. A. Cohen, P. Jeavons, C. Jefferson, K. E. Petrie, and B. M. Smith, "Symmetry definitions for constraint satisfaction problems," *Constraints*, vol. 11, no. 2-3, pp. 115–137, 2006. [Online]. Available: https://doi.org/10.1007/s10601-006-8059-8

[121] T. Walsh, "Breaking value symmetry," *CoRR*, vol. abs/0903.1146, 2009.

[122] W. Chen and L. Shi, "A variant of examination timetabling problem," in *IEEE International Conference on Automation Science and Engineering*, 2008, pp. 353–358.

[123] T. A. Nguyen, M. B. Do, A. Gerevini, I. Serina, B. Srivastava, and S. Kambhampati, "Generating diverse plans to handle unknown and partially known user preferences," *Artificial Intelligence*, vol. 190, pp. 1–31, 2012.

[124] E. Hebrard, B. O'Sullivan, and T. Walsh, "Distance constraints in constraint satisfaction," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence IJCAI*, 2007, pp. 106–111.

[125] O. Bailleux and P. Marquis, "DISTANCE-SAT: complexity and algorithms," in *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, J. Hendler and D. Subramanian, Eds. The MIT Press, 1999, pp. 642–647.

[126] O. Bailleux and P. Marquis, "Some computational aspects of distance-sat," *J. Autom. Reasoning*, vol. 37, no. 4, pp. 231–260, 2006. [Online]. Available: https://doi.org/10.1007/s10817-006-9063-9

[127] T. Eiter, E. Erdem, H. Erdogan, and M. Fink, "Finding similar/diverse solutions in answer set programming," *Computing Research Repository*, vol. abs/1108.3260, 2011. [Online]. Available: http://arxiv.org/abs/1108.3260

[128] C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh, “Filtering algorithms for the n value constraint,” *Constraints*, vol. 11, no. 4, pp. 271–293, 2006. [Online]. Available: https://doi.org/10.1007/s10601-006-9001-9

[129] I. Abío, M. Deters, R. Nieuwenhuis, and P. J. Stuckey, “Reducing chaos in sat-like search: Finding solutions close to a given one,” in *Theory and Applications of Satisfiability Testing - SAT*, 2011, pp. 273–286.

[130] J. L. Gross, J. Yellen, and P. Zhang, *Handbook of graph theory*, ser. Discrete mathematics and its applications. CRC Press, 2004.

[131] R. Zivan, A. Grubshtein, and A. Meisels, “Hybrid search for minimal perturbation in dynamic CSPs,” *Constraints*, vol. 16, no. 3, pp. 228–249, 2011.

[132] J. Kingston, “Specifying and solving minimal perturbation problems in timetabling,” in *Proceeding of 11th International Conference of the Practice and Theory of Automated Timetabling(PATAT)*, 2016, pp. 207–210.

[133] S. G. Vadlamudi and S. Kambhampati, “A combinatorial search perspective on diverse solution generation,” in *Proceedings of the 13th Conference on Artificial Intelligence*, 2016, pp. 776–783.

[134] R. Samaga, A. von Kamp, and S. Klamt, “Computing combinatorial intervention strategies and failure modes in signaling networks,” *Journal of Computational Biology*, vol. 17, no. 1, pp. 39–53, 2010.

[135] N. Roos, Y. Ran, and H. J. van den Herik, “Combining local search and constraint propagation to find a minimal change solution for a dynamic CSP,” in *AIMSA Artificial Intelligence: Methodology, Systems, and Applications, 9th International Conference*, 2000, pp. 272–282.

[136] A. Perez-Lopez, R. Baltazar, J. M. Carpio, H. Terashima-Marín, D. J. Magaña-Lozano, and H. J. Puga, “Homogeneous population solving the minimal perturbation problem in dynamic scheduling of surgeries,” in *Advances in Artificial Intelligence and Its Applications - MICAI 12th Mexican International Conference on Artificial Intelligence*, 2013, pp. 473–484.

[137] M. Goerigk and A. Schöbel, “An empirical analysis of robustness concepts for timetabling,” in *ATMOS 2010 - 10th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, 2010, pp. 100–113.

[138] M. Goerigk, M. Schmidt, A. Schöbel, M. Knoth, and M. Müller-Hannemann, “The price of strict and light robustness in timetable information,” *Transportation Science*, vol. 48, no. 2, pp. 225–242, 2014.

[139] D. Bertsimas and M. Sim, “The price of robustness,” *Operations Research*, vol. 52, no. 1, pp. 35–53, 2004.

[140] D. Bertsimas, I. Dunning, and M. Lubin, “Reformulation versus cutting-planes for robust optimization,” *Computational Management Science*, vol. 13, no. 2, pp. 195–217, 2016.

[141] G. N. Frederickson and R. Solis-Oba, “Efficient algorithms for robustness in resource allocation and scheduling problems,” *Theoretical Computer Science*, vol. 352, no. 1-3, pp. 250–265, 2006.

[142] E. Demirović, N. Schwind, T. Okimoto, and K. Inoue, “Recoverable team formation: Building teams resilient to change,” in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems(AAMAS)*, E. André, S. Koenig, M. Dastani, and G. Sukthankar, Eds. ACM, 2018, pp. 1362–1370.

[143] N. Schwind, E. Demirovic, K. Inoue, and J. Lagniez, “Partial robustness in team formation: Bridging the gap between robustness and resilience,” in *Proceedings of the 21th International Conference on Autonomous Agents and MultiAgent Systems(AAMAS)*, F. Dignum, A. Lomuscio, U. Endriss, and A. Nowé, Eds. ACM, 2021, pp. 1154–1162.

[144] E. Hebrard, B. Hnich, and T. Walsh, “Super solutions in constraint programming,” in *Proceedings of 1th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, ser. Lecture Notes in Computer Science, J. Régin and M. Rueher, Eds., vol. 3011. Springer, 2004, pp. 157–172. [Online]. Available: https://doi.org/10.1007/978-3-540-24664-0_11

[145] M. L. Ginsberg, A. J. Parkes, and A. Roy, “Supermodels and robustness,” in *Proceedings of the 15th National Conference on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conference*, 1998, pp. 334–339.

[146] W. Kocjan, “Symmetric cardinality constraints,” Master’s Thesis, Mälardalen University, 2005.

[147] H. E. Sakkout and M. Wallace, “Probe backtrack search for minimal perturbation in dynamic scheduling,” *Constraints*, vol. 5, no. 4, pp. 359–388, 2000.

[148] R. Barták, T. Muller, and H. Rudová, “Minimal perturbation problem-a formal view,” *Neural Network World*, vol. 13, no. 5, pp. 501–512, 2003.

[149] A. Caprara, P. Toth, D. Vigo, and M. Fischetti, “Modeling and solving the crew rostering problem,” *Oper. Res.*, vol. 46, no. 6, pp. 820–830, 1998.

[150] B. Adenso-Díaz, M. Oliva González, and P. González-Torre, “On-line timetable re-scheduling in regional train services,” *Transportation Research Part B: Methodological*, vol. 33, no. 6, pp. 387 – 398, 1999.

[151] R. Acuna-Agost, P. Michelon, D. Feillet, and S. Gueye, “SAPI: Statistical analysis of propagation of incidents. a new approach for rescheduling trains after disruptions,” *European Journal of Operational Research*, vol. 215, no. 1, pp. 227–243, 2011.

[152] M. Schachtebeck and A. Schöbel, “To wait or not to wait—and who goes first? delay management with priority decisions,” *Transportation Science*, vol. 44, no. 3, pp. 307–321, 2010.

[153] L. P. Veelenturf, M. P. Kidd, V. Cacchiani, L. G. Kroon, and P. Toth, “A railway timetable rescheduling approach for handling large-scale disruptions,” *Transportation Science*, vol. 50, no. 3, pp. 841–862, 2016.

[154] X. Li, B. Shou, and D. A. Ralescu, “Train rescheduling with stochastic recovery time: A new track-backup approach,” *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 44, no. 9, pp. 1216–1233, 2014.

[155] F. Corman, A. D’Ariano, D. Pacciarelli, and M. Pranzo, “Optimal inter-area coordination of train rescheduling decisions,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 71 – 88, 2012, select Papers from the 19th International Symposium on Transportation and Traffic Theory.

[156] S. Binder, Y. Maknoon, and M. Bierlaire, “The multi-objective railway timetable rescheduling problem,” *Transportation Research Part C: Emerging Technologies*, vol. 78, pp. 78–94, 2017.

[157] K. Sato, K. Tamura, and N. Tomii, “A MIP-based timetable rescheduling formulation and algorithm minimizing further inconvenience to passengers,” *Journal of Rail Transport Planning & Management*, vol. 3, no. 3, pp. 38–53, 2013.

[158] S. V. Aken, N. Bešinović, and R. M. Goverde, “Solving large-scale train timetable adjustment problems under infrastructure maintenance possessions,” *Journal of Rail Transport Planning & Management*, vol. 7, no. 3, pp. 141–156, 2017.

[159] A. Higgins, E. Kozan, and L. Ferreira, “Optimal scheduling of trains on a single line track,” *Transportation Research Part B: Methodological*, vol. 30, no. 2, pp. 147 – 161, 1996.

[160] K. Molloy, “Artificial intelligence in train scheduling problems,” Ph.D. dissertation, Manchester Metropolitan University, 2020.

[161] A. Z. Zeng, C. F. Durach, and Y. Fang, “Collaboration decisions on disruption recovery service in urban public tram systems,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 3, pp. 578–590, May 2012.

[162] Y.-Y. Tseng and E. T. Verhoef, “Value of time by time of day: A stated-preference study,” *Transportation Research Part B: Methodological*, vol. 42, no. 7-8, pp. 607–618, 2008.

[163] K. Hoffmann, U. Buscher, J. S. Neufeld, and F. Tamke, “Solving practical railway crew scheduling problems with attendance rates,” *Business & Information Systems Engineering*, vol. 59, no. 3, pp. 147–159, 2017.

[164] E. Demirović, N. Musliu, and F. Winter, “Modeling and solving staff scheduling with partial weighted maxSAT,” *Annals of Operations Research*, vol. 275, no. 1, pp. 79–99, 2017.

[165] A. Lemos, H. Galhardas, P. T. Monteiro, and I. Lynce, “Academic data profiling and cleaning: A university timetable case study,” *unpublished*, 2018.

[166] S. K. Mirrazavi, S. J. Mardle, and M. Tamiz, “A two-phase multiple objective approach to university timetabling utilising optimisation and evolutionary solution methodologies,” *J. Oper. Res. Soc.*, vol. 54, no. 11, pp. 1155–1166, 2003.

[167] F. Gurski, “Efficient binary linear programming formulations for boolean functions,” *Statistics, Optimization & Information Computing*, vol. 2, no. 4, 2014.

[168] J. Edmonds, “Submodular functions, matroids, and certain polyhedra,” *Combinatorial Optimization*, vol. 11, pp. 11–26, 1970.

[169] S. Bernardini, F. Fagnani, and C. Piacentini, “Through the lens of sequence submodularity,” in *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*. AAAI Press, 2020, pp. 38–47.

[170] “When greedy algorithms are good enough: Submodularity and the (1 - 1/e) approximation,” https://jeremykun.com/2014/07/07/when-greedy-algorithms-are-good-enough-submodularity-and-the-1-1e-approximation/, 2014, accessed: 2017-07-03.

[171] E. K. Burke, J. Mareček, A. J. Parkes, and H. Rudová, “Penalising patterns in timetables: Novel integer programming formulations,” in *Operations Research Proceedings*. Springer, 2008, pp. 409–414.

[172] W. Ku and J. C. Beck, “Mixed integer programming models for job shop scheduling: A computational analysis,” *Computers & Operations Research*, vol. 73, pp. 165–173, 2016.

[173] O. Roussel, “Controlling a solver execution with the runsolver tool,” *Journal on Satisfiability, Boolean Modelling and Computation*, vol. 7, no. 4, pp. 139–144, 2011.

[174] L. Gurobi Optimization, “Gurobi optimizer reference manual,” 2018. [Online]. Available: http://www.gurobi.com

[175] C. Beyrouthy, E. K. Burke, D. Landa-Silva, B. McCollum, P. McMullan, and A. J. Parkes, “Towards improving the utilization of university teaching space,” *Journal of the Operational Research Society*, vol. 60, no. 1, pp. 130–143, 2009.

[176] D. H. Fylstra, L. S. Lasdon, J. Watson, and A. D. Waren, “Design and use of the Microsoft Excel solver,” *Interfaces*, vol. 28, no. 5, pp. 29–55, 1998.

[177] M. Banbara, T. Soh, N. Tamura, K. Inoue, and T. Schaub, “Answer set programming as a modeling language for course timetabling,” *TPLP*, vol. 13, no. 4-5, pp. 783–798, 2013.

[178] M. Banbara, K. Inoue, B. Kaufmann, T. Schaub, T. Soh, N. Tamura, and P. Wanko, “$teaspoon$: Solving the curriculum-based course timetabling problems with Answer Set Programming,” in *Proceeding of 11th International Conference of the Practice and Theory of Automated Timetabling (PATAT)*, 2016, pp. 13–32.

[179] F. De Cesco, L. Di Gaspero, and A. Schaerf, “Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation, and results,” in *Proceedings of the 7th PATAT Conference*, 2008, pp. 86–92.

[180] A. Lemos, P. T. Monteiro, and I. Lynce, “ITC-2019: A MaxSAT approach to solve university timetabling problems,” in *Proceedings of the International Timetabling Competition 2019*, 2020. [Online]. Available: https://www.itc2019.org/papers/itc2019-lemos.pdf

[181] M. W. Carter, “A comprehensive course timetabling and student scheduling system at the University of Waterloo,” in *Proceedings of the 3th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, 2000, pp. 64–84.

[182] D. Schindl, “Optimal student sectioning on mandatory courses with various sections numbers,” *Annals of Operations Research*, vol. 275, no. 1, pp. 209–221, 2019.

[183] R. Hoshino and I. Fabris, “Optimizing student course preferences in school timetabling,” in *Proceedings of 17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*. Springer, 2020, pp. 283–299.

[184] A. Nadel, “Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization,” in *Proceedings of the 19th Conference on Formal Methods in Computer Aided Design (FMCAD)*, 2019.

[185] J. P. Warners, “A linear-time transformation of linear inequalities into conjunctive normal form,” *Information Processing Letters*, vol. 68, no. 2, pp. 63–69, 1998.

[186] C. Ansótegui and F. Manyà, “Mapping problems with finite-domain variables into problems with boolean variables,” in *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, vol. 3542, 2004, p. 1–15.

[187] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce, “Boolean lexicographic optimization: algorithms & applications,” *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 3-4, pp. 317–343, 2011.

[188] F. Heras, J. Larrosa, and A. Oliveras, “MiniMaxSAT: An efficient weighted Max-SAT solver,” *Journal of Artificial Intelligence Research*, vol. 31, pp. 1–32, 2008.

[189] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, ser. Mathematics and Its Applications. Springer, 1987, vol. 37. [Online]. Available: https://doi.org/10.1007/978-94-015-7744-1

[190] J. Jordi and S. Mohanty, *Train Schedule Optimisation Challenge*, 2018, https://www.crowdai.org/challenges/train-schedule-optimisation-challenge (accessed Apr 15, 2020).

[191] “Transport outlook,” p. 200, 2019.

[192] R. Berger, *World Rail Market Study forecast 2018 to 2023*. The European Rail Supply Industry Association (UNIFE), 2018.

[193] P. Serafini and W. Ukovich, "A mathematical model for periodic scheduling problems," *SIAM Journal on Discrete Mathematics*, vol. 2, no. 4, pp. 550–581, 1989.

[194] I. van Heuven van Staereling, "Tree decomposition methods for the periodic event scheduling problem," in *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, (ATMOS)*, ser. OASICS, vol. 65. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 6:1–6:13.

[195] T. Ministry of Land, Infrastructure and Tourism, *WHITE PAPER ON LAND, INFRASTRUCTURE, TRANSPORT AND TOURISM IN JAPAN*, 2019.

[196] A. Schutt, "Improving scheduling by learning," Ph.D. dissertation, The University of Melbourne, 2011.

[197] A. Schutt, T. Feydy, P. J. Stuckey, and M. G. Wallace, "Solving RCPSP/max by lazy clause generation," *Journal of Scheduling*, vol. 16, no. 3, pp. 273–289, 2013.

[198] D. Abels, J. Jordi, M. Ostrowski, T. Schaub, A. Toletti, and P. Wanko, *SBB Train Scheduling Validator*, 2020, https://github.com/potassco/train-scheduling-with-hybrid-asp/releases (accessed Apr 10, 2020).

[199] M. Goerigk, *Verification of the PESPlib library*, 1989, http://num.math.uni-goettingen.de/ m.goerigk/pesplib/programs/verification.cpp (accessed Aug, 2020).

[200] G. Anagnostopoulos and V. Moosavi, "Stationrank: Aggregate dynamics of the swiss railway," *CoRR*, vol. abs/2006.02781, 2020.

[201] A. D. Marra and F. Corman, "From delay to disruption: Impact of service degradation on public transport networks," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2674, no. 10, pp. 886–897, 2020.

[202] W. Burgholzer, G. Bauer, M. Posset, and W. Jammernegg, "Analysing the impact of disruptions in intermodal transport networks: A micro simulation-based model," *Decis. Support Syst.*, vol. 54, no. 4, pp. 1580–1586, 2013.

[203] B. Büchel, T. Partl, and F. Corman, "The disruption at rastatt and its effects on the swiss railway system," in *Proceedings of 8th International Conference on Railway Operations Modelling and Analysis (ICROMA)*, ser. RailNorrköping. Linköping Electronic Conference, 2019, pp. 201–218.

[204] F. Bacchusand, M. Järvisalo, and R. Martins, Eds., *MaxSAT Evaluation 2019 Solver and Benchmark Descriptions*. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, Finland, 2020.

[205] A. Nadel, “On optimizing a generic function in sat,” in *Formal Methods in Computer Aided Design (FMCAD)*, 2020.

[206] S. Joshi, P. Kumar, S. Rao, and R. Martins, “Open-wbo-inc in maxsat evaluation 2020,” pp. 26–27, 2020.

[207] S. C. Zhendong Lei, “Satlike-c(w): Solver description,” p. 15, 2020.

[208] Z. Lei and S. Cai, “Satlike3.0-c: solver description,” *MaxSAT Evaluation*, p. 26, 2019.

[209] K. Eggensperger, M. Lindauer, and F. Hutter, “Pitfalls and best practices in algorithm configuration,” *J. Artif. Intell. Res.*, vol. 64, pp. 861–893, 2019.

[210] G. Audemard and L. Simon, “Lazy clause exchange policy for parallel SAT solvers,” in *Proceedings of 17th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, ser. Lecture Notes in Computer Science, C. Sinz and U. Egly, Eds., vol. 8561. Springer, 2014, pp. 197–205.

[211] A. Ignatiev, A. Morgado, and J. Marques-Silva, “RC2: an efficient maxsat solver,” *J. Satisf. Boolean Model. Comput.*, vol. 11, no. 1, pp. 53–64, 2019.

[212] J. Berg, E. Demirović, and P. Stuckey, “Loandra in the 2020 maxsat evaluation,” pp. 10–11, 2020.

[213] J. Berg, E. Demirović, and P. Stuckey, “Core-boosted linear search for incomplete MaxSAT,” in *Proceedings of 16th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, ser. Lecture Notes in Computer Science, vol. 11494. Springer, 2019, pp. 39–56.

[214] E. Demirović and P. J. Stuckey, “Techniques inspired by local search for incomplete MaxSAT and the linear algorithm: Varying resolution and solution-guided search,” in *Proceedings of 25th International Conference on Principles and Practice of Constraint Programming (CP)*, ser. Lecture Notes in Computer Science, vol. 11802. Springer, 2019, pp. 177–194.

[215] M. Piotrow, “Uwrmaxsat: an efficient solver in maxsat evaluation 2020,” pp. 34–35, 2020.

[216] S. Joshi, P. Kumar, S. Rao, and R. Martins, “Open-WBO-Inc: Approximation strategies for incomplete weighted MaxSAT,” *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 11, no. 1, pp. 73–97, 2019.

[217] M. Riser, *Train Schedule Solver*, 2018, https://github.com/iquadrat/sbb-train-scheduler (accessed Jan 30, 2020).

[218] F. Baldi, H. Butun, I. Kantor, L. Middelhauve, and R. Suciu, *MILP: Train Schedule Solver*, 2018, https://github.com/iquadrat/sbb-train-scheduler (accessed Jan 30, 2020).

[219] A. Nadel, “Polarity and variable selection heuristics for sat-based anytime maxsat,” *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 12, no. 1, pp. 17–22, 2020.

[220] K. Ham, “Openrefine (version 2.5). http://openrefine. org. free, open-source tool for cleaning and transforming data,” *Journal of the Medical Library Association: JMLA*, vol. 101, no. 3, p. 233, 2013.

[221] H. Galhardas, D. Florescu, D. E. Shasha, E. Simon, and C. Saita, “Declarative data cleaning: Language, model, and algorithms,” in *International Conference on Very Large Data Bases (VLDB), Roma, Italy*, 2001, pp. 371–380.

Data Cleaning

This appendix presents the data profiling, transformation, and cleaning process that needs to be applied to the data extracted from the FénixEdu™ system. The steps performed in the context of this work are organized according to the workflow represented in Figure A.1. The first step, (a) Converting JSON to relational, is the step where the data extracted from the FénixEdu™ system is converted from JSON to relational format. This step is essential since one of the data profiling tools and the data cleaning prototype do not support the JSON format as input. Step (b) Data Profiling focuses on assessing the data output quality by data conversion in step (a). This step produces a report containing all the data quality problems found. Step (c) Data Cleaning is the step where the data converted in step (a) is cleaned based on the findings in step (b). The last step (d) Data Transformation transforms the output of step (c) in order to obtain data in the format required for automating the production of timetables (ITC 2019).

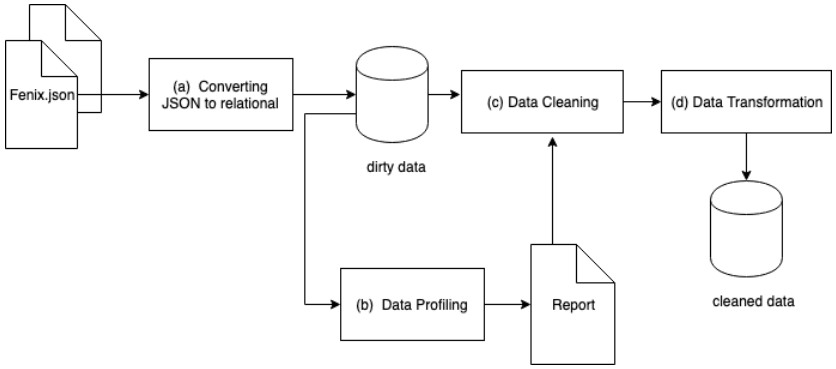


Figure A.1: Data profiling, transformation and cleaning process.

To address the first step (a) converting JSON to relational format, we first used Open-Refine [220]. However, the program built could not finish loading in a reasonable amount of time when opening multiple JSON files. Therefore, we decided to use the Pentaho Data Integration (PDI)[1] tools to perform the data conversion. Converting data from JSON to a relational schema is purely a data transformation without adding any type of constraints or pre-defined structure. The idea was to represent, as faithfully as possible, in a relational format, the JSON data obtained from FénixEdu™. Using constraints in the initial relational schema would impose a certain level of quality on the data imported from FénixEdu™. For this reason, we only considered the constraints for the relational model used to store the final cleaned data.

The data profiling step (b) was handled by the Arrahtec's Open Source Data Quality and Profiling (OSQD)[2] and PDI (without plug-ins). OSQD provides a wide range of functionalities, and it is easy to use. However, OSQD does not provide all the functionality needed. For example, the task of verifying if the room capacity is enough for the scheduled class is impossible to perform with OSQD. It only enables us to check for inclusion dependencies between fields. To fill this gap, we used PDI. PDI provides more expressiveness than OSQD since its base functionalities can be combined in order to obtain more complex functionalities, for example, checking if the values resulting from the concatenation of columns A and B are equal to the values of column C.

[1] PDI: Pentaho Data Integration documentation is available at http://community.pentaho.com/.
[2] Arrahtec's Open Source Data Quality and Profiling (OSQD) is available at http://www.arrahtec.com/.

The data profiling task identified the following data quality problems: exact duplicates; approximate duplicates, redundant data, heterogeneous date representations, domain violation, and integrity constraint violation. A detailed list of data quality problems is available at [165].

The last two steps, (c) Data Cleaning and (d) Data Transformation, are performed using the data cleaning prototype named Cleenex [221]. The prototype successfully solved all the data quality problems found. The programs are available at https://github.com/ADDALemos/CleanDataPrograms.