ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

UNIVERSITY OF IOANNINA
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PhD Dissertation

Improving Solver Performance in Scheduling Problems through Symmetry-Aware Modeling

Βελτίωση απόδοσης επιλυτών σε προβλήματα χρονοπρογραμματισμού μέσω μοντελοποίησης εξάλειψης συμμετριών

Angelos Dimitsas
Άγγελος Δήμητσας

Arta 2025

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διδακτορική Διατριβή με θέμα στα ελληνικά:

«Βελτίωση απόδοσης επιλυτών σε προβλήματα χρονοπρογραμματισμού μέσω μοντελοποίησης εξάλειψης συμμετριών»

και στα αγγλικά:

‘Improving Solver Performance in Scheduling Problems through Symmetry-Aware Modeling’

του κ. Άγγελου Δήμητσα, παρουσιάστηκε δημόσια στο Τμήμα Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων (υβριδικά)

στις 21/7/2025

και εξετάστηκε και εγκρίθηκε από την ακόλουθη επταμελή Εξεταστική Επιτροπή:

1. Γκόγκο Χρήστο, Καθηγητή Α’ Βαθμίδας του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων
2. Αντωνιάδη Νικόλαο, Καθηγητή Α’ Βαθμίδας του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων
3. Τζάλλα Αλέξανδρο, Αναπληρωτή Καθηγητή του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων
4. Αλεφραγκή Παναγιώτη, Αναπληρωτή Καθηγητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ του Πανεπιστημίου Πελοποννήσου
5. Ρεπούση Παναγιώτη, Αναπληρωτή Καθηγητή του Τμήματος Μάρκετινγκ και Επικοινωνίας του Οικονομικού Πανεπιστημίου Αθηνών
6. Σαμαρά Νικόλαο, Καθηγητή Α΄ βαθμίδας του Τμήματος Εφαρμοσμένης Πληροφορικής του Πανεπιστημίου Μακεδονίας
7. Φουτσιτζή Γεωργία, Καθηγήτρια Α΄ βαθμίδας Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων

Άρτα 22/7/2025

Ο Πρόεδρος του Τμήματος και Κοσμήτορας της Σχολής Πληροφορικής και Τηλεπικοινωνιών

Alexandros Tzallas
Alexandros Tzallas
22.07.2025 09:15

Αναπληρωτής Καθηγητής Αλέξανδρος Τζάλλας

Ο Επιβλέπων Καθηγητής

CHRISTOS GKOGKOS
22.07.2025 06:19 UTC

Καθηγητής Χρήστος Γκόγκος

Abstract

Symmetry is a pervasive and often problematic feature in combinatorial optimization problems, particularly in scheduling and timetabling. While symmetry is traditionally associated with elegance and balance in mathematics and the sciences, in optimization it frequently leads to redundant search spaces, inefficiencies, and computational bottlenecks. This thesis investigates the role of symmetry in a variety of combinatorial optimization contexts, with a focus on its identification, elimination, and exploitation to improve solution quality and computational performance.

The work begins with a theoretical foundation, exploring symmetry in linear, mixed-integer, constraint, and quadratic programming, as well as in metaheuristics, QUBO, and SAT formulations. It then presents five detailed case studies: the Uncapacitated Examination Timetabling Problem (UETP), Post Enrollment Course Timetabling (PE-CTT), Thesis Defense Timetabling Problem (TDTP), Sports Scheduling, and One-Machine Scheduling with Time-Dependent Capacity. Each case study demonstrates how symmetry manifests in real-world datasets and how its careful handling can lead to significant improvements in solution quality and solver efficiency.

The thesis introduces novel symmetry-breaking constraints, reformulations, and hybrid methodologies, including exact and heuristic approaches. Notably, it proposes a QUBO-based model for UETP suitable for quantum annealers, a hybrid framework for PE-CTT, and a symmetry-aware CP model for TDTP that achieves near-optimal results. In the sports scheduling domain, the work contributes to the ITC2021 competition with a constraint programming-based solver. Finally, a hybrid exact-local search method is developed for the one-machine scheduling problem, outperforming state-of-the-art memetic algorithms.

The findings underscore the critical importance of symmetry handling in combinatorial optimization and provide a comprehensive toolkit for researchers and practitioners seeking to enhance solver performance across diverse problem domains.

Περίληψη

Η συμμετρία αποτελεί ένα διαδεδομένο και συχνά προβληματικό χαρακτηριστικό στα προβλήματα συνδυαστικής βελτιστοποίησης, ιδιαίτερα στον προγραμματισμό και την κατάρτιση χρονοδιαγραμμάτων. Αν και η συμμετρία συνδέεται παραδοσιακά με την κομψότητα και την ισορροπία στα μαθηματικά και τις επιστήμες, στη βελτιστοποίηση οδηγεί συχνά σε πλεονάζοντες χώρους αναζήτησης, αναποτελεσματικότητα και υπολογιστικά εμπόδια. Η παρούσα διατριβή διερευνά τον ρόλο της συμμετρίας σε διάφορα πλαίσια συνδυαστικής βελτιστοποίησης, με έμφαση στην αναγνώριση, την εξάλειψη και την αξιοποίησή της για τη βελτίωση της ποιότητας των λύσεων και της υπολογιστικής απόδοσης.

Η εργασία ξεκινά με μια θεωρητική θεμελίωση, εξετάζοντας τη συμμετρία στον γραμμικό, μικτό ακέραιο, προγραμματισμό ύπο περιορισμούς και τετραγωνικό προγραμματισμό, καθώς και σε μεταευρετικούς αλγορίθμους, QUBO και SAT διατυπώσεις. Στη συνέχεια παρουσιάζονται πέντε λεπτομερείς μελέτες περίπτωσης: το πρόβλημα κατάρτισης εξεταστικού προγράμματος χωρίς χωρητικότητα, το πρόβλημα κατάρτισης προγράμματος μαθημάτων μετά την εγγραφή , το πρόβλημα κατάρτισης προγράμματος υποστηρίξεων διπλωματικών εργασιών, ο προγραμματισμός αθλητικών διοργανώσεων και ο προγραμματισμός μηχανής με χρονικά εξαρτώμενη χωρητικότητα. Κάθε μελέτη περίπτωσης δείχνει πώς εκδηλώνεται η συμμετρία σε πραγματικά δεδομένα και πώς η προσεκτική διαχείρισή της μπορεί να οδηγήσει σε σημαντικές βελτιώσεις στην ποιότητα των λύσεων και την αποδοτικότητα των επιλυτών.

Η διατριβή εισάγει νέους περιορισμούς διάσπασης συμμετρίας, αναδιατυπώσεις και υβριδικές μεθοδολογίες, συμπεριλαμβανομένων ακριβών και ευρετικών προσεγγίσεων. Ιδιαίτερα, προτείνεται ένα μοντέλο βασισμένο σε QUBO για το UETP κατάλληλο για κβαντικούς υπολογιστές, ένα υβριδικό πλαίσιο για το PE-CTT και ένα μοντέλο CP με επίγνωση συμμετρίας για το TDTP που επιτυγχάνει σχεδόν βέλτιστα αποτελέσματα. Στον τομέα του αθλητικού προγραμματισμού, η εργασία συμβάλλει στον διαγωνισμό ITC2021 με έναν επιλυτή βασισμένο στον προγραμματισμό περιορισμών. Τέλος, αναπτύσσεται μια υβριδική μέθοδος ακριβούς-τοπικής αναζήτησης για το πρόβλημα προγραμματισμού μηχανής, η οποία ξεπερνά τους πιο σύγχρονους μιμητικούς αλγορίθμους.

Τα ευρήματα υπογραμμίζουν τη ζωτική σημασία της διαχείρισης της συμμετρίας στη συνδυαστική βελτιστοποίηση και παρέχουν ένα ολοκληρωμένο εργαλείο για ερευνητές και επαγγελματίες που επιδιώκουν να ενισχύσουν την απόδοση των επιλυτών σε ποικίλους τομείς προβλημάτων.

- Yesterday's Future is Today

Contents

List of Figures

List of Tables

Acronyms

CB-CTT Curriculum-based course timetabling

COP Constraint Optimization Problem

CP Constraint Programming

CSP Constraint Satisfaction Problem

ITC International timetabling competition

LP Linear Programming

MINLP Mixed Integer Non Linear Programming

MIP Mixed Integer Programming

MIQP Mixed Integer Quadratic Programming

NLP Nonlinear Programming

NP Non Polynomial

PE-CTT Post-enrollment-based course timetabling

SAT Boolean Satisfiability

SBDD Symmetry Breaking by Dominance Detection

SBDS Symmetry Breaking During Search

QP Quadratic Programming

QUBO Quadratic Unconstrained Binary Optimization

TDTP Thesis defense timetabling problem

UETP Uncapacitated Examination Timetabling Problem

Chapter 1

Introduction

1.1 Context

Symmetry is a fundamental concept in both the natural and formal sciences, broadly defined as the property by which an object, system, or equation remains invariant under a set of transformations or operations. In academic discourse, symmetry is often associated with notions of balance, proportion, and harmony, but it is more rigorously characterized through mathematical formalism.

In mathematics and physics, symmetry refers to the invariance of a structure under specific transformations such as reflection, rotation, translation, or scaling. These transformations form a group under the framework of group theory, a branch of abstract algebra. A system is said to exhibit symmetry if it is invariant under the action of a symmetry group. For instance, a geometric figure like a square has rotational symmetry of order four, as it appears identical when rotated by 90°, 180°, 270°, or 360°.

In theoretical physics, symmetry principles underpin the formulation of physical laws. Symmetries are deeply linked to conservation laws through Noether's theorem, which states that every continuous symmetry of a physical system corresponds to a conserved quantity. For example, spatial translational symmetry (the invariance of a system of equations without rotation) is associated with the conservation of linear momentum, while temporal symmetry (the concept that the laws of physics are the same regardless of whether time is running forward or backward) corresponds to the conservation of energy.

In biology, chemistry, and aesthetics, symmetry is observed in morphological structures, molecular configurations, and design patterns, often associated with notions of stability, efficiency, and visual appeal.

Thus, symmetry serves as a unifying and organizing principle across disciplines, providing insight into both the structural and dynamical aspects of complex systems.

Symmetry, linked to balance, harmony, and beauty, has been extolled in several statements throughout the ages:

- "If measure and symmetry are absent from any composition in any degree, ruin awaits both the ingredients and the composition... Measure and symmetry are beauty and virtue the world over." - *Socrates*
- "The mathematical sciences particularly exhibit order, symmetry, and limitation; and these are the greatest forms of the beautiful." - *Aristotle*
- "We find, therefore, under this orderly arrangement, a wonderful symmetry in

the universe, and a definite relation of harmony in the motion and magnitude of the orbs, of a kind that is not possible to obtain in any other way." - *Johannes Keple*

- "Symmetry is what we see at a glance; based on the fact that there is no reason for any difference, and based also on the face of man; whence it happens that symmetry is only wanted in breadth, not in height or depth." - *Blaise Pascal*
- "Nature seems to take advantage of the simple mathematical representations of the symmetry laws. When one pauses to consider the elegance and the beautiful perfection of the mathematical reasoning involved and contrast it with the complex and far-reaching physical consequences, a deep sense of respect for the power of the symmetry laws never fails to develop." - *Chen-Ning Yang*
- "To a physicist, beauty means symmetry and simplicity. If a theory is beautiful, this means it has a powerful symmetry that can explain a large body of data in the most compact, economical manner. More precisely, and equation is considered to be beautiful if it remains the same when we interchange its components among themselves." - *Michio Kaku*

It is impossible to disagree with any of these significant scholars. This thesis though adopts a grim approach to symmetry. For the field of combinatorial optimization in specific:

- Symmetry is tedious.
- Symmetry is redundant.
- Symmetry is wasting resources.
- Symmetry is a bottleneck.
- Symmetry must be exterminated, if not possible, at least exploited.

Scheduling is subject to the No Free Lunch theorem by Wolpert et al. (1997) [4], for a review of the theorem you are referred to Adam et al. (2019) [5].

1.2 Examples of symmetries in combinatorial optimization problems.

To provide intuition for symmetries in combinatorial optimization problems two fundamental combinatorial optimization problems will be briefly examined for symmetries. Mathematical formulas are intentionally excluded from this introductory section.

1.2.1 0/1 Knapsack

The 0/1 Knapsack Problem is a fundamental problem in combinatorial optimization. Given a set of n items, where each item i has a positive integer value v_i and weight w_i, and a knapsack of maximum capacity W, the goal is to determine the subset of items to include in the knapsack such that the total value is maximized without exceeding the knapsack's capacity. Each item can be either included or excluded — hence "0/1".

The 0/1 Knapsack Problem is NP-complete, see [6]. However, it admits a pseudo-polynomial time solution using dynamic programming with time complexity $O(nW)$, making it weakly NP-complete.

Suppose we have 4 items and a knapsack of capacity 8. Table 1.1 lists items' properties:

Table 1.1: Properties of items.

Item (i)	Value (v_i)	Weight (w_i)
1	16	2
2	14	3
3	14	3
4	14	3

The goal is to choose a subset of items with total weight ≤ 8 that maximizes the total value. In Table 1.2 both feasible and infeasible solutions are enumerated:

Table 1.2: All solutions to the 0/1 knapsack instance. Infeasible solutions in red. Optimal solutions in green.

Solution	Item 1	Item 2	Item 3	Item 4	Total weight	Total value
1	0	0	0	0	0	0
2	0	0	0	1	3	14
3	0	0	1	0	3	14
4	0	0	1	1	6	28
5	0	1	0	0	3	14
6	0	1	0	1	6	28
7	0	1	1	0	6	28
8	0	1	1	1	9	42
9	1	0	0	0	2	16
10	1	0	0	1	5	30
11	1	0	1	0	5	30
12	1	0	1	1	8	44
13	1	1	0	0	5	30
14	1	1	0	1	8	44
15	1	1	1	0	8	44
16	1	1	1	1	11	58

So, this instance has 16 solutions in total, 2 are infeasible and 3 are optimal. If one is to take a good look on Table 1.1 he will notice that items 2, 3 and 4 are identical. In our optimal solutions two of these items as long as item 1 is selected. We can exploit this symmetry in two ways.

Lexicographical

We enforce that someone can not choose item 3 if he hasn't chosen item 2. We also don't allow choosing item 4 if item 3 is not also chosen. The solutions with this approach are listed in Table 1.3.

Table 1.3: All solutions to the 0/1 knapsack instance, with the lexicographical rules. Infeasible solutions in red. Optimal solutions in green.

Solution	Item 1	Item 2	Item 3	Item 4	Total weight	Total value
1	0	0	0	0	0	0
2	0	1	0	0	3	14
3	0	1	1	0	6	28
4	0	1	1	1	9	42
5	1	0	0	0	2	16
6	1	1	0	0	5	30
7	1	1	1	0	8	44
8	1	1	1	1	11	58

By eliminating the symmetries, this instance has 8 solutions in total, 2 are infeasible

and there is a single optimal solution.

Integer variables

We will alter our decision variables from binary to integer, one can choose an integer number of similar items (up to the number of these items of course) and 0-1 for non unique items. The solutions with this approach are listed in Table 1.4.

Table 1.4: All solutions to the 0/1 knapsack instance, with the Integer variables rules. Infeasible solutions in red. Optimal solutions in green.

Solution	Item 1	Counter for items (2,3,4)	Total weight	Total value
1	0	0	0	0
2	0	1	3	14
3	0	2	6	28
4	0	3	9	42
5	1	0	2	16
6	1	1	5	30
7	1	2	8	44
8	1	3	11	58

The result is the same as before, this instance has 8 solutions in total, 2 are infeasible and there is a single optimal solution. This is to be excepted, we are eliminating the same symmetries after all.

1.2.2 Symmetric Traveling Salesman Problem

The Traveling Salesman Problem is a classic problem in combinatorial optimization and theoretical computer science. Imagine a salesman who needs to visit a bunch of cities, but there's a catch: they want to visit each city exactly once and return to where they started. The objective is to take the shortest possible route. In the symmetric version of the problem, the distance from city A to city B is the same as from city B to city A.

Here's a small example with four cities: A, B, C, and D. The distances, let's say in minutes, between them are shown in the graph in Figure 1.1 and Table 1.5.

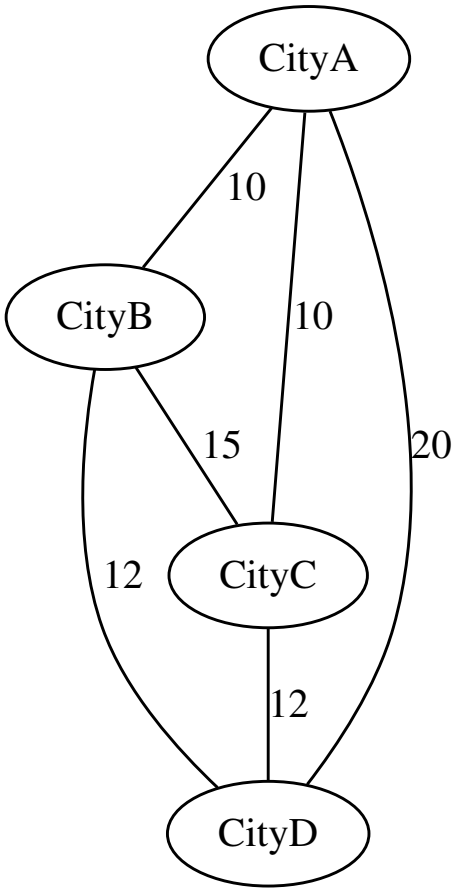

Figure 1.1: Graph for the symmetrical Traveling Salesman Problem instance with four cities.

Table 1.5: Distances in minutes between cities.

	City A	City B	City C	City D
City A	0	10	10	20
City B	10	0	15	12
City C	10	15	0	12
City D	20	12	12	0

In Table 1.6 all solutions are enumerated:

Table 1.6: All solutions to the symmetrical traveling salesman instance. Optimal solutions in green.

Solution	Start City	Second City	Third City	Last City	Total time
1	City A	City B	City C	City D	57
2	City A	City B	City D	City C	44
3	City A	City C	City B	City D	57
4	City A	City C	City D	City B	44
5	City A	City D	City B	City C	57
6	City A	City D	City C	City B	57
7	City B	City A	City C	City D	44
8	City B	City A	City D	City C	57
9	City B	City C	City A	City D	57
10	City B	City C	City D	City A	57
11	City B	City D	City A	City C	57
12	City B	City D	City C	City A	44
13	City C	City A	City B	City D	44
14	City C	City A	City D	City B	57
15	City C	City B	City A	City D	57
16	City C	City B	City D	City A	57
17	City C	City D	City A	City B	57
18	City C	City D	City B	City A	44
19	City D	City A	City B	City C	57
20	City D	City A	City C	City B	57
21	City D	City B	City A	City C	44
22	City D	City B	City C	City A	57
23	City D	City C	City A	City B	44
24	City D	City C	City B	City A	57

There are 24 solutions in total. 8 of them are optimal. As the name implies the **symmetrical** Traveling Salesman Problem is inherently symmetrical. Regardless of the city that we start we will make a full circle. For this reason we choose to always start from City A, we could have chosen any city in fact. The solutions are now listed in Table 1.7.

Table 1.7: All solutions to the symmetrical traveling salesman instance, starting from City A. Optimal solutions in green.

Solution	Start City	Second City	Third City	Last City	Total time
1	City A	City B	City C	City D	57
2	City A	City B	City D	City C	44
3	City A	City C	City B	City D	57
4	City A	City C	City D	City B	44
5	City A	City D	City B	City C	57
6	City A	City D	City C	City B	57

We are now left with 6 solutions two of them optimal. But there is more, if we examine the graph in Figure 1.1 we can see that the distances for Cities B and C to other cities are the same (10 for City A and 12 for City D) the distance between them is 15 bidirectional since this is the symmetrical version of the problem. If we now enforce than one must travel first through City B and then C (lexicographical approach) we can further eliminate symmetrical solutions. The result is shown in Table 1.8.

Table 1.8: All solutions to the symmetrical traveling salesman instance, starting from City A and visiting City B before C. Optimal solutions in green.

Solution	Start City	Second City	Third City	Last City	Total time
1	City A	City B	City C	City D	57
2	City A	City B	City D	City C	44
3	City A	City D	City B	City C	57

Finally there are three non symmetrical solutions to the problem instance, one of them is optimal.

1.3 Motivations and objectives

The motivational factor for this thesis is the complexity of solving integer optimization problems. I engage with deterministic single (mostly, sometimes multi but they are not in the scopes of this thesis) objective combinatorial optimization problems both in Academia and professionally. Getting rid of symmetries in scheduling and timetabling problems can provide huge benefits in the solution process. For an overview on scheduling problems readers are referred to the work of Pinedo (2022) [7].

The objectives are:

1. Use public available instances of scheduling and timetabling problems. So there can be a comparison.
2. Identify symmetries and rewrite a model that either nullifies symmetries or at least minimizes their effect.
3. For problems that allow for different solution methods, eliminate symmetries for all methodologies.
4. Assess the impact of symmetry elimination on the solution methodology.

1.4 Structure of the thesis

The remaining chapters of the thesis are organized as follows. Chapter 2 provides an overview of the methods that are used in this thesis.

The next three Chapters are dedicated to case studies from the world of educational timetabling:

- Chapter 3 presents the Uncapacitated Examination Timetabling Problem. Mixed Integer Programming, Metaheuristics and Quadratic unconstrained binary optimization is employed here. The instances are taken from real world datasets. For the Quadratic Unconstrained Binary Optimization approach a new dataset is constructed so that it can be tested on a Quantum annealer.
- Chapter 4 revolves around the Post Enrollment Course Timetabling Problem. Mixed Integer Programming, Constraint Programming and Matheuristics (optimization algorithms that make use of mathematical programming (MP) techniques in order to obtain heuristic solutions) are used here. Again the instances are publicly available and used in benchmarking.
- Chapter 5 is dedicated to the Thesis Defense Timetabling Problem. Quadratic programming and Constraint Programming manage to produce near optimal or optimal for the most part solutions to two publicly available datasets.

Chapter 6 examines sports scheduling problems. Specifically Constraint Programming and metaheuristics are used to provide solutions for round robin tournament instances. The participation in the International Timetabling Competition of 2021 is covered here.

Finally, the last case study is in Chapter 7. This time in the field of task scheduling, constraint Programming is tested upon a task scheduling problem: One-Machine Scheduling with Time-Dependent Capacity.

This thesis concludes in Chapter 8 with remarks and highlights of this work. A list of publications by the author follows in 8.3.

Chapter 2

Methods

In this chapter an introduction of different methods with the equivalent problem spaces and exact solvers is presented.

2.1 Linear Programming

2.1.1 Definition

Linear Programming (LP) is a mathematical optimization technique aiming at the maximization or minimization of a linear objective function while taking into account a set of linear equality and/or inequality constraints. All of the relationships between the variables in LP problems are linear, and the problems are formulated over continuous decision variables. The standard form of a linear program is:

$$\begin{aligned} \text{maximize (or minimize)} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where $x \in \mathbb{R}^n$ is the vector of decision variables, $c \in \mathbb{R}^n$ is the objective coefficient vector, $A \in \mathbb{R}^{m \times n}$ is the matrix of constraint coefficients, and $b \in \mathbb{R}^m$ is the vector of constraint bounds.

Linear programming has been widely used in various domains such as operations research, economics, engineering, transportation, and manufacturing. Classical solution methods include the Simplex algorithm introduced by Dantzig (1963) [8], Khachiyan (1979) [9] and Interior Point Methods as developed in later years, see Kojima et al. (1989) [10].

For more comprehensive discussions and theoretical background about LP, see works by Chvátal (1983) [11] and Bertsimas et al. (1997) [12].

2.1.2 Symmetries in LP

A symmetry in a linear program is a permutation or linear transformation, a function between vector spaces that preserves vector addition and scalar multiplication, of the variables that maps feasible solutions to feasible solutions without affecting the objective function value.

A mapping $\pi : \mathbb{R}^n \to \mathbb{R}^n$ is a symmetry if:

- $A\pi(x) \leq b \Leftrightarrow A \leq b$,
- $c^T(x) = c^T\pi(x)$, for all feasible x.

As an example of a symmetrical LP consider:

$$\begin{aligned} \text{maximize} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 10 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Here, swapping x_1 and x_2 preserves feasibility and the objective value. But since the feasible region is convex and the objective linear, this symmetry causes no performance issues. The reason for this is that Simplex only examines only edge points in Figure 2.1 only points A, B and C will be examined, any solution in $x + y = 10$ is symmetrical and optimal.

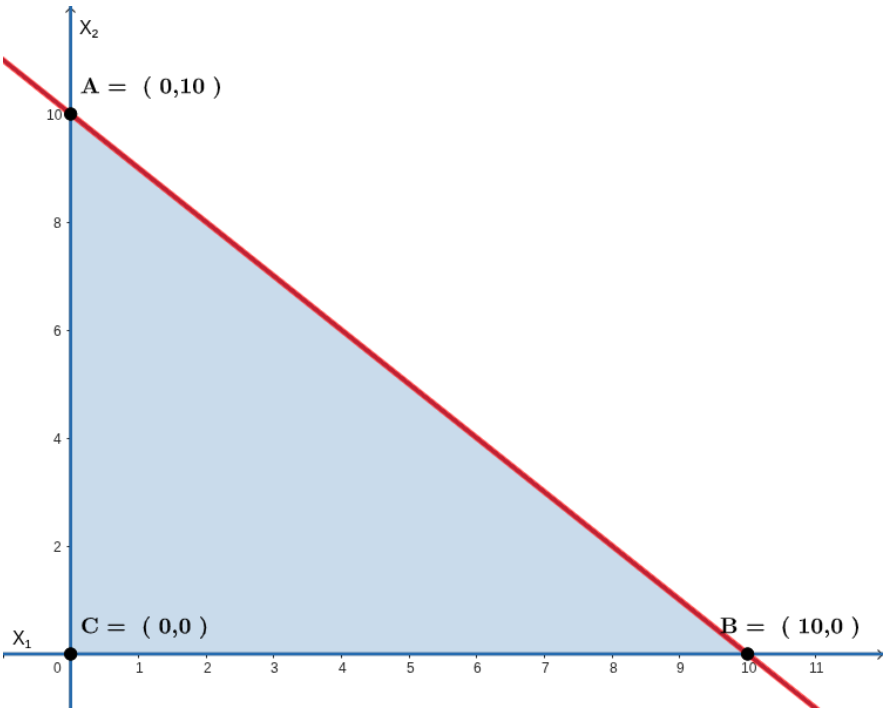

Figure 2.1: Graphical representation of the minimal problem.

Symmetries in LP are often harmless and rarely require explicit handling. The nature of Scheduling problems most often requires Integer variables, for this reason LP problems will not be featured in this thesis. There are multiple reasons why symmetries are benign in LP:

- Convexity: The feasible region of an LP is a convex polyhedron, so the presence of symmetry does not create multiple disconnected feasible regions or cause branching issues.

- Deterministic Solvers: Simplex and interior-point methods follow deterministic paths (e.g., toward extreme points or through the interior), ignoring symmetrical redundancies.
- Unique Optimal Solutions: Most LPs (after minor perturbations) have a unique optimal solution, minimizing issues due to symmetry.

It is worth noting that there are cases (though rare) when symmetry in fact Does matter in LP:

- Degenerate LPs: Where multiple basic feasible solutions have the same objective value; symmetry can cause cycling or performance issues in the simplex method.
- Model simplification: Detecting symmetry can reduce model size or complexity (e.g., aggregation in network flow models).
- Preprocessing or problem structure detection: In model generation, symmetry may indicate redundant structure that could be exploited.

2.2 Mixed-Integer Programming

2.2.1 Definition

Mixed Integer Programming (MIP) is a class of optimization problems where some decision variables are constrained to assume integer values, while others may be continuous. It generalizes Linear Programming (LP) by introducing integrality constraints, which significantly increase the problem's computational complexity. MIP problems are NP-hard due to the combinatorial complexity introduced by integer constraints. Unlike continuous linear programming, the feasible region in MIP is non-convex and often requires enumerative techniques for exact solutions. MIP models are extensively used in operations research, economics, engineering design, and artificial intelligence to model decisions involving discrete choices, such as:

- Resource allocation
- Scheduling and timetabling
- Network design
- Capital budgeting
- Logistics and supply chain optimization

The general form of a MIP problem is:

$$
\begin{aligned}
\text{minimize} \quad & c^T x \\
\text{subject to} \quad & Ax_k \leq b \quad k \in \mathcal{I} + \mathcal{J} \\
& x_i \in \mathbb{Z} \quad \text{for } i \in \mathcal{I} \\
& x_j \in \mathbb{R} \quad \text{for } j \in \mathcal{J}
\end{aligned}
$$

where $x \in \mathbb{R}^n$, $\mathcal{I} \subseteq \{1, \ldots, n\}$ is the index set of integer-constrained variables, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$.

MIP is used in a wide range of applications including scheduling, logistics, production planning, and combinatorial optimization. Solving MIP problems is typically NP-hard, and solution methods often rely on techniques such as branch-and-bound, branch-and-cut. Further information about these methods can be consulted in Nemhauser et al. (1988) [13] and Wolsey (1988) [14].

Significant advancements in solver technology have been made in recent decades, with modern solvers leveraging strong formulations, preprocessing, heuristics, and parallel computing. Achterberg (2009) [15],and Lodi (2009) [16] provide insights about the implementation in solvers.

For a detailed mathematical and computational treatment, readers are referred to Conforti et al. (2014) [17], and Bertsimas et al. (2005) [18].

2.2.2 Symmetries in MIP

Symmetries in MIP are fundamentally different from those in LP due to the convex and continuous nature of LP solution spaces. Symmetries in MIP are structural properties where different solutions (variable assignments) are equivalent in terms of the objective function and feasibility. While symmetries may seem benign, they pose significant challenges for MIP solvers by expanding the search space unnecessarily and causing redundant exploration. See Liberti (2012) [19], Margot (2010) [20], Pfetsch and Rehn (2019) [21].

Formally, a symmetry in a MIP model is a permutation of variables that maps feasible solutions to other feasible solutions without changing the objective value. That is, a mapping $\pi : \mathbb{R}^n \to \mathbb{R}^n$ is a symmetry if:

- x is feasible$\pi(x)$ is feasible,
- $f(x) = f(\pi(x))$ where f is the objective function.

Sources of symmetry can vary. Identical machines, facilities, or agents in assign-

ment and scheduling problems. Similar weighted edges in graph optimization problems. Binary decision variables that are interchangeable. In this thesis, focusing on scheduling problems, different kinds of symmetries will be presented. Ranging over educational timetabling, sports scheduling and job scheduling a plethora of these symmetries will appear and of course will be dealt with.

Symmetries cause:

- Redundant search in branch-and-bound trees.
- Slow convergence due to exploration of symmetric solutions.
- Increased memory usage.
- In the worst case, exponential growth in the number of symmetric branches.

The detection of symmetries is sometimes being done by the solvers. Graph-based approaches where a MIP is modeled as a colored graph and specialized tools find automorphisms. Constraint analysis where solvers automatically identify symmetric structures in constraints and variables. These techniques can provide significant performance boosts when solving MIP models, however they fall short where Entities which have a plethora of decision variables assigned for them can not be detected by them.

Handling these symmetries can be done automatically in solvers using Orbitopes (Kaible et al. (2008) [22]) and Orbital branching (Ostrowski et al. (2011) [23]). A survey in symmetries in Integer programming was presented by Margot (2010) [24]. Symmetry breaking constraints that eliminate symmetric solutions can also be used especially for symmetries that involve groups of variables.

2.3 Quadratic Programming

2.3.1 Definition

Quadratic Programming (QP) refers to the class of optimization problems where the objective function is quadratic and the constraints are linear. QP is a fundamental subclass of nonlinear programming and arises frequently in economics, finance, machine learning (e.g., support vector machines), and control systems.

A standard QP problem is formulated as:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} x^T Q x + c^T x \\ \text{subject to} \quad & Ax \leq b, \\ & Ex = d, \\ & x \in \mathbb{R}^n, \end{aligned}$$

where:

- $x \in \mathbb{R}^n$ is the vector of decision variables,
- $Q \in \mathbb{R}^{n \times n}$ is a symmetric matrix defining the quadratic part of the objective,
- $c \in \mathbb{R}^n$ is a vector defining the linear part,
- $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ represent inequality constraints,
- $E \in \mathbb{R}^{p \times n}$, $d \in \mathbb{R}^p$ represent equality constraints.

When Q is positive semidefinite, symmetric matrix where the quadratic form, formed by multiplying the matrix with a vector and its transpose, is always non-negative for any vector, the QP is convex and can be solved efficiently using interior-point methods, active-set methods, or gradient-based algorithms, Nocedal et al. (2006) [25]. If Q is not positive semi-definite, the problem is non-convex and generally harder to solve due to the possibility of local minima.

Quadratic programming plays a key role in machine learning—particularly in support vector machines (SVMs) Cortes et al. (1995) [26]—and in portfolio optimization problems in finance Markowitz (1952) [27].

2.3.2 Symmetries in QP

Symmetries in Quadratic Programming (QP) present a richer and more nuanced landscape than in Linear Programming (LP) due to the nonlinear objective function. These symmetries can impact both theory and computation, especially in non-convex QPs or when integer variables are involved (e.g., in MIQPs: Mixed Integer Quadratic Programs).

A symmetry in QP is a permutation or linear transformation of the variables that preserves the feasible set and the objective function value.

Formally, a permutation $\pi \in S_n$ is a symmetry if for all feasible x:

$$\frac{1}{2} x^T Q x + c^T x = \frac{1}{2} \pi x^T Q x + c^T \pi(x)$$

Where $Q = P^T Q P$ P is the permutation matrix, $c = P^T c$ and that all constraints are preserved.

Structural symmetry is especially common in portfolio optimization, energy systems, and combinatorial QP models. In convex QPs, symmetry is usually benign, similar to LP. In non-convex QPs, symmetry can cause:

- Multiple local minima,
- Redundant search space in global optimization,
- Difficulty in branch-and-bound or spatial branch-and-bound.

In MIQPs, symmetries cause the same computational issues as in MIPs, plus the added complexity of nonlinearity.

Symmetry-Breaking in QP may contain:

- Symmetry-breaking constraints: Imposed to reduce redundant solutions.
- Reformulations: Exploit symmetry to simplify the QP (e.g., by block-diagonalizing Q).
- Orbital Branching & Isomorphism Pruning: Extended to QP/MIQP from MIP theory.
- Group-theoretic methods: Used to analyze and exploit symmetry in non-convex QP models.

2.4 Constraint Programming

2.4.1 Definition

Constraint Programming (CP) is a paradigm for solving combinatorial problems that involves specifying a set of variables, their respective domains, and a collection of constraints that restrict the values the variables can simultaneously take. Unlike traditional optimization methods, CP focuses on feasibility—finding an assignment that satisfies all constraints—and supports a rich set of variable types and constraint expressions.

A Constraint Satisfaction Problem (CSP) is formally defined as the triplet (X, D, C), where:

- $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables,
- $D = \{D_1, D_2, \ldots, D_n\}$ is a set of finite domains, where $x_i \in D_i$,

- $C = \{c_1, c_2, \ldots, c_m\}$ is a set of constraints over subsets of X.

A solution is an assignment of values to all variables such that all constraints in C are satisfied. CP can also be extended to Constraint Optimization Problems (COPs) by associating an objective function to be minimized or maximized, adding an optimization layer to feasibility.

CP is especially powerful in dealing with scheduling, timetabling, configuration, and resource allocation problems due to its expressiveness and support for global constraints, Rossi et al. (2006) [28]. Constraint propagation, domain reduction, backtracking search, and consistency techniques (e.g., arc-consistency) are key algorithmic foundations see Mackworth (1977) [29], and van Hentenryck (1989) [30].

CP is often implemented through declarative modeling languages and solvers, see Nethercote et al. (2007) [31].

Although similar there are structural differences from MIP:

- CP focuses on constraint satisfaction; MIP focus on optimization (though CP can optimize too).
- CP handles logical constraints and combinatorial structures more naturally.
- MIP models rely on linear or nonlinear algebraic constraints; CP can handle arbitrary constraints.
- CP search is often guided by domain reduction and propagation rather than relaxations.

2.4.2 Symmetries in CP

While the causes for symmetries in CP remain the same, handling them is done differently. CP uses symmetry-breaking constraints, dynamic symmetry breaking, and symmetry-aware search heuristics. Tools like SBDS (Symmetry Breaking During Search) and SBDD (Symmetry Breaking by Dominance Detection) are common. For a review on symmetry breaking in the world of CP, see Gent et al. (2000) [32]. The focus of CP is mainly feasibility so this is a topic often neglected. In this thesis a few symmetry breaking constraints will be applied to be exploited by CP solvers.

2.5 Non Linear

2.5.1 Definition

Nonlinear Programming (NLP) refers to the process of solving optimization problems where the objective function and/or at least one constraint is nonlinear. These problems are generally more difficult to solve than their linear counterparts due to issues such as non-convexity, multiple local optima, and complex feasible regions.

A general NLP problem is formulated as:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & x \in \mathbb{R}^n \end{aligned}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear objective function, $g_i(x)$ are inequality constraints.

Solving NLPs typically involves iterative numerical methods, such as:

- Gradient-based methods (e.g., Sequential Quadratic Programming, Interior Point Methods)
- Derivative-free methods (e.g., Genetic Algorithms, Simulated Annealing)
- Convex optimization techniques for special cases when convexity is present

NLP has broad applications in engineering design, economics, machine learning, control systems, and energy optimization, see Nocedal et al. (2006) [25] and Bertsekas (1999) [33]. Conditions for optimality, such as the Karush-Kuhn-Tucker (KKT) conditions (1951) [34], play a central role in theoretical analysis and algorithm development.

2.5.2 Symmetries in NLP

Similar to MIP, symmetry can cause redundant search and slow convergence. Symmetry detection and breaking techniques from MIP can be adapted. Nonlinearities add complexity to symmetry exploitation.

2.6 Metaheuristics

2.6.1 Definition

Metaheuristics are high-level, problem-independent algorithmic frameworks designed to find good-quality solutions to complex optimization problems, especially when exact methods are impractical due to problem size or complexity.

They balance exploration (searching new areas) and exploitation (refining known good solutions) through iterative improvement, randomization, and adaptive mechanisms.

Metaheuristics are divided into subcategories:

- Evolutionary Algorithms (EAs): Inspired by natural evolution, e.g., Genetic Algorithms (GA).
- Swarm Intelligence: Inspired by collective behavior, e.g., Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO).
- Local Search and Neighborhood Search: e.g., Simulated Annealing (SA), Tabu Search (TS).
- Hybrid Methods: Combine elements of different metaheuristics or combine metaheuristics with exact methods.

Metaheuristics recently have gained a bad name. The widespread introduction of metaphor-based metaheuristics has led to a surge of algorithms whose novelty and scientific grounding are often questionable, see Sorensen (2013) [35]. There is even a hilarious Gihub page titled Evolutional Computation Bestiary[1] that is trying to collect all the metaphor-based metaheuristics. The arguments against this situation are valid but this should not discourage anyone from trying out metaheuristics. They are proven to be effective in a variety of situations and complex scenarios.

Metaheuristics do not guarantee optimality but often find near-optimal solutions efficiently. They are widely applicable across domains: scheduling, routing, machine learning, design, bio-informatics. Parameters and operators are often tuned to problem specifics. They have been proven effective on large, nonlinear, multimodal, or combinatorial problems. For overviews in Metaheuristics, see Blum et al. (2003) [36], Glover and Kochenberger (2003) [37], and Talbi (2009) [38].

[1] https://fcampelo.github.io/EC-Bestiary/

2.6.2 Symmetries in Metaheuristics

Symmetries in metaheuristics refer to situations where multiple solutions are equivalent under some transformation (like variable permutations), causing the search algorithm to explore redundant or equivalent regions of the solution space. This can slow convergence or lead to wasted computational effort. A study on integrating symmetry-breaking techniques in evolutionary algorithms was presented by Morris and Segura (2014) [39].

As in MIP the causes for symmetry remain the same their effect however on Metaheuristics differs:

- Redundant Search: Without addressing symmetry, metaheuristics may repeatedly explore symmetric solutions.
- Premature Convergence: Symmetry can cause the search to get stuck cycling among equivalent solutions.
- Reduced Diversity: Solution diversity decreases if symmetric solutions dominate the population or candidate set.

Symmetry handling is consequently different:

- Symmetry-Breaking Operators: Custom crossover, mutation, or neighborhood operators designed to avoid generating symmetric solutions.
- Diversity Preservation: Maintaining a diverse population to reduce the chance of getting trapped in symmetric regions.
- Memory Structures: Using tabu lists or archives to prevent revisiting symmetric solutions.
- Problem Reformulation: Incorporating symmetry-breaking constraints or reformulating the problem to minimize symmetry.

2.7 Quadratic unconstrained binary optimization

2.7.1 Definition

Quadratic Unconstrained Binary Optimization (QUBO) is a mathematical formulation used to express a wide range of combinatorial optimization problems. A QUBO problem seeks to minimize a quadratic objective function over binary variables without any explicit constraints. QUBO can express constrained problems by incorporating penalty terms into the quadratic objective.

The standard QUBO formulation is:

$$\begin{aligned} \text{minimize} \quad & x^T Q x \\ \text{subject to} \quad & x \in \{0,1\}^n \end{aligned}$$

where x is a binary vector of decision variables and $Q \in \mathbb{R}^{n \times n}$ is a symmetric matrix of coefficients. The objective function may include both linear and quadratic terms due to the form of Q.

QUBO is equivalent to Ising models in statistical physics and has applications in combinatorial optimization, machine learning, finance, and particularly in quantum annealing and adiabatic quantum computing, Ising formulations of many NP problems are presented by Lucas (2014) [40]. The QUBO formulation serves as the standard input for quantum annealers like those developed by D-Wave Systems, for a description of adiabatic quantum computation [41].

Many NP-hard problems, including Max-Cut, Graph Coloring, and Set Packing, can be reformulated as QUBO problems see Glover et al. (2018) [42]. Methods for solving QUBO problems include classical metaheuristics (e.g., Tabu Search, Simulated Annealing), semi-definite programming relaxations, and emerging quantum hardware approaches [43]. All MIP and many combinatorial problems can be reformulated as QUBO. QUBO generalizes unconstrained binary quadratic programming.

A nice introduction to the subject can be found at Glover et al. (2022) [44]. More specifically, QUBO models have been tried for several scheduling and timetabling problems by Stollenwerk et al. (2016) [45]. For example the nurse scheduling problem has been addressed using QUBO by Ikeda et al. (2019) [46]. Other examples can be found in Castillo et al. (2022) [47], Huang et al. (2024) [48].

Another resource that is worth mentioning is: List of QUBO formulations[2] which presents a list of QUBO formulations for several optimization problems.

Quantum computing is a fascinating relatively new computing paradigm that holds the promise of surpassing the limits of computation that currently exist. It is based on a new non Von Neumann architecture and several technology companies invest large amounts of money and resources in an effort to realize such systems. D-Wave is a leading company for quantum computing and in this experiments I use the so-called hybrid solver of D-Wave for the experiments. An evaluation of quantum and hybrid solvers for combinatorial problems can be found by Bertuzzi et al. [49].

[2] https://blog.xa0.de/post/List-of-QUBO-formulations

2.7.2 Symmetries in QUBO

Symmetry arises when permuting variables leaves the quadratic form invariant:

$$x^T Q x = \pi(x)^T Q \pi(x)$$

Such symmetries can be exploited to reduce problem size or improve solver efficiency when solved on a classical computer or improve the solvers accuracy when solved with non von Neumann architectures. In the case of annealers where couplings i.e., how many connections exist between the qubit variables, or how many constants are non zero in the objective function if you prefer, is especially important. Eliminating these symmetries can result in less decision variables and couplings.

2.8 Boolean satisfiability problem

2.8.1 Definition

Boolean Satisfiability (SAT) is the problem of determining whether there exists an assignment of truth values to Boolean variables that makes a given propositional logic formula true. It is the first problem that was proven to be NP-complete by Cook (1971) [50] and plays a central role in automated theorem proving, theoretical computer science, artificial intelligence, and formal methods. Modern SAT solvers efficiently handle very large problem instances with millions of variables and clauses. An overview on SAT is provided by Biere et al (2009) [51].

A SAT problem is typically represented in Conjunctive Normal Form (CNF), where a formula is a conjunction of clauses, and each clause is a disjunction of literals. Formally, given a Boolean formula $\phi(x_1, \ldots, x_n)$, the SAT problem asks whether there exists an assignment $x_i \in \{0, 1\}$ such that ϕ evaluates to true.

A simple CNF formula:

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3)$$

is satisfiable, for example with $x_1 = 1, x_2 = 0, x_3 = 1$.

Modern SAT solvers are based on conflict-driven clause learning (CDCL) Marques-Silva et al. (2009) [52] and utilize various heuristics for variable selection, backtracking, and clause learning. Despite the NP-completeness of the problem, solvers like MiniSAT, Eén et al. (2003) [53] and Glucose, Audemard et al. (2018) [54] perform efficiently on large-scale industrial instances.

SAT has broad applications in hardware and software verification, model checking,

automated planning, cryptanalysis, and as a foundation for other decision problems such as SMT (Satisfiability Modulo Theories) see Barrett et al. [55] and MaxSAT.

2.8.2 Symmetries in SAT

Symmetry in SAT instances can cause solvers to explore redundant branches. Symmetry breaking techniques (adding constraints or using symmetry-aware heuristics) improve solver efficiency. Tools for detecting symmetries include graph automorphism software like nauty and saucy exist. Concept from CP like the ones presented by Gent and Smith (2000) [32] can also be extended to SAT.

2.9 Solvers

It is difficult or even unfair to link specific solvers to specific methods as usually they are capable of executing more than a single method. A non-exhaustive list is presented in Table 2.1. Only single objective problems without uncertainty are considered here as this is the type of problems this thesis examines.

Table 2.1: Solvers

Solver	LP	MIP	MIQP	MINLP	CP
ILOG CPLEX[1]	✓	✓	✓		✓
Gekko[2]	✓	✓	✓	✓	
Gurobi[3]	✓	✓	✓	✓	
Hexaly[4]	✓	✓	✓	✓	
Highs[5]	✓	✓	✓		
Insideopt-seeker[6]	✓	✓	✓	✓	
Knitro[7]	✓	✓	✓	✓	
MOSEK[8]	✓	✓	✓		
OR-Tools[9]	✓	✓			✓
SCIP[10]	✓	✓	✓	✓	

3 4 5 6 7 8 9 10 11 12

[3]https://www.ibm.com/products/ilog-cplex-optimization-studio
[4]https://gekko.readthedocs.io/en/latest
[5]www.gurobi.com
[6]www.hexaly.com
[7]https://highs.dev
[8]https://insideopt.com
[9]www.artelys.com/solvers/knitro
[10]www.mosek.com
[11]https://developers.google.com/optimization
[12]www.scipopt.org

Different packages exist for metaheuristics for Python MEALPY[13] and DEAP[14], for Julia Optimization.jl[15] among others.

For QUBO D-Wave[16] and Qiskit[17] are prominent at this time.

For SAT CaDiCaL[18], Glucose[19], MaxSAT Evaluations[20], and MiniSat[21] can serve as starting points.

[13]https://mealpy.readthedocs.io/en/latest/pages/general/introduction.html
[14]https://deap.readthedocs.io/en/master
[15]https://docs.sciml.ai/Optimization/stable
[16]www.dwavequantum.com
[17]https://www.ibm.com/quantum/qiskit
[18]https://fmv.jku.at/cadical
[19]https://www.labri.fr/perso/lsimon/research/glucose
[20]https://maxsat-evaluations.github.io
[21]http://minisat.se

Chapter 3

Case Study: Uncapacitated Examination Timetabling Problem

The Uncapacitated Examination Timetabling Problem is presented in this Chapter. A MIP approach augmented with Metaheuristics and CP, leads to the first proven optimal result of an instance. Additionally, a new dataset is generated and tested in a Quantum Annealer.

3.1 Problem description

The Uncapacitated Examination Timetabling Problem (UETP) is a classic timetabling problem. The task is to schedule examinations in available periods such as that no student has to take more than one examination in each period. Moreover, the resulting timetable should have adequate distances among examinations for all students so as to promote better preparation and less anxiety. Carter et al. (1996) [56], provided 13 real world instances which became known as the Carter datasets and are since used frequently as benchmarks.

Each UETP instance contains information about the set of examinations that each student is enrolled in. Each instance has a specific number of periods that can be used to schedule the examinations to. The single hard constraint is that no student is allowed to participate in more than one examination per period. To allow time for each student to study between his examinations, for each student s, for each pair of examinations taken by s, a penalty of 16 is imposed if the two examinations occur in adjacent time slots (called distance 1), penalty 8 is imposed for distance 2, 4 for distance 3, 2 for distance 4, and 1 for distance 5.

The natural way to represent an instance is as a pair consisting of the number of available periods P and an undirected weighted graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ where each vertex in $\mathbb{V}$ is an examination and each edge in $\mathbb{E}$ connects two examinations with common students. The weight of each edge is the number of common students for the examinations it connects.

3.1.1 UETP formulation terms

In this section, terms that are used throughout later are defined. Set $\mathbb{S}$ is the set of students, set $\mathbb{X}$ is the set of examinations and set $\mathbb{P}$ is the set of periods, arranged in $1..P$ consecutive time-slots. For each student $s \in \mathbb{S}$, $\mathbb{X}_s$ is the subset of examinations that student s is enrolled to. As mentioned before $\mathbb{V}$ is the set of vertices and $\mathbb{E}$ is the set of edges of the corresponding graph $\mathbb{G}$. The number of common students between two examinations $x_i, x_j \in \mathbb{X}$ is given by w_{x_i,x_j}, while the number of enrolled

students in each examination x_i is given by r_{x_i}. Finally, b_{x_i} is the set of examinations that have common students with examination x_i.

3.2 Related Work

Early foundational work by Carter et al. (1996) [56] introduced a standardized dataset and benchmark problems that have been extensively used for evaluation in the literature. These datasets and the associated performance metrics laid the groundwork for comparative studies in the field.

Heuristic and metaheuristic approaches have dominated the research landscape for UETP. Techniques such as simulated annealing, tabu search, and genetic algorithms have been widely applied. For instance, Burke and Newall (1999) [57] applied a heuristic-based approach that combined hill-climbing with heuristic ordering, achieving competitive results on benchmark datasets.

More advanced metaheuristics, such as ant colony optimization and hybrid evolutionary algorithms, have also been explored. Socha, Knowles, and Samples (2002) [58] introduced a MAX-MIN Ant System tailored for the UETP, which showed promising results compared to traditional heuristics.

Graph coloring is another common modeling framework for the UETP, where exams are treated as vertices and student conflicts are represented as edges. Di Gaspero and Schaerf (2001) [59] investigated local search techniques within this graph-based representation, highlighting the effectiveness of neighborhood structures and move operators .

In recent years, hybrid models combining integer programming with metaheuristics have gained attention for their balance of solution quality and computational efficiency. Pillay (2014) [60] provides a comprehensive survey of these hybrid approaches, comparing the strengths and weaknesses of different algorithms across multiple variants of the examination timetabling problem.

Despite the lack of room constraints in UETP, soft constraints such as exam spread (spacing exams for individual students), compactness, and fairness continue to drive algorithmic improvements. The continued evolution of benchmark datasets and objective functions reflects the dynamic nature of this research area.

In [61] our team presented a novel way of estimating lower bounds for UETP instances was proposed. Ideas about symmetry elimination, problem decomposition and cleansing of the instances were also presented there.

3.3 Datasets

The standard benchmark dataset for UETP is Carter's dataset (a.k.a. Toronto dataset). Those instances were contributed in [56] back in 1996 and since then were used in many papers. Recently, 19 new instances that are modified versions of other more complex formulations, were added by Bellio et al. [62]. All of them are publicly available in `https://opthub.uniud.it/` which is a site that hosts definitions, datasets and solutions of several timetabling problems that have attracted the interest of the timetabling community.

The characteristics of the instances used in this paper are shown in Table 3.1. Conflict density is a metric that is computed by dividing the number of edges of the problem's corresponding graph by $n(n-1)/2$, where n is the number of vertices. Moreover, the table presents the best known values that were obtained by solutions that was downloaded from `https://opthub.uniud.it/` in April 2022. Costs assume integer values and since the problem is of minimization nature, lower values are favored. Normalized costs are shown in the rightmost column of the table and are computed by dividing each integer cost by the corresponding number of students.

Table 3.1: UETP: Problem Datasets

Instance id	Exams	Students	Periods	Conflict density	Best known cost	Best known normalized cost
car92	543	18419	32	0.137986	67084	3.6421
car91	682	16925	35	0.128386	71727	4.2379
ear83	190	1125	24	0.266945	36473	32.4204
hec92	81	2823	18	0.420679	28325	10.0337
kfu93	461	5349	20	0.055579	68462	12.7990
lse91	381	2726	18	0.062592	26643	9.7737
pur93	2419	30029	42	0.029495	120144	4.0009
rye93	486	11483	23	0.075279	89999	7.8376
sta83	139	611	13	0.143989	95947	157.0327
tre92	261	4360	23	0.180696	33094	7.5904
uta92	622	21266	35	0.125557	62675	2.9472
ute92	184	2749	10	0.084937	68090	24.7690
yor83	181	941	21	0.288889	32375	34.4049
ITC2007_1	607	7883	54	0.050495	5628	0.7139
ITC2007_2	870	12484	40	0.011695	1538	0.1232
ITC2007_3	934	16365	36	0.026187	20768	1.2690
ITC2007_4	273	4421	21	0.149968	47869	10.8276
ITC2007_5	1018	8719	42	0.008693	1567	0.1797
ITC2007_6	242	7909	16	0.061555	30343	3.8365
ITC2007_7	1096	13795	80	0.019323	262	0.0190
ITC2007_8	598	7718	80	0.045489	409	0.0530
ITC2007_9	169	624	25	0.078402	2909	4.6619
ITC2007_10	214	1415	32	0.049713	12184	8.6106
ITC2007_11	934	16365	26	0.026187	54347	3.3209
ITC2007_12	78	1653	12	0.184482	10631	6.4313
D1-2-17	281	37	38	0.053254	2428	65.6216
D5-1-17	277	43	45	0.087166	3653	84.9535
D5-1-18	306	49	45	0.066560	3245	66.2245
D5-2-17	344	43	45	0.092447	8362	194.4651
D5-2-18	425	47	59	0.083629	6619	140.8298
D5-3-18	132	43	22	0.081309	1406	32.6977
D6-1-18	511	57	60	0.059975	9793	171.8070
D6-2-18	539	57	78	0.067639	7883	138.2982

3.4 Symmetries

This section explores the structure of the problem and identifies underlying symmetries. It suggests ways to exploit these symmetries, aiming to reduce the complexity of the search space. Both exact and approximate solvers can benefit from symmetry breaking.

3.4.1 Bidirectional symmetry

It can be observed that if the period p_x that each examination x is scheduled to, changes to $P - p_x$, then we effectively get the original solution reversed. Since the cost is computed based on the distance among periods of scheduled examinations, it stays unaffected when reverse occurs, as demonstrated in equation 3.1.

$$\left|p_{x_i} - p_{x_j}\right| = \left|P - p_{x_i} - (P - p_{x_j})\right| \quad \forall x_i, x_j \in \mathbb{X} \tag{3.1}$$

So, in order to break this type of symmetry, two examinations x_i and x_j with common students can be selected and impose inequality 3.2.

$$p_{x_i} < p_{x_j} \quad \forall x_i, x_j \in \mathbb{X} : (x_i, x_j) \in \mathbb{E} \tag{3.2}$$

3.4.2 Interchangeable examinations

Let $\mathbb{I}$ be a set of sets where all examinations in a set are not adjacent with other examinations in the same set and each examination has edges to the same other examinations with equal weight values. In mathematical terms examinations x_i and x_j with neighbor sets b_{x_i} and b_{x_j} respectively belong to set $\mathbb{I}$ if equations in 3.3 hold. All examinations in each set of $\mathbb{I}$ must be scheduled at the same "best" period in an optimal solution. Since the examinations of each set are not in conflict, and each examination conflicts with exactly the same other examinations, there is no benefit in positioning them in different periods.

$$\begin{aligned} & b_{x_i} \equiv b_{x_j} \\ & w_{x_i,x_n} = w_{x_j,x_n} \quad x_n \in b_{x_i} \end{aligned} \tag{3.3}$$

The symmetry involving examinations in $\mathbb{I}$ can be broken by imposing constraints that schedule at the same period, all examinations of each set in it.

Another symmetry is revealed by defining set $\mathbb{I}^+$ as a set of sets where all examinations in a set are adjacent with all other examinations in the same set and each examination has edges to the same other examinations using the same weight values. The set $\mathbb{I}^+$ is formally defined in equations 3.4, where the symmetric difference of neighbor sets b_{x_i} and b_{x_j} of every pair of examinations x_i and x_j in a set of $\mathbb{I}^+$ is equal to $\{x_i, x_j\}$.

$$\begin{aligned} & b_{x_i} \triangle b_{x_j} = \left\{x_i, x_j\right\} \\ & w_{x_i,x_n} = w_{x_j,x_n} \quad \forall x_n \in b_{x_i} : x_n \neq x_j \end{aligned} \tag{3.4}$$

The symmetry involving examinations in $\mathbb{I}^+$ can be broken by imposing constraints to artificially order (e.g. by id) examinations belonging to the same set. In Figure 3.1, two examinations are symmetrical, A with B.

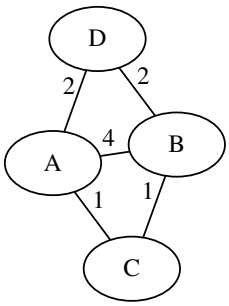

Figure 3.1: Two symmetrical examinations.

3.5 Mixed Integer Programming

As optimality is the main concern the first thoughts that come to mind are Linear Programming and Mixed Integer Programming. The mathematical model described below can solve an UETP instance, provided that the instance size is manageable. For a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ where vertices $\mathbb{V}$ serve as the exams, each edge in $\mathbb{E}$ means that two examinations have common students. The weight of an edge W_{v_1,v_2} connecting vertices v_1 and v_2 is equal to the number of common students these examinations have. P is the number of available periods.

The integer decision variables p_v in Equation 3.5 denote the period each examination v will take place while the derived binary decision variables in Equation 3.6 help us to activate or deactivate penalties in the objective function in Equation 3.7. In particular variable $y16_{v_1,v_2}$ assumes value 1 when examinations v_1 and v_2 are positioned 1 period away from each other or 0 otherwise. Likewise, $y8_{v_1,v_2}$, $y4_{v_1,v_2}$, $y2_{v_1,v_2}$ and $y1_{v_1,v_2}$ are for distances of 2, 3, 4 and 5 periods correspondingly, between examinations v_1 and $v2$. The constraint in Equation 3.8 forces examinations with common students to take place in different periods. Equation 3.9 forces binary decision variables in Equation 3.6 to indicate the distance between two exams. Equations 3.8 and 3.9 are obviously non linear, so the logical constraints feature of IBM ILOG CPLEX was used to model both of them. In particular, for Equation 3.8 the operator != having meaning "different from" was used, while for Equation 3.9 the operator == means "equivalence". IBM ILOG CPLEX uses a method called logical constraints extraction that automatically transforms logical constraints into equivalent linear formulations. This transformation involves automatic creation of new variables and constraints. Note that for each equality of Equation 3.9, the right part consists of adding two equivalences that could not possible be both true at the same time (i.e., the equivalences have the same left part but different right parts). This ensures that the variable at the left part of the equality assumes a binary value. Equation 3.10 allows only one of the penalty indicating variables in Equation 3.6 to be active at

any time. This constraint is redundant but its presence seems to help the solver in reaching better solutions.

$$p_v \in [0, P) \quad \forall v \in \mathbb{V} \tag{3.5}$$

$$\begin{aligned} y16_{v_1,v_2} &\in \{0,1\} \quad \forall (v_1, v_2) \in \mathbb{E} \\ y8_{v_1,v_2} &\in \{0,1\} \quad \forall (v_1, v_2) \in \mathbb{E} \\ y4_{v_1,v_2} &\in \{0,1\} \quad \forall (v_1, v_2) \in \mathbb{E} \\ y2_{v_1,v_2} &\in \{0,1\} \quad \forall (v_1, v_2) \in \mathbb{E} \\ y1_{v_1,v_2} &\in \{0,1\} \quad \forall (v_1, v_2) \in \mathbb{E} \end{aligned} \tag{3.6}$$

$$\begin{aligned} \min \quad & 16 * \sum_{v_1,v_2 \in \mathbb{E}} W_{v_1,v_2} * y16_{v_1,v_2} \quad +8 * \sum_{v_1,v_2 \in \mathbb{E}} W_{v_1,v_2} * y8_{v_1,v_2} \\ & + 4 * \sum_{v_1,v_2 \in \mathbb{E}} W_{v_1,v_2} * y4_{v_1,v_2} \quad +2 * \sum_{v_1,v_2 \in \mathbb{E}} W_{v_1,v_2} * y2_{v_1,v_2} \\ & + \sum_{v_1,v_2 \in \mathbb{E}} W_{v_1,v_2} * y1_{v_1,v_2} \end{aligned} \tag{3.7}$$

Subject to:

$$p_{v_1} \neq p_{v_2} \quad \forall (v_1, v_2) \in \mathbb{E} \tag{3.8}$$

$$\begin{aligned} y16_{v_1,v_2} &= (p_{v_1} - p_{v_2} = 1) + (p_{v_1} - p_{v_2} = -1) \quad \forall (v_1, v_2) \in \mathbb{E} \\ y8_{v_1,v_2} &= (p_{v_1} - p_{v_2} = 2) + (p_{v_1} - p_{v_2} = -2) \quad \forall (v_1, v_2) \in \mathbb{E} \\ y4_{v_1,v_2} &= (p_{v_1} - p_{v_2} = 3) + (p_{v_1} - p_{v_2} = -3) \quad \forall (v_1, v_2) \in \mathbb{E} \\ y2_{v_1,v_2} &= (p_{v_1} - p_{v_2} = 4) + (p_{v_1} - p_{v_2} = -4) \quad \forall (v_1, v_2) \in \mathbb{E} \\ y1_{v_1,v_2} &= (p_{v_1} - p_{v_2} = 5) + (p_{v_1} - p_{v_2} = -5) \quad \forall (v_1, v_2) \in \mathbb{E} \end{aligned} \tag{3.9}$$

$$y16_{v_1,v_2} + y8_{v_1,v_2} + y4_{v_1,v_2} + y2_{v_1,v_2} + y1_{v_1,v_2} \leq 1 \quad \forall (v_1, v_2) \in \mathbb{E} \tag{3.10}$$

In order to break a symmetry of the problem we enforce an order over the examinations belonging to each set. This is formulated in Equation 3.11, where members of each set $\mathbb{S}$ of the sets in $\mathbb{I}^+$ are ordered among each other.

$$v_i \leq v_{i+1} \quad \forall v_i \in \mathbb{S} : i \in 1 \ldots \mathbb{S} - 1, \quad \forall \mathbb{S} \in \mathbb{I}^+ \tag{3.11}$$

Table 3.2: Results for the Carter dataset

Instance id	Exams	Students	Conflict Density	Best known cost	Best known normalized cost
car92_1(E533_S18328_ID1)	533	18328	0.143139	67084	3.6421
car91_1(E669_S16750_ID1)	669	16750	0.133375	71727	4.2379
ear83_1(E190_S1125_ID1)	190	1125	0.266945	36473	32.4204
hec92_1(E81_S2823_ID1)	81	2823	0.420679	28325	10.0337
kfu93_1(E428_S5194_ID1)	428	5194	0.064326	68462	12.7990
lse91_1(E378_S2724_ID1)	378	2724	0.063576	26643	9.7737
pur93_1(E2336_S29766_ID1)	2336	29766	0.031566	120144	4.0009
rye93_1(E485_S11425_ID1)	485	11425	0.075590	89999	7.8376
sta83_1(E30_S162_ID1)	30	162	0.717241	*16002	*26.1899
sta83_2(E47_S210_ID3)	47	210	0.351526	*47250	*77.3322
sta83_3(E62_S239_ID4)	62	239	0.364357	*32695	*53.5106
tre92_1(E258_S4355_ID1)	258	4355	0.184840	33094	7.5904
uta92_1(E617_S21264_ID1)	617	21264	0.127539	62675	2.9472
ute92_1(E7_S20_ID30)	7	20	0.904762	*645	*0.2346
ute92_2(E177_S2729_ID1)	177	2729	0.090588	67445	24.5344
yor83_1(E181_S941_ID1)	181	941	0.288889	32375	34.4049

3.6 Results

Some problems are decomposed to subproblems. For most instances a number of examinations and students are removed since they are in effect noise. The resulting subproblems are presented in Tables 3.2, 3.3, 3.4. The name of each subproblem follows the pattern d_i_(Ex_Sy_IDz), where d is the name of the originating instance, i is a number that assumes value 1 for the smallest subproblem and is incremented by 1 for each subsequent subproblem (subproblems are ordered by size = number of exams), x is the number of examinations, y is the number of students and z is the smallest examination number that exists in the subproblem. Number z is needed in order to differentiate among subproblems having the same number of examinations and same number of students. This is indeed the case for subproblems D1-2-17_1 and D1-2-17_2 that both have 8 examinations and 1 student but in the first case the identifying examination is 217 while for the second case the identifying examination is 257. Note that the number of examinations and the number of students exclude noise examinations and noise students respectively. Again, the presence of symbol $*$ denotes that the corresponding integer cost is optimal. It should be also noted that the normalized cost is computed by dividing the integer cost by the number of students (including noise ones) that exists in the originating instance. We opt to use two values for the cost (i.e., an exact integer one and an approximate decimal one) since the values in the relevant bibliography are decimal, but the integer cost is needed for precise results.

Table 3.3: Results for the ITC dataset

Instance id	Exams	Students	Conflict Density	Best known cost	Best known normalized cost
ITC2007_1_1(E582_S7798_ID1)	582	7798	0.054563	5628	0.7139
ITC2007_2_1(E9_S33_ID396)	9	33	0.888889	*0	*0.0000
ITC2007_2_2(E623_S9636_ID1)	623	9636	0.020856	1538	0.1232
ITC2007_3_1(E810_S15726_ID1)	810	15726	0.034214	20768	1.2690
ITC2007_4_1(E273_S4421_ID1)	273	4421	0.149968	47869	10.8276
ITC2007_5_1(E11_S9_ID434)	11	9	0.690909	*0	*0.0000
ITC2007_5_2(E13_S41_ID206)	13	41	0.487179	*0	*0.0000
ITC2007_5_3(E14_S263_ID120)	14	263	0.989011	189	0.0217
ITC2007_5_4(E637_S7559_ID1)	637	7559	0.018236	1378	0.1580
ITC2007_6_1(E4_S12_ID5)	4	12	1.000000	*33	*0.0042
ITC2007_6_2(E7_S75_ID122)	7	75	0.666667	*7	*0.0009
ITC2007_6_3(E27_S210_ID9)	27	210	0.293447	146	0.0185
ITC2007_6_4(E189_S7386_ID3)	189	7386	0.093662	30157	3.8130
ITC2007_7_1(E18_S143_ID178)	18	143	0.732026	*0	*0.0000
ITC2007_7_2(E720_S10034_ID2)	720	10034	0.040604	262	0.0190
ITC2007_8_1(E497_S7388_ID1)	497	7388	0.062764	409	0.0530
ITC2007_9_1(E143_S603_ID2)	143	603	0.105683	2909	4.6619
ITC2007_10_1(E7_S81_ID1)	7	81	1.000000	*196	*0.1385
ITC2007_10_2(E9_S91_ID78)	9	91	0.888889	*14	*0.0099
ITC2007_10_3(E11_S29_ID87)	11	29	1.000000	*54	*0.0382
ITC2007_10_4(E12_S111_ID121)	12	111	0.984848	1021	0.7216
ITC2007_10_5(E15_S59_ID200)	15	59	0.857143	292	0.2064
ITC2007_10_6(E16_S220_ID133)	16	220	0.958333	878	0.6205
ITC2007_10_7(E16_S124_ID166)	16	124	0.800000	338	0.2389
ITC2007_10_8(E16_S56_ID51)	16	56	0.550000	76	0.0537
ITC2007_10_9(E17_S143_ID149)	17	143	0.757353	836	0.5908
ITC2007_10_10(E19_S208_ID13)	19	208	0.964912	2356	1.6650
ITC2007_10_11(E23_S215_ID98)	23	215	0.909091	6123	4.3272
ITC2007_11_1(E841_S15857_ID1)	841	15857	0.031989	54347	3.3209
ITC2007_12_1(E5_S62_ID35)	5	62	0.900000	*22	*0.0133
ITC2007_12_2(E69_S1464_ID1)	69	1464	0.232310	10609	6.4180

Table 3.4: Results for the D dataset

Instance id	Exams	Students	Conflict Density	Best known cost	Best known normalized cost
D1-2-17_1(E8_S1_ID217)	8	1	1.000000	*5	*0.1351
D1-2-17_2(E8_S1_ID257)	8	1	1.000000	*5	*0.1351
D1-2-17_3(E10_S1_ID119)	10	1	1.000000	*17	*0.4595
D1-2-17_4(E11_S1_ID218)	11	1	1.000000	*26	*0.7027
D1-2-17_5(E12_S1_ID189)	12	1	1.000000	*36	*0.9730
D1-2-17_6(E13_S2_ID100)	13	2	0.538462	*0	*0.0000
D1-2-17_7(E14_S1_ID173)	14	1	1.000000	*62	*1.6757
D1-2-17_8(E18_S1_ID1)	18	1	1.000000	*150	*4.0541
D1-2-17_9(E18_S1_ID51)	18	1	1.000000	*150	*4.0541
D1-2-17_10(E28_S2_ID7)	28	2	0.592593	*190	*5.1351
D1-2-17_11(E120_S18_ID44)	120	18	0.164286	1787	48.2973
D5-1-17_1(E11_S3_ID98)	11	3	1.000000	*48	*1.1163
D5-1-17_2(E13_S3_ID99)	13	3	0.846154	*12	*0.2791
D5-1-17_3(E200_S34_ID5)	200	34	0.158945	3593	83.5581
D5-1-18_1(E9_S2_ID263)	9	2	1.000000	*8	*0.1633
D5-1-18_2(E13_S3_ID88)	13	3	0.846154	*12	*0.2449
D5-1-18_3(E14_S2_ID200)	14	2	0.736264	*10	*0.2041
D5-1-18_4(E223_S41_ID1)	223	41	0.118046	3215	65.6122
D5-2-17_1(E18_S1_ID199)	18	1	1.000000	*108	*2.5116
D5-2-17_2(E324_S42_ID1)	324	42	0.101307	8254	191.9535
D5-2-18_1(E18_S1_ID97)	18	1	1.000000	54	1.1489
D5-2-18_2(E56_S5_ID94)	56	5	0.318182	140	2.9787
D5-2-18_3(E345_S41_ID1)	345	41	0.116144	6425	136.7021
D5-3-18_1(E5_S2_ID40)	5	2	1.000000	*6	*0.1395
D5-3-18_2(E7_S1_ID59)	7	1	1.000000	*18	*0.4186
D5-3-18_3(E118_S40_ID3)	118	40	0.097349	1382	32.1395
D6-1-18_1(E12_S1_ID470)	12	1	1.000000	*7	*0.1228
D6-1-18_2(E22_S2_ID85)	22	2	0.636364	*32	*0.5614
D6-1-18_3(E403_S52_ID1)	403	52	0.092947	9754	171.1228
D6-2-18_1(E14_S1_ID1)	14	1	1.000000	*1	*0.0175
D6-2-18_2(E22_S1_ID343)	22	1	1.000000	*56	*0.9825
D6-2-18_3(E493_S54_ID3)	493	54	0.077904	7826	137.2982

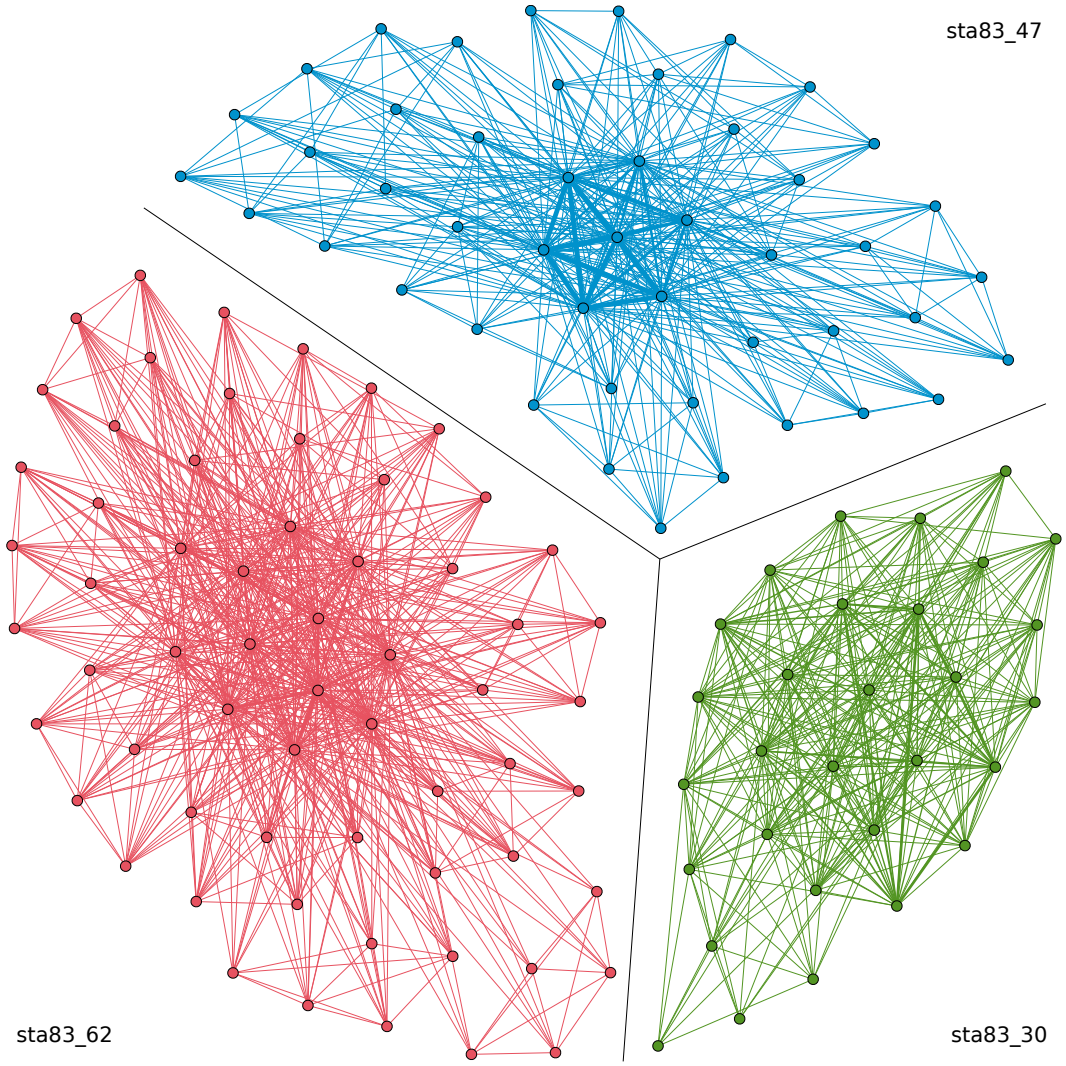

Figure 3.2: Disconnected components of sta83. The weight of each edge is indicated by its thickness.

3.7 sta83 optimal solution

No optimality has ever been proved for any Carter's dataset instance until now. In this section we show that the solution for sta83 having value 95947 (95947/611=157.0327 in decimal value, where 611 is the total number of students for sta83) which appears in many papers is indeed optimal.

Instance sta83 consists of 139 exams, 13 periods and has a relatively low conflict density of value 0.14. The instance is comprised of 3 disconnected components as shown in Fig. 3.2.

We can divide the problem into three independent subproblems because these components are disconnected. That is, there are three unique groups of students, each of which does not have an examination in common with the other two groups, allowing us to work on each component independently. The sum of these answers would be

the optimal solution provided that all three of them are solved optimally. Motivated by the prospect of proving optimality for a Carter's dataset instance, we focused our attention on this task, and we managed to optimality solve each subproblem using a different approach, resulting in a novel way of handling high conflict-density components.

3.7.1 Component sta83_62

This is the largest component of sta83, having 62 examinations and a conflict density of 0.36. We tried to solve it using the model described in Section 3.5 using the IBM ILOG CPLEX IP [63] solver. Unfortunately, after several hours the solver was unable to prove optimality. We tried to warm start the solution with the current best solution and have set the MIP emphasis parameter first to "emphasize optimality over feasibility" and then to "emphasize moving best bound". Both attempts were unsuccessful.

We noticed that the component has a special structure. It contains 10 sets of examinations with each set consisting of exactly 5 interchangeable examinations. These examinations amount for 50 of the 62 examinations that the component has in total. Details of these sets are presented in Table 3.5. Since interchangeable examinations can freely swap places with each other while keeping the objective value unchanged, the introduction of the symmetry breaking constraints of Equation 3.10 greatly improved the solver's efficiency in proving the optimal solution.

We also noticed that 3 examinations existed (72, 133, 136) in the graph that had connections with all other exams. So, we tried an approach that fixed these 3 examinations in specific periods and then tried to solve the remaining problem using IBM ILOG CPLEX. This time, the result was successful, the solver was able to return a result, either optimal or infeasible in a few minutes. It should be noted that infeasibility occurs because the cost of the best known solution is used as a cutoff constraint. So, we had only to try all possible places for positioning the 3 examinations and then solve the resulting problem. Since there are only 13 periods in instance sta83, this would mean that only $\binom{13}{3} = 286$ configurations existed that should be multiplied by $\frac{3!}{2}$ since the 3 examinations can occupy the fixed periods in any order (divided by 2 due to the inherent symmetry of the problem).

By exploiting the above observations, IBM ILOG CPLEX IP solver was able to solve each subproblem in a few minutes. After solving all subproblems, the optimal solution for sta83_62 was proved to be 32695. This solution occurred when examinations 72, 133 and 136 were fixed to periods 3, 6 and 8 respectively. The symmetric solution also exists and is produced by fixing examinations 72, 133 and 136 to periods 9,

6 and 4. Of course, many more symmetric solutions exist due to the interchangeable exams.

Table 3.5: Component sta83_62, sets of interchangable examinations and their characteristics.

Set	Degree	Weighted Degree
{17, 38, 58, 85, 120}	8	8
{18, 39, 59, 86, 121}	16	240
{19, 40, 60, 87, 122}	16	264
{20, 41, 61, 88, 123}	15	168
{21, 42, 62, 89, 124}	12	88
{22, 43, 63, 90, 125}	16	160
{23, 44, 64, 91, 126}	15	160
{24, 45, 65, 92, 127}	16	264
{25, 46, 66, 93, 128}	16	280
{26, 47, 67, 94, 129}	16	280

3.7.2 Component sta83_47

This component proved to be the easy part. It consists of 47 examinations and has a conflict density of 0.35. We can estimate a lower bound by adding the minimum cost each student's schedule could possibly inflict. So, for each student in isolation, an IP model is formulated that given only the number of periods and the number of examinations that this student participates, decides about the schedule that results to the minimum possible cost. Of course, since each student is examined in isolation if two students share the same number of examinations then the problem needs to be solved just once. In practice, this is the case for several students. By adding minimum penalties of all students we have a lower bound for this component, which is 42750. The best known solution turns out to have cost equal to the lower bound obtained in this manner. Thus, the optimal solution for this component is 47250.

3.7.3 Component sta83_30

This was the last component to solve. It's the smallest one with just 30 examinations but a high conflict density of 0.72. With high hopes since just the smallest piece of the puzzle was missing, we were surprised to find out that to the best of our ability our MIP models were not able to prove an optimal solution. We have tried the same trick that we have used successfully in component sta83_62. We noticed that in the case of sta83_30 there is only one examination (134) that is connected to every other one. So, we tried to fix this examination to each period in turn and then to solve the remaining problems using IBM ILOG CPLEX. Unfortunately, this did not helped

the solver to prove the optimality of the solution. Each subproblem seemed to run forever.

By observing closely the high density graph of this component we came up with the idea of separating examinations with high degrees and examinations with relatively low degrees. In our approach, we isolated the maximum clique, which for this particular instance comprises of 12 examinations and tried to arrange those examinations to the 13 periods leaving one period empty for each possible arrangement.

A significant observation is that irrelevant of the periods that the clique occupies, the possible placements for the remaining examinations will be the same because their possible positions are constrained by the examinations of the clique. By multiplying the number of those possibilities with the number of permutations of the periods we were able to count all possible solutions to be $13! * 109152$ where $13!$ is the number of possible period permutations and 109152 is the number of possible ways to schedule the remaining examinations for the specific component. We aim to find a set of examinations that has minor impact on the cost but at the same time possible final positions of the sets' examinations might be disproportionate large.

3.8 Unconstrained Binary Model

QUBO problems are a specific type of optimization problems where the goal is to find the optimal assignment of binary variables that minimizes or maximizes an objective function subject to no constraints.

The general form of a QUBO objective function can be expressed as follows:

$$\min \quad \sum_{i=1}^{n} q_{ii} x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j$$

where:

- x_i are binary variables.
- n is the number of binary variables.
- q_{ii} represents the linear coefficient associated with variable x_i.
- q_{ij} represents the quadratic coefficient associated with the interaction between variables x_i and x_j.

The model for a QUBO problem will always be the same. What makes the difference is the choice of the values in the QMatrix. For this problem the binary decision vari-

ables assume the value 1 when a specific exam is scheduled in a period. For exams in conflict the corresponding quadratic coefficient is calculated as $F_{p_1,p_2} W_{v_1,v_2}/2$. We divide by two because the QMatrix is symmetric. As the nature of QUBO formulation is inherently unconstrained we choose a large enough number M to impose penalties and incentives in the objective function that can act as constraints. We chose M to equal the sum of all edges multiplied by 16 to ensure that no worse solution exists when you violate the conflicting exams constraint. To provide the incentive to schedule all exams, as dictated by constraint, we set the value of an exam being placed to $-M$ and to M if the exam is placed twice.

We use a minimal problem presented in Figure 3.3 to demonstrate the resulting QMatrix in Table 3.6. Note that this toy example involves 5 examinations and 3 periods.

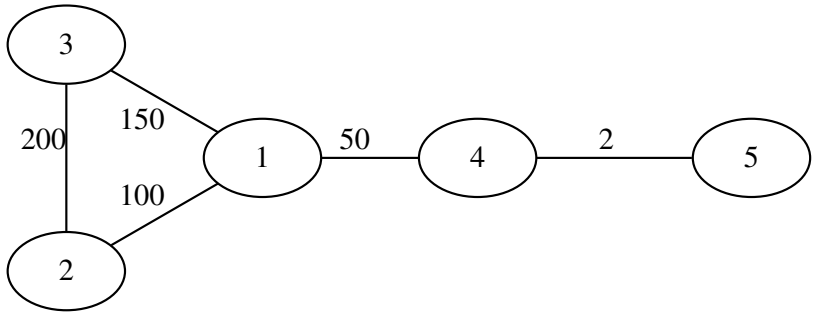

Figure 3.3: UETP: Minimal problem graph (5 examinations, 3 periods).

Table 3.6: UETP: Q Matrix

	E_1P_1	E_1P_2	E_1P_3	E_2P_1	E_2P_2	E_2P_3	E_3P_1	E_3P_2	E_3P_3	E_4P_1	E_4P_2	E_4P_3	E_5P_1	E_5P_2	E_5P_3
E_1P_1	-M	M	M	M	800	400	M	1200	1200	M	400	200	0	0	0
E_1P_2	M	-M	M	800	M	800	1200	M	1200	400	M	400	0	0	0
E_1P_3	M	M	-M	400	800	M	1200	1200	M	200	400	M	0	0	0
E_2P_1	M	800	400	-M	M	M	M	1600	800	0	0	0	0	0	0
E_2P_2	800	M	800	M	-M	M	1600	M	1600	0	0	0	0	0	0
E_2P_3	400	800	M	M	M	-M	800	1600	M	0	0	0	0	0	0
E_3P_1	M	1200	1200	M	1600	800	-M	M	M	0	0	0	0	0	0
E_3P_2	1200	M	1200	1600	M	1600	M	-M	M	0	0	0	0	0	0
E_3P_3	1200	1200	M	800	1600	M	M	M	-M	0	0	0	0	0	0
E_4P_1	M	400	200	0	0	0	0	0	0	-M	M	M	M	16	8
E_4P_2	400	M	400	0	0	0	0	0	0	M	-M	M	16	M	16
E_4P_3	200	400	M	0	0	0	0	0	0	M	M	-M	8	16	M
E_5P_1	0	0	0	0	0	0	0	0	0	M	16	8	-M	M	M
E_5P_2	0	0	0	0	0	0	0	0	0	16	M	16	M	-M	M
E_5P_3	0	0	0	0	0	0	0	0	0	8	16	M	M	M	-M

We can also choose to eliminate the bidirectional symmetry by restricting a single pair of two conflicting exams to be placed in a certain order. If we choose that exam 1 must be scheduled before exam 3 the binary value E_1P_1 is not needed anymore as

exam 1 cannot be placed in the first period and for each combination where exam 1 is scheduled before exam 3 we again provide the value of M to place a heavy penalty if such a combination is selected. The QMatrix with these modifications is presented in Table 3.7.

Table 3.7: UETP: Q Matrix with bidirectional symmetry elimination

	E_1P_2	E_1P_3	E_2P_1	E_2P_2	E_2P_3	E_3P_1	E_3P_2	E_3P_3	E_4P_1	E_4P_2	E_4P_3	E_5P_1	E_5P_2	E_5P_3
E_1P_2	-M	M	800	M	800	1200	M	**M**	400	M	400	0	0	0
E_1P_3	M	-M	400	800	M	1200	1200	M	200	400	M	0	0	0
E_2P_1	800	400	-M	M	M	M	1600	800	0	0	0	0	0	0
E_2P_2	M	800	M	-M	M	1600	M	1600	0	0	0	0	0	0
E_2P_3	800	M	M	M	-M	800	1600	M	0	0	0	0	0	0
E_3P_1	1200	1200	M	1600	800	-M	M	M	0	0	0	0	0	0
E_3P_2	M	1200	1600	M	1600	M	-M	M	0	0	0	0	0	0
E_3P_3	**M**	M	800	1600	M	M	M	-M	0	0	0	0	0	0
E_4P_1	400	200	0	0	0	0	0	0	-M	M	M	M	16	8
E_4P_2	M	400	0	0	0	0	0	0	M	-M	M	16	M	16
E_4P_3	400	M	0	0	0	0	0	0	M	M	-M	8	16	M
E_5P_1	0	0	0	0	0	0	0	0	M	16	8	-M	M	M
E_5P_2	0	0	0	0	0	0	0	0	16	M	16	M	-M	M
E_5P_3	0	0	0	0	0	0	0	0	8	16	M	M	M	-M

3.8.1 Dataset

Different datasets regarding the UETP problem were made public over the years, but the sheer size of the included instances make them to big to fit in current state of the art annealers. While the number of qubits required for some small instances is acceptable, the nature of the problem i.e., the relation of two exams with students in common, results in an increase of the Non Zero Couplings in the matrix that is sent to the solver (usually called a QMatrix) provided to the solver, thus making most of these instances unfit for the annealer.

In order to demonstrate the proof of concept I opted to generate a dataset consisting of 50 small instances able to run on current annealers. To create an instance I randomly choose between 3 and 7 exams and generate a complete graph with them (all of them have students in common) the number of the periods available equals the number of nodes in the complete graph to make the instance compact e.g., there exists no solution with an empty period, then I proceed to add more exams and more conflicts while keeping the number of conflicts under 60. The students in common between the conflicting exams (the weight of their edge) is chosen arbitrarily between 1 and 100. However, this number could be higher as this will not result in more variables.

To test the annealer against the optimal solutions GoogleOR-Tools CP-SAT Solver [64]

is employed to solve the problem instances to optimality. The characteristics and optimal solutions values are presented in Table 3.8.

Table 3.8: UETP: QUBO Instances and characteristics

Instance	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Examinations	20	19	8	13	22	19	20	17	18	20
Periods	6	6	5	5	4	6	7	5	7	6
C.D.[1]	0.23	0.28	0.37	0.55	0.19	0.25	0.24	0.3	0.27	0.27
Optimal	10512	10736	13540	17376	16152	9916	7458	13242	8365	12720
Instance	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Examinations	24	24	20	18	19	15	23	17	27	8
Periods	4	4	6	5	4	5	5	5	7	6
C.D.[1]	0.15	0.15	0.23	0.29	0.25	0.38	0.17	0.33	0.12	0.38
Optimal	16312	13460	10018	13088	16700	12724	9640	11704	8281	12255
Instance	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Examinations	19	19	26	17	20	19	27	23	19	17
Periods	4	5	7	7	6	7	7	7	6	4
C.D.[1]	0.25	0.27	0.14	0.29	0.24	0.24	0.13	0.17	0.24	0.32
Optimal	15592	15628	5316	7464	10445	8301	5718	9265	7700	19680
Instance	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
Examinations	21	25	17	15	22	26	20	17	18	19
Periods	6	6	7	4	4	7	6	5	5	7
C.D.[1]	0.21	0.16	0.33	0.39	0.19	0.14	0.24	0.31	0.29	0.25
Optimal	8749	8834	8780	20364	17412	6735	11691	12356	14434	7221
Instance	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**
Examinations	24	15	17	21	23	8	18	24	19	18
Periods	7	5	5	4	6	6	5	6	6	6
C.D.[1]	0.16	0.44	0.33	0.23	0.17	0.38	0.27	0.18	0.25	0.3
Optimal	5885	15108	18630	24604	8456	9731	7964	8723	9407	9654

[1]Conflict Density.

3.8.2 Experiments and results

Experiments were performed using the hybrid Quantum Annealer provided by D-Wave. A time limit of 20 seconds was given for each problem instance and the results are presented in Table 3.9. The justification for using only 20 seconds of running time per instance is due to the small sizes of the problems and the limited time that the hybrid solver of D-Wave can use the Quantum infrastructure for the non-pay version of D-Wave Leap.

Results show that there is potential for using Quantum Annealers for solving UETP problems. Some results are optimal, while others are near optimal, as can be seen in Figure 3.4 which shows how far from the optimal solution the results for the 50 problem instances are.

Table 3.9: Results

Instance	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Decision Variables	119	113	79	64	87	113	139	84	125	119
Non Zero Couplings	1950	2061	1290	1215	908	1851	2596	1230	2370	2226
Objective	12082	12687	13540	17376	16784	11265	9515	13302	9176	14899
Difference	3.47%	3.26%	11.38%	0.00%	6.62%	0.60%	6.58%	4.12%	3.56%	4.16%
Instance	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Decision Variables	95	95	119	89	75	74	114	84	188	95
Non Zero Couplings	864	864	1950	1310	842	1165	1405	1330	2669	1926
Objective	17504	15336	11244	13996	16700	13312	11346	12630	11224	12255
Difference	0.00%	0.00%	0.96%	3.18%	6.06%	0.11%	2.31%	3.94%	1.76%	2.88%
Instance	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Decision Variables	75	94	181	118	119	132	188	160	113	67
Non Zero Couplings	826	1375	2729	2277	2004	2392	2904	2645	1821	814
Objective	15592	16662	8448	8584	12264	9922	9499	11914	9091	19680
Difference	1.68%	0.00%	1.13%	4.06%	1.90%	7.54%	0.00%	0.00%	1.60%	3.49%
Instance	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
Decision Variables	125	149	118	59	87	181	119	84	89	132
Non Zero Couplings	1971	2229	2524	770	880	2788	2016	1250	1325	2439
Objective	10426	11357	10342	20364	17968	10187	12908	12564	14794	8798
Difference	4.01%	4.45%	12.42%	6.25%	4.14%	0.00%	4.37%	6.25%	4.08%	0.79%
Instance	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**
Decision Variables	167	74	84	83	137	95	89	143	113	107
Non Zero Couplings	2638	1320	1335	946	1977	1890	1250	2214	1857	1974
Objective	8789	15108	18790	24604	11036	10404	8156	11369	11097	11133
Difference	10.20%	2.47%	0.42%	0.62%	4.92%	9.90%	0.00%	0.21%	0.00%	1.67%

[1]Results from hybrid solver.

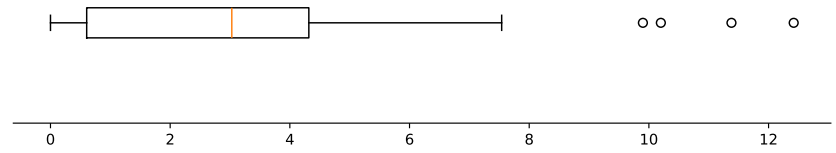

Figure 3.4: Difference in percentage from the optimal solution for 50 problem instances.

3.8.3 Conclusion

The examination of the Uncapacitated Examination Timetabling Problem led to some of the best known solution to public available instances. The first instance to be proven optimal was presented. A new dataset suitable for Quantum Annealers was created. This study resulted in the following publications:

C. Gogos, A. Dimitsas, V. Nastos, and C. Valouxis, "Some insights about the Uncapacitated Examination Timetabling Problem," in *2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, Sep. 2021, pp. 1–7

A. Dimitsas, C. Gogos, C. Valouxis, V. Nastos, and P. Alefragis, "A proven optimal

result for a benchmark instance of the uncapacitated examination timetabling problem," *Journal of Scheduling*, Mar. 2024

A. Dimitsas, P. Alefragis, C. Valouxis, and C. Gogos, "An unconstrained binary model for the Uncapacitated Examination Timetabling Problem," in *PATAT Conference 2024 proceedings of the 14th International Conference of the Practice and Theory of Automated Timetabling*, 2024

Chapter 4

Case Study: Post Enrollment Course Timetabling

In this Chapter the Post Enrollment Course Time Tabling problem is examined. A combination of Simulated Annealing with CP and MIP is employed here.

4.1 Problem Description

A well-known scheduling problem having both theoretical and practical significance is course time-tabling. Curriculum-based course time-tabling (CB-CTT) and post-enrollment course time-tabling (PE-CTT) are two problem variations. While we have information about each student's enrollments in PE-CTT, in CB-CTT these enrollments are "hidden" behind courses. The goal of the PE-CTT variation, which is the subject examined here, is to schedule events in the available time-slots and rooms. Each day has 9 time-slots, and a week consists of 5 days, so 45 time-slots are available for all instances.

Various problem instances for Course Timetabling exist, adhering to several assumptions about the problem. The International Timetabling Competition held in 2002 and its sequel in 2007 [67] provided a set of instances that were later used by several researchers as a common testbed. These are the datasets, ITC2002 and ITC2007, that are used hereafter. For these datasets the hard constraints are:

- Each event must be scheduled in a time-slot and a room.
- Events with common students must be placed at different time-slots.
- A room can host at most one event in each time-slot.
- The capacity and feature requirements of each event must be met by the room that will eventually host it.
- Certain time-slot requirements may apply to specific events.
- There may be relationships of precedence between events.

Note that the two last hard constraints of the above list manifest themselves only in problem instances of ICT2007 and not in ITC2002. On the other hand, the soft constraints are:

- A student attends an event at the last time-slot of a day.
- A student attends three (or more) events in a row on the same day.
- A student attends only one event in a day.

One penalty point is imposed for each of the conditions above.

4.2 Related Work

The PE-CTT variant was more formally introduced during the International Timetabling Competition 2007 (ITC 2007), which provided standardized benchmarks and instances Di Gaspero et al., (2007) [67]. These benchmarks facilitated consistent comparisons and inspired a surge in heuristic and metaheuristic methods.

Several notable approaches have been proposed. Müller (2009) [68] applied a hybrid method combining constraint-based reasoning with Simulated Annealing, showing robust performance across multiple ITC 2007 instances. Lü and Hao (2010) [69] introduced a memetic algorithm that hybridized genetic algorithms with local search, achieving state-of-the-art results on many PE-CTT benchmarks.Mathematical programming formulations have also been explored. Lach and Lübbecke (2008) [70] proposed integer programming techniques, though scalability remained an issue due to the problem's NP-hard nature. More recently, machine learning-enhanced heuristics and hyper-heuristics have gained traction. Pillay (2014) [60] provides a survey of hyper-heuristics for educational timetabling, including their application to PE-CTT, emphasizing the adaptability of such methods to changing instance features.

For this version of the CTT problem, many papers were published during and after the competitions. The interest in this problem remains strong, with recent publications proposing various methodologies. Recent papers focusing on local search methods are the papers by Goh et al. (2019) [71], (2020) [72] and Nagata et al. (2018) [73]. Metaheuristics like Simulated Annealing also effectively tackled the problem by Ceschia et al. (2012) [74]. Cambazard et al. (2012) [75] presented a Constraint Programming approach. Lewis et al. (2015) [76] explored the connectivity of the solution space in course timetabling problems under various neighborhood operators. Valouxis et al. (2012) [77] proposed a decomposition method for High school time-tabling.

Despite substantial progress, challenges remain in handling real-world constraints, dynamic enrollment changes, and fairness metrics—areas increasingly addressed in recent studies Qu et al., 2020 [78].

4.3 Datasets

Details about problem instances belonging to datasets ITC2002 and ITC2007 are presented in Table 4.1 and Table 4.2 respectively. The former table has three fewer columns than the latter since problem instances of ITC2002 have neither event-

precedence relations nor event-time-slot restrictions.

Table 4.1: PE-CTT: Dataset ITC2002

Instance Name	Events	Rooms	Features	Students	Conflict Density	Average Room Capacity	Average Room Suitability
o01.tim	400	10	10	200	0.20	10.40	1.96
o02.tim	400	10	10	200	0.21	10.40	1.92
o03.tim	400	10	10	200	0.23	10.80	3.42
o04.tim	400	10	5	300	0.23	15.30	2.45
o05.tim	350	10	10	300	0.31	17.30	1.78
o06.tim	350	10	5	300	0.26	17.90	3.59
o07.tim	350	10	5	350	0.21	20.60	2.87
o08.tim	400	10	5	250	0.17	12.80	2.93
o09.tim	440	11	6	220	0.17	10.36	2.58
o10.tim	400	10	5	200	0.20	10.70	3.49
o11.tim	400	10	6	220	0.20	11.40	2.06
o12.tim	400	10	5	200	0.20	10.30	1.96
o13.tim	400	10	6	250	0.21	12.60	2.43
o14.tim	350	10	5	350	0.25	20.30	3.08
o15.tim	350	10	10	300	0.25	17.40	2.19
o16.tim	440	11	6	220	0.18	10.73	3.17
o17.tim	350	10	10	300	0.31	17.20	1.11
o18.tim	400	10	10	200	0.21	10.50	1.75
o19.tim	400	10	5	300	0.20	15.30	3.94
o20.tim	350	10	5	300	0.25	17.50	3.43

Table 4.2: PE-CTT: Dataset ITC2007

Instance Name	Events	Rooms	Features	Students	Conflict Density	Average Period Unavailability	Average Room Capacity	Average Room Suitability
i01.tim	400	10	10	500	0.34	0.44	37.70	4.08
i02.tim	400	10	10	500	0.37	0.43	36.10	3.95
i03.tim	200	20	10	1000	0.47	0.43	86.60	5.04
i04.tim	200	20	10	1000	0.52	0.43	89.15	6.40
i05.tim	400	20	20	300	0.31	0.43	21.55	6.80
i06.tim	400	20	20	300	0.30	0.44	21.80	5.07
i07.tim	200	20	20	500	0.53	0.60	42.00	1.57
i08.tim	200	20	20	500	0.51	0.62	44.50	1.92
i09.tim	400	10	20	500	0.34	0.44	37.90	2.91
i10.tim	400	10	20	500	0.38	0.43	36.30	3.20
i11.tim	200	10	10	1000	0.50	0.44	84.10	3.38
i12.tim	200	10	10	1000	0.58	0.43	84.10	3.35
i13.tim	400	20	10	300	0.32	0.43	22.10	8.68
i14.tim	400	20	10	300	0.32	0.43	22.25	7.56
i15.tim	200	10	20	500	0.53	0.61	44.00	2.23
i16.tim	200	10	20	500	0.45	0.61	43.90	1.74
i17.tim	100	10	10	500	0.70	0.43	138.20	2.77
i18.tim	200	10	10	500	0.65	0.43	70.40	3.48
i19.tim	300	10	10	1000	0.47	0.44	56.30	3.66
i20.tim	400	10	10	1000	0.28	0.44	44.80	3.73
i21.tim	500	20	20	300	0.23	0.42	17.40	7.36
i22.tim	600	20	20	500	0.26	0.43	24.85	5.65
i23.tim	400	20	30	1000	0.44	0.21	68.65	2.89
i24.tim	400	20	30	1000	0.31	0.44	42.65	1.59

4.4 Symmetries and Preprocessing

The format of the problem instances assumes rooms with varying capacities alongside features that each room might have (e.g., video projector, smart-board, laboratory equipment, etc.). Additionally, it lists the events that each student participates and any additional obligations associated with those events. In the ITC2007 dataset, events can additionally have time-slot restrictions (i.e., a time-slot might be prohibited for certain events) and precedence relations (i.e., an event may be required to take place earlier than another event). The possibility of tracking all the feasible combinations consisting of three events in order to reduce the model size is investigated.

4.4.1 Event-Room eligibility

Let $\mathbb{E}$ by the set of all events. Equation 4.1 determines when a student attends an event and the total number of attendees of each event is given by Equation 4.2. This information is extracted from the problem data.

$$a_{se} = \begin{cases} 1 & \text{if student } s \text{ attends event } e \\ 0 & \text{otherwise} \end{cases} \qquad \forall s \in \mathbb{S}, \quad \forall e \in \mathbb{E} \tag{4.1}$$

$$S_e = \sum_{s \in \mathbb{S}} a_{se} \quad \forall e \in \mathbb{E} \tag{4.2}$$

4.4.2 Event Conflicts

Events with common students are prohibited from taking place at the same time-slot because, by definition, no student can attend more than one event at once. Such pairs of events are considered conflicting events. Let $\mathbb{R}$ be the set of all rooms, and $\mathbb{R}_e$ be the set of rooms that can host event e. The number of conflicting events is increased by adding pair of events without common students if for two events e_1 and e_2 the relations $\mathbb{R}_{e_1} \equiv \mathbb{R}_{e_2}$ and $|\mathbb{R}_{e_1}| = |\mathbb{R}_{e_2}| = 1$ hold true (singleton sets). This condition means that e_1 and e_2 can be hosted only in the same room and, therefore, can not be hosted in the same time-slot. Finally, the pairs of all conflicting events form set $\mathbb{G}$. Based on the event conflicts, the conflict density of each problem is twice the size of set $\mathbb{C}$ divided by the square of the size of set E. Conflict density can be considered a measure of each problem instance's difficulty, but room existence seems to distort its relevance.

4.4.3 Event Combinations

In the preprocessing stage, all combinations of three events are computed and the number of students participating in each combination is stored. More formally, let $\mathbb{S}$ be the set of students, and $\mathbb{E}_s$ be the set of events that student s attends. For every student s, all possible combinations consisting of three events e_1, e_2, e_3, where $e_1 \in \mathbb{E}_s, e_2 \in \mathbb{E}_s, e3 \in \mathbb{E}_s$ are generated. Finally, $\mathbb{C}$ is the set comprised of all previously generated three event combinations.

4.5 Formulation

4.5.1 Mathematical Model

In this section the base mathematical model of the problem is presented.

Let $\mathbb{S}$ be the set of all students.

Let S_e be the total number of students attending event e.

Let $\mathbb{R}$ be the set of all rooms.

Let R_e be the set of rooms that can not host event e.

Let $\mathbb{T}$ be the set of all time-slots.

Let T_e be the set of time-slot that event e cannot be scheduled.

Let $\mathbb{L} = [9, 18, 27, 36, 45]$. These numbers refer to the last time-slot of each one of the 5 days.

Let $\mathbb{G}$ be the set of conflicting event pairs.

Let $\mathbb{C}$ be the set consisting of combinations of three events with students in common.

Let C_{e_1,e_2,e_3} be the total number of students attending all three events e_1, e_2 and e_3.

Let $\mathbb{P}$ be the set of pairs of events having a precedence relation.

The binary decision variables x_{etr} are defined in equation 4.3. Binary variables y_{sd} and $z_{e_1e_2e_3}$ in equations 4.4 and 4.5 are auxiliary variables.

$$x_{etr} = \begin{cases} 1 & \text{if event } e \text{ is scheduled in time-slot } t \text{ at room } r \\ 0 & \text{otherwise} \end{cases} \qquad \forall e \in \mathbb{E}, \forall t \in \mathbb{T}, \forall r \in \mathbb{R} \tag{4.3}$$

$$y_{sd} = \begin{cases} 1 & \text{if student } s \text{ has a single event in day } d \\ 0 & \text{otherwise} \end{cases} \quad \forall s \in \mathbb{S}, \forall d \in [1..5] \tag{4.4}$$

$$z_{e_1 e_2 e_3} = \begin{cases} 1 & \text{if events } e_1,\ e_2 \text{ and } e_3 \text{ are placed in 3 sequential time-slots in the same day} \\ 0 & \text{otherwise} \end{cases}$$

$$\forall (e_1, e_2, e_3) \in \mathbb{C} \tag{4.5}$$

$$\text{Minimize} \sum_{e \in \mathbb{E}} \sum_{t \in \mathbb{L}} \sum_{r \in \mathbb{R}} S_e * x_{etr} + \sum_{s \in \mathbb{S}} \sum_{d=1}^{5} y_{sd} + \sum_{(e_1,e_2,e_3) \in \mathbb{C}} C_{e_1,e_2,e_3} z_{e_1,e_2,e_3} \tag{4.6}$$

Subject to

$$\sum_{t \in \mathbb{T}_e} \sum_{r \in \mathbb{R}} x_{etr} = 0 \quad \forall e \in \mathbb{E} \tag{4.7}$$

$$\sum_{t \in \mathbb{T}} \sum_{r \in \mathbb{R}_e} x_{etr} = 0 \quad \forall e \in \mathbb{E} \tag{4.8}$$

$$\sum_{t \in \mathbb{T}} \sum_{r \in \mathbb{R}} x_{etr} = 1 \quad \forall e \in \mathbb{E} \tag{4.9}$$

$$\sum_{e \in \mathbb{E}, r \in \mathbb{R}} x_{etr} \leq 1 \quad \forall r \in \mathbb{R}, \quad \forall t \in \mathbb{T} \tag{4.10}$$

$$\sum_{r \in \mathbb{R}} x_{e_1 tr} + \sum_{r \in \mathbb{R}} x_{e_2 tr} \leq 1 \quad \forall (e_1, e_2) \in \mathbb{G}, \quad \forall t \in \mathbb{T} \tag{4.11}$$

$$\sum_{t \in \mathbb{T}} \sum_{r \in \mathbb{R}} t * x_{e_1 tr} + 1 \leq \sum_{t \in \mathbb{T}} \sum_{r \in \mathbb{R}} t * x_{e_2 tr} \quad \forall (e_1, e_2) \in \mathbb{P} \tag{4.12}$$

$$\sum_{t=t'}^{t'+2} \sum_{r \in \mathbb{R}} x_{e_1 tr} + x_{e_2 tr} + x_{e_3 tr} \leq 2 + z_{e_1 e_2 e_3} \quad \forall (e_1, e_2, e_3) \in \mathbb{C}, \quad \forall t' \in T, \quad t' \notin [8, 9, 17, 18, 26, 27, 35, 36, 44, 45] \tag{4.13}$$

$$y_{sd} = 1, \quad \text{if} \sum_{e \in \mathbb{E}_s} \sum_{t=1+(d-1)*9}^{d*9} \sum_{r \in \mathbb{R}} x_{etr} = 1 \quad \forall s \in \mathbb{S}, \quad \forall d \in [1..5] \tag{4.14}$$

Equation 4.6 is the objective function that incorporates the costs associated with the

three soft constraints. The first term imposes penalty S_e for any event e scheduled in the day's final time slot. The second term imposes a single penalty point for each student who participates in only one event during a day. The last term imposes a penalty equal to the total number of students attending a combination of three events if these events are in three adjacent time-slots on the same day.

Constraints 4.7, 4.8 handle time-slot and room availability limitations, while constraint 4.9 ensures that each event is scheduled once and only once. Constraint 4.10 ensures that at most one event is scheduled in each room in each time-slot. Conflicting events are banned from the same time-slot through the use of constraint 4.11, and precedence relations are respected as a consequence of constraint 4.12. To enforce a penalty for three consecutive events constraint 4.13 will activate z decision variables. Finally, since constraint 4.14 is nonlinear, it can be handled by CP solvers and MIP solvers equipped with automatic linearization capabilities.

4.5.2 Model Modifications

The base mathematical model may not be able to handle large instances, mainly due to the large size of $\mathbb{C}$. However, a decomposition of the problem that operates over 2 or 3 days can produce partial solutions that hopefully will drive a Mathheuristic approach to promising subdomains of the search space.

The three modifications to the base model follows.

- **Improve day by day** Optimizing each day in isolation results in some advantages. Since the number of events that are scheduled in one day is fewer than all events, the number of combinations of three events becomes significantly smaller. Moreover, the second term of the objective function is redundant now because the events of the day are determined. Finally, since the involved students participate in a number of events that cannot be changed, constraint 4.14 becomes redundant.

- **Improve days** In this modification of the base model, two or three days are considered. Again, as in the previous modification, the three event combinations are fewer than combinations involving all events. In this setting, students can be considered to attend a subset of the events they actually attend since some events have not been scheduled on the days considered. The advantage is that now more, temporarily identical students can be identified and consolidated in the second term of the objective function and constraint 4.14.

- **Fix room** Starting from a given solution, all events may not to change rooms but they can change time-slots. Practically, events placed in the same room at

different time-slots are allowed to swap places.

4.5.3 Neighborhood operators

Three different neighborhood operators are used in this work:

- **Transfer Event**: An event $e \in \mathbb{E}$ is moved from its currently designated time-slot $t \in \mathbb{T}$, to a new, randomly selected time-slot $t_1 \in \mathbb{T}$. The move is executed only if t_1 is available for e, a compatible free room $r \in R$ exists in t_1 for event e and the precedence relations for event e are not violated .
- **Swap Events**: Two time-slots $t_1, t_2 \in \mathbb{T}$ that are designated to two events $e1, e2 \in \mathbb{E}$ are swapped. The move is executed only if $e1$ and $e2$ are in conflict, a suitable room $r_1 \in \mathbb{R}$ for $e1$ is available at t_2 and a suitable room $r_2 \in \mathbb{R}$ is available at t_1 and is all the precedence relations for e_1 and e_2 are not violated.
- **Kempe Chain**: An event e residing in time-slot $t_1 \in \mathbb{T}$ is selected randomly and moved to time-slot $t_2 \in \mathbb{T}$. All events in t_2 conflicting with e are moved to t_1. An ejection procedure follows until no conflicting events co-exist in t_1 or t_2. The move is executed if:
  - At each step for an event $e_s \in \mathbb{E}$ a compatible room exists in the selected time-slot.
  - All the precedence relations of e are not violated.
  - The selected period is available for the event e.

4.5.4 Simulated Annealing (SA)

Many versions of Simulated Annealing have been proposed in the literature, see Dowsland et al. (2012) [79]. The version used is based on the classic one proposed by Kirkpatrick et al. (1983) [80]. In detail, at each iteration, a neighborhood operator 4.5.3 is randomly selected. The move is performed if the objective value is reduced. If $D_f > 0$, where D_f is the difference between the cost of the current iteration and the cost of the previous iteration, then the candidate solution has the potential of being accepted. The acceptance depends on the probability defined in equation 4.15 where T is the current temperature value and T_s is the temperature value used for initiating the procedure. A geometric cooling scheme is employed $T = a * T$, where $a \in [0.9, 0.999]$ is the cooling rate. Parameters are selected by parameter tuning. Also, a freezing temperature $F_t = 1$ is used in the procedure. When T reaches F_t, two or three random days are selected, and an improve_days model is solved to reduce the objective value. When the previous step ends, the temperature is set to a random

value in the range $[0.5 * T_s, 1.5 * T_s]$. The procedure terminates when the time limit expires.

$$P = e^{-D_f/T} \tag{4.15}$$

4.6 Results

The experiments were programmed in Python, and used Gurobi MIP solver [81] and Google OR-Tools CP-SAT solver. The Gurobi MIP solver constructs the initial solution using the base model. Then, the solution improves by employing the day-by-day modification of the base model. The Simulated Annealing procedure follows as described in section 4.5.4. In each iteration of the Simulated Annealing procedure, a neighborhood operator is selected randomly either from the operators described in subsection 4.5.3, or one of the model modifications described in subsection 4.5.2. The implementations of the model modifications were programmed using CP-SAT solver. A generous time duration was provided for each execution (all steps) which amounted to approximately two hours for each problem instance. The experiments ran in a workstation equipped with an AMD Ryzen 5700G(8C/16T) processor and 32GB of RAM, running Windows 11. Results are presented in table 4.3 for both ITC2002 ITC2007 datasets. This approach manages to find solutions for all instances.

Course time-tabling is a problem with a special relation to Academia it's easy to understand and NP-Hard to solve. In a relatively short time modern exact solvers can handle real life sized instances. This work produced the following conference publications:
A. Dimitsas, V. Nastos, C. Valouxis, and C. Gogos, "A mathematical formulation for constructing feasible solutions for the Post Enrollment Course Timetabling Problem," in *2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE, 2022, pp. 1–5
A. Dimitsas, V. Nastos, C. Gogos, and C. Valouxis, "An exact based approach for the Post Enrollment Course Timetabling Problem," in *Proceedings of the 26th Pan-Hellenic Conference on Informatics*. Athens Greece: ACM, Nov. 2022, pp. 77–82

Table 4.3: ITC_2002 and ITC_2007 results

Instance ITC 2002	Best solution	This work	Instance ITC 2007	Best solution	This work
o01.tim	N/A	454	i01.tim	0	0
o02.tim	14	19	i02.tim	0	0
o03.tim	36	38	i03.tim	31	193
o04.tim	76	77	i04.tim	21	92
o05.tim	56	65	i05.tim	0	35
o06.tim	1	8	i06.tim	0	0
o07.tim	2	18	i07.tim	0	0
o08.tim	6	14	i08.tim	0	39
o09.tim	8	12	i09.tim	0	31
o10.tim	41	50	i10.tim	0	31
o11.tim	19	27	i11.tim	39	76
o12.tim	N/A	460	i12.tim	0	0
o13.tim	51	58	i13.tim	0	0
o14.tim	13	28	i14.tim	0	0
o15.tim	3	17	i15.tim	0	10
o16.tim	4	13	i16.tim	0	34
o17.tim	35	41	i17.tim	0	89
o18.tim	11	20	i18.tim	0	56
o19.tim	46	48	i19.tim	0	50
o20.tim	0	20	i20.tim	543	574
			i21.tim	5	6
			i22.tim	5	19
			i23.tim	1292	1335
			i24.tim	0	12

Chapter 5

Case Study: Thesis Defense Timetabling Problem

The last case study from the world of Educational Timetabling is about Thesis Defense Timetabling. In this chapter a Quadratic Programming problem is examined with MIQP and CP. The symmetry elimination leads to near optimal results.

5.1 Problem Description

The Thesis Defense Timetabling Problem (TDTP) is a specialized variant of academic timetabling, sharing similarities with course timetabling, examination scheduling, and meeting scheduling problems. It involves assigning students, advisors, and committee members to defense slots while satisfying a range of constraints such as availability, room capacity, and time conflicts.

Three entities exist in this specific version, introduced by Battistutta et al. (2019) [2], of the thesis defense timetabling problem: candidates, faculty members, and sessions. The effort that should be undertaken would result in allocating candidates and faculty members to sessions.

Candidates are the students who will be defending their theses once. Each candidate has a single faculty member as his supervisor and may have suitable faculty members to serve as opponents.

Faculty members can participate in numerous sessions and range in academic levels. Faculty members may only be able to attend some sessions.

Sessions may overlap, and obviously faculty members can not be present in concurrent sessions.

The constraints of the problem are either hard constraints or soft constraints. A hard constraint violation makes a schedule infeasible, while soft constraints penalize the objective if unsatisfied. The goal is to satisfy all hard constraints while minimizing soft constraint violations.

The hard constraints are:

- **H1. Supervision:** The supervisor of a candidate must be present in the session.
- **H2. Students Per Session:** The number of candidates in the same session must be, at most, a given maximum.
- **H3. Overlapping Sessions:** A faculty member cannot attend sessions that overlap.

- **H4. Availability:** Each faculty member must be available in the sessions he attends.
- **H5. Committee Composition:** Sessions committees must respect a minimum and maximum number of faculty staff possessing a specific academic level. Staff with a higher academic level can replace staff with a lower level but not vice versa.

The soft constraints are:

- **S1. Multiple Duties:** If a is the number of sessions a faculty member attends, then penalty $(a-1)^2$ is applied if $a > 0$.
- **S2. Opponent Presence:** A penalty occurs if a candidate attends a session without any of his suitable opponents present. If the candidate has no suitable opponents, no penalty is applied.

The objective function comprises of the sum of soft constraint violations multiplied by a weight. The weights in the original proposition were $w_{S1} = 1$ and $w_{S2} = 3$ for each soft constraint violation, respectively.

5.2 Related Work

The TDTP problem is less studied when compared to other educational timetabling problems like examination timetabling and course timetabling. Differences in regulations among universities and the rather limited number of students usually affected by this problem may play a part in this. However, researchers have proposed different problem formulations along with different solution approaches. A university specific formulation was proposed by Limanto et al. (2018) [84] who then realized a solution with genetic algorithms. A multi-objective variant of the problem was introduced by Huynh et al. (2012) [85], and solved using a mixed integer linear programming model. Another variant of the problem alongside with a carefully constructed dataset of real and artificial problem instances was proposed by Battistutta et al. (2019) [2]. Their approaches for solving the problem are threefold, metaheuristics, constraint programming and mixed integer programming which performs in this work best for most cases.

Various approaches have been applied to the TDTP problem, as usually occurs with difficult combinatorial optimization problems. So, approaches like hybrid algorithms in Su et al. (2020) [86], metaheuristics Tawakkal et al. (2020) [87], and mixed integer programming Almeida (2022) [88] should be able to find good solutions to TDTP, provided that careful implementations are used. Recent studies have tackled

thesis defense scheduling as a multi-constraint, multi-objective optimization problem.

TDTP has some conceptual similarities with the Conference Scheduling problem see Stidsen et al. (2018) [89], since in both cases committees are involved. Nevertheless, in Conference Scheduling the objective is to create an optimal schedule that maximizes attendee satisfaction and minimizes conflicts between sessions of interest while at the same time ensures a balanced and diverse program.

Despite these advances, TDTP remains relatively underexplored compared to broader timetabling problems. Its unique blend of constraints (e.g., synchronization of multiple faculty members for a single event) and the dynamic nature of academic calendars call for tailored algorithms that emphasize feasibility, user satisfaction, and computational efficiency.

5.3 Dataset

Here the public dataset from Battistutta et al. (2019) [2] is briefly described. This dataset consists of 21 real instances taken from Italian Universities and 45 additional instances artificially generated that share similar characteristics with the real instances. Table 5.1 presents the main characteristics of problem instances. Since the dataset size is relatively large, only the base features of each problem instance are presented (i.e., number of candidates, number of sessions, number of faculty members, maximum capacity of all sessions). Further features for each problem instance, like the number of simultaneous sessions, and others, can be consulted at `https://opthub.uniud.it/`.

Table 5.1: TDTP: Descriptive Statistics for the Thesis Defense dataset

Instance	Candidates	Faculty Members	Max Candidates	Sessions	Instance	Candidates	Faculty Members	Max Candidates	Sessions
real01	143	155	9	16	art13	110	52	10	13
real02	34	79	10	5	art14	459	504	9	55
real03	144	77	8	18	art15	522	290	10	56
real04	171	101	10	18	art16	532	461	10	60
real05	442	296	12	37	art17	532	796	10	60
real06	79	99	8	10	art18	249	413	10	28
real07	102	66	13	8	art19	183	274	10	19
real08	55	67	10	6	art20	555	671	10	57
real09	173	280	14	15	art21	457	724	10	51
real10	209	435	7	31	art22	276	405	8	37
real11	164	287	12	16	art23	269	127	9	31
real12	222	322	14	16	art24	42	62	10	5
real13	430	435	13	35	art25	71	37	8	9
real14	551	454	13	45	art26	169	187	8	23
real15	428	454	7	66	art27	576	277	10	59
real16	124	84	10	13	art28	326	441	10	34
real17	18	76	10	2	art29	366	196	8	48
real18	172	80	11	18	art30	361	617	10	42
real19	69	79	11	7	art31	263	351	11	26
real20	146	81	11	14	art32	73	192	10	8
real21	58	31	15	4	art33	134	240	10	14
art01	307	251	8	41	art34	52	172	10	6
art02	40	30	8	6	art35	117	205	10	12
art03	208	240	9	24	art36	112	204	10	12
art04	347	409	9	43	art37	90	207	10	10
art05	137	261	8	18	art38	176	296	11	18
art06	434	267	10	45	art39	282	360	11	28
art07	467	261	9	54	art40	243	339	11	24
art08	427	705	9	54	art41	104	207	10	11
art09	73	64	10	8	art42	164	290	11	17
art10	40	39	8	6	art43	141	269	11	15
art11	395	209	9	49	art44	89	227	10	10
art12	255	420	9	32	art45	225	330	11	23

5.4 Symmetries and descriptive analytics

Instances from the dataset are characterized by a great amount of symmetry. Some symmetries were identified by a descriptive analysis of different entities in the problem. A systematic search is conducted upon entities like Candidates, Opponents, Faculty Members and Sessions to identify which have the same characteristics and the frequency of these similarities.

5.4.1 Candidate symmetry

A candidate c is identified by his supervisor F_c and the set of suitable opponents $\mathbb{Q}_c$. All candidates $c_1, c_2, ..., c_n$ where $F_{c_1} = F_{c_2} =, ..., = F_{c_n}$ and $\mathbb{Q}_{c_1} \equiv \mathbb{Q}_{c_2} \equiv, ..., \equiv \mathbb{Q}_{c_n}$ are identical. They can be grouped together, the sessions they can be placed are the same sessions as their supervisor F_c is the same his session requirements will apply for all of them. The order of all candidates in a candidate group in the final schedule will be irrelevant as they have exactly the same impact on the objective function.

5.4.2 Opponent symmetry

After grouping candidates based on their supervisor and suitable opponents we can group them once again based on their suitable opponents disregarding their supervisor. This can serve to reduce the amount of decision variables and constraints needed to enforce soft constraint **S2** regarding opponent presence. The rationale behind this is that candidate groups with the same suitable opponents seek the same opponents regardless of their supervisor.

5.4.3 Faculty members only useful for their Academic Level

Each faculty member belongs to one of the three categories:

- **Supervisors** A faculty member that supervises at least one candidate.
- **Suitable opponent** A faculty member that is not a supervisor but has at least one candidate he can suitably oppose.
- **Only for academic level** A faculty member that is neither a supervisor nor can he oppose any candidate, his only role in the schedule is to cover the hard constraint **H5** regarding Committee Composition.

A special mention is needed for the last category. Faculty members used only for their academic level can impose a great amount of symmetry while offering little in the objective function. They cannot be omitted entirely but can be ignored to calculate some lower bounds as described in section 5.5.

5.4.4 Session symmetry

A session s is defined by the faculty members that can participate. A groups of sessions $s_1, s_2, ..., s_n$ where the exactly same set of faculty members can participate is identical. Overlapping sessions in the instances are always identical as faculty members not available in a session are also not be available in an overlapping session.

5.4.5 Descriptive Analytics

Table 5.2 contains statistics regarding section 5.4. All instances contain groups of candidates and opponents. The amount of faculty staff useful only for academic level varies widely between instances from 0 to the majority of all faculty staff falling under this category.

Table 5.2: TDTP: Symmetry statistics

Instance	Candidate groups	Opponent groups	Faculty only for Academic Level	Instance	Candidate groups	Opponent groups	Faculty only for Academic Level
real01	103	41	81	art13	86	82	0
real02	31	28	30	art14	428	417	30
real03	115	110	3	art15	441	408	2
real04	140	136	19	art16	467	446	15
real05	174	5	125	art17	492	474	73
real06	67	57	28	art18	236	233	70
real07	87	27	28	art19	175	170	37
real08	41	15	35	art20	494	484	44
real09	104	70	178	art21	422	411	84
real10	174	133	247	art22	257	249	33
real11	108	76	168	art23	211	202	0
real12	146	101	180	art24	41	40	7
real13	166	3	270	art25	54	51	0
real14	195	8	263	art26	142	141	14
real15	328	198	192	art27	493	467	1
real16	104	100	7	art28	292	284	37
real17	17	17	41	art29	311	289	0
real18	149	145	2	art30	338	329	90
real19	61	61	9	art31	185	142	170
real20	130	128	3	art32	60	60	82
real21	25	7	7	art33	115	115	82
art01	281	266	5	art34	46	46	90
art02	36	34	1	art35	100	99	55
art03	190	184	15	art36	97	95	71
art04	318	304	26	art37	65	61	112
art05	127	124	41	art38	123	108	161
art06	367	351	1	art39	196	160	146
art07	395	370	0	art40	157	135	161
art08	392	381	111	art41	80	66	108
art09	60	58	2	art42	114	90	168
art10	33	32	4	art43	97	89	143
art11	338	317	0	art44	64	55	141
art12	237	230	58	art45	142	119	169

5.4.6 Identical Sessions

Table 5.3 contains all instances and the respective sets of identical sessions as defined in 5.4. Note that overlapping sessions are not always identical. A simple explanation is that these sessions do overlap but do not necessarily start and end simultaneously, so a faculty member may be able to attend some of them and not others.

Table 5.3: TDTP: Symmetrical Sessions

Instance	Identical Sessions
real04	{13, 14}, {17, 18}
real05	{1, 2}, {3, 4}, {5, 6}, {8, 9}, {10, 11}, {12, 13}, {14, 15}, {16, 17}, {18, 19}, {20, 21}, {22, 23}, {24, 25}, {26, 27}, {28, 29}, {30, 31}, {32, 33}, {34, 35}, {36, 37}
real08	{2, 3}
real09	{1, 2, 3, 4, 5, 6, 7, 8}, {9, 10, 11, 12, 13, 14, 15}
real11	{1, 2, 3, 4, 5, 6, 7, 8}, {9, 10, 11, 12, 13, 14, 15, 16}
real12	{1, 2, 3, 4, 5, 6, 7, 8}, {9, 10, 11, 12, 13, 14, 15, 16}
real15	{8, 7}, {20, 21}, {46, 47}, {49, 50}, {51, 52}, {53, 54}
real18	{3, 4}, {13, 14}
real20	{1, 2}, {11, 12}
real21	{2, 3}
art31	{1, 2}, {4, 21}, {8, 11}, {10, 18}, {12, 20}, {13, 22}, {26, 19}
art32	{4, 7}
art33	{2, 11}, {4, 7}, {8, 5}
art34	{2, 6}
art35	{1, 10}, {9, 3}
art36	{2, 12}, {5, 6}
art37	{1, 3}, {8, 2}
art38	{1, 13}, {12, 5}, {8, 6}, {17, 11}
art39	{1, 11}, {26, 4}, {16, 7}, {8, 24}, {28, 13}, {25, 15}, {17, 22}, {27, 19}
art40	{1, 5}, {2, 4}, {18, 6}, {8, 21}, {10, 11}, {22, 15}
art41	{5, 6}, {9, 7}
art42	{1, 5}, {2, 12}, {3, 4}, {16, 14}
art43	{1, 11}, {2, 14}, {8, 12}
art44	{2, 5}, {3, 7}
art45	{1, 18}, {9, 3}, {13, 6}, {8, 22}, {12, 21}, {20, 23}

5.5 Formulation

This section describes the formulation of the model. Table 5.4 presents the notation used hereafter for sets, parameters and constants.

Table 5.4: TDTP: Notation

Sets	
$\mathbb{F}$	Set of faculty members.
$\mathbb{G}$	Set of faculty members useful only for their academic level.
$\mathbb{S}$	Set of sessions.
$\mathbb{O}$	Set of pairs of overlapping sessions.
$\mathbb{E}$	Set of pairs of symmetrical sessions.
$\mathbb{S}_f$	Set of sessions a faculty member f is unavailable.
$\mathbb{Q}_o$	Set of sets of suitable opponents.
$\mathbb{L} = \{1, ..., L\}$	Set of academic levels.
Parameters and constants	
C	Max number of candidates a session can host.
C_{fq}	Number of candidates whose set of suitable opponents is q supervised by faculty member f.
D	$min(C, \sum_{f \in \mathbb{F}} C_{fq})$
S	Total number of sessions.
L	Maximum academic level.
m_l	Minimum number of faculty members having at least academic level l.
M_l	Maximum number of faculty members having at least academic level l.
B_f	A numeric identifier assigned for each faculty member assuming consecutive values starting at 1.
A_f	Academic level for faculty member f.

5.5.1 Base model

A Constraint Programming formulation is described below. The two primary decision variables are:

$$x_{fqs} \in [0, min(C, C_{fq})] \quad \forall f \in \mathbb{F}, \quad \forall q \in \mathbb{Q}, \quad \forall s \in \mathbb{S} \tag{5.1}$$

$$y_{fs} = \begin{cases} 1, & \text{if faculty member } f \text{ is scheduled to be in the committee for session } s. \\ 0, & \text{else.} \end{cases}$$

$$\forall f \in \mathbb{F}, \forall s \in \mathbb{S} \tag{5.2}$$

Then, the following auxiliary decision variables that are used in the objective function are defined.

$$z_{qs} \in [0, D] \quad \forall q \in \mathbb{Q}, \quad |q| > 0, \quad \forall s \in \mathbb{S} \tag{5.3}$$

$$v_f \in [0, S - 1] \quad \forall f \in \mathbb{F} \tag{5.4}$$

$$u_f \in [0, (S-1)^2] \quad \forall f \in \mathbb{F} \tag{5.5}$$

The objective function:

$$\min \quad w_{s1} * \sum_{f \in \mathbb{F}} u_f + w_{s2} * \sum_{q \in \mathbb{Q}, |q|>0} \sum_{s \in \mathbb{S}} z_{qs} \tag{5.6}$$

Subject to:

$$\sum_{s \in \mathbb{S}} x_{fqs} = C_{fq} \quad \forall f \in \mathbb{F} \quad \forall q \in \mathbb{Q} \tag{5.7}$$

$$min(C, C_{fq}) * y_{fs} >= \sum_{q \in \mathbb{Q}} x_{fqs} \quad \forall f \in \mathbb{F} \quad \forall s \in \mathbb{S} \tag{5.8}$$

$$\sum_{f \in \mathbb{F}} \sum_{q \in \mathbb{Q}} x_{fqs} \leq C \quad \forall s \in \mathbb{S} \tag{5.9}$$

$$y_{fs_1} + y_{fs_2} \leq 1 \quad \forall s_1, s_2 \in \mathbb{O}, \quad \forall f \in \mathbb{F} \tag{5.10}$$

$$y_{fs} = 0 \quad \forall f \in \mathbb{F}, \forall s \in \mathbb{S}_f \tag{5.11}$$

$$m_l \leq \sum_{f \in \mathbb{F}, A_f \leq l} y_{fs} \leq M_l \quad \forall l \in \mathbb{L}, \forall s \in \mathbb{S} \tag{5.12}$$

$$z_{qs} \geq \sum_{f \in \mathbb{F}} x_{fqs} - D * \sum_{f \in q} y_{fs} \quad \forall q \in \mathbb{Q}, \quad |q| > 0, \quad \forall s \in \mathbb{S} \tag{5.13}$$

$$\sum_{s \in \mathbb{S}} (y_{fs}) - 1 \leq v_f \quad \forall f \in \mathbb{F} \tag{5.14}$$

$$u_f = v_f * v_f \quad \forall f \in \mathbb{F} \tag{5.15}$$

$$\sum_{f \in \mathbb{F}, A_f \leq 1} B_f * y_{fs_1} \leq \sum_{f \in \mathbb{F}, A_f \leq 1} B_f * y_{fs_2} \quad \forall s_1, s_2 \in \mathbb{E} \tag{5.16}$$

$$v_{f_1} \geq v_{f_2} \quad \forall f_1, f_2 \in \mathbb{F}, \quad A_{f_1} = A_{f_2}, \quad \mathbb{S}_{f_1} \equiv \mathbb{S}_{f_2}, \quad f_1 \notin \mathbb{G}, \quad f_2 \in \mathbb{G} \tag{5.17}$$

As already mentioned, the two sets of primary decision variables are x_{fqs} and y_{fs}. The first one assumes integer values in order to eliminate the candidate symmetry. This occurs because equation 5.1 represents how many candidates supervised by faculty member f whose set of suitable opponents is q are scheduled in session s, which means that candidates with the same supervisor and the same suitable opponents group can swap sessions without affecting the solutions quality. The second set of primary decision variables, y_{fs} are binary and they are defined in equation 5.2. They assume value 1 if faculty member f participates in the committee of session s, or 0 otherwise.

Auxiliary decision variables z_{qs}, defined in equation 5.3, keep count of candidates having set q as their opponent group are left without a suitable opponent in session s. For the quadratic part of the problem's definition variables v_f in equation 5.4 are introduced, whose role is to count the excess sessions a faculty member f has to attend over one session. Finally, decision variables u_f are defined in equation 5.5 assuming values of the corresponding variables v_f raised to the power of 2.

Constraint 5.7 ensures that all candidates with q as their opponent group and faculty member f as their supervisor will be scheduled. Constraint 5.8 states that each supervisor will be present in all sessions that his candidates attend. Constraint 5.9 limits the number of candidates in each session according to the maximum allowed capacity. Constraint 5.10 ensures that no faculty member is scheduled in overlapping sessions. Constraint 5.11 prohibits faculty members to appear in sessions when they are unavailable. The committee requirements of hard constraint **H5** (committee composition) are respected by effect of constraint 5.12. Constraint 5.13 ensures that the decision variable z_{qs} will be equal to the number of candidates with q as their opponents group provided that these candidates have not been placed with at least one suitable opponent. Constraints 5.14 and 5.15 set the penalty of the quadratic part of the objective. Note that in the latter constraint decision variables are raised to the power of 2 which would require a quadratic solver if mathematical programming was used. Since the solver of choice is a constraint programming one capable of handling multiplication of decision variables, this constraint is directly included in the model.

The last two constraints serve for symmetry reduction. In section 5.4 four different kinds of symmetry are presented. The integer primary variables 5.1 and 5.3 to eliminate candidate and opponent symmetries respectively. To reduce the effect of session symmetry constraint 5.16 is enforced. Using a standard lexicographical

technique, symmetrical sessions are forced to be lexicographical ordered (by identifier) by faculty members of academic level 1 assigned to them. To limit the effect of faculty members used only for their academic level, the final constraint 5.17 ensures that these faculty members will be scheduled in the resulting timetable only if no other faculty member can replace them without imposing a penalty. The rationale is that any other faculty member of the same academic level, having the same unavailabilities, has merit to be used instead since he/she can also serve as an opponent.

5.5.2 Zero cost solutions

To search for solutions without a cost, some facts about these solutions can be considered, as to reduce variables and constraints. The only variable needed is y_{fs}, defined in equation 5.2, which denotes the session a faculty member may be scheduled to. From the main model described earlier in the present section I opt to keep only constraints 5.10, 5.11, and 5.12. Additionally, the following constraints are introduced:

$$\sum_{s \in \mathbb{S}} y_{fs} <= 1 \quad \forall f \in \mathbb{F} \tag{5.18}$$

$$\sum_{s \in \mathbb{S}} y_{fs} = 1 \quad \forall f \in \mathbb{F}, \quad \sum_{q \in \mathbb{Q}} C_{fq} > 0 \tag{5.19}$$

$$\sum_{f \in \mathbb{F}} \sum_{q \in \mathbb{Q}} C_{fq} * y_{fs} \leq C \quad \forall s \in \mathbb{S} \tag{5.20}$$

$$y_{fs} \leq \sum_{r \in q} y_{rs} \quad \forall f \in \mathbb{F}, \quad \forall q \in \mathbb{Q}, \quad \forall s \in \mathbb{S}, \quad |q| > 0, \quad C_{fq} > 0 \tag{5.21}$$

Constraint 5.18 ensures that any faculty member will appear at most once in the final schedule and constraint 5.19 ensures that supervisors must be scheduled once. As supervisors appear once and only once their candidates are assumed to be in the same session. So, to limit the number of candidates under the threshold of hard constraint **H2**, in each session constraint 5.20 is introduced. Finally, every candidate must have a suitable opponent and this is enforced by constraint 5.21. As the search is for solutions that satisfy the soft constraints there is no need for an objective function and the problem becomes a satisfaction one.

5.6 Experiments and Results

5.6.1 Instances with zero cost

For the following instances the model described in section 5.5.2 proves that a solution with zero cost exists and this solution is optimal as the lowest bound for any instance is also zero:

- Real Dataset: 2, 5, 6, 13, 19.
- Art Dataset: 4, 5, 8, 12, 17, 18, 19, 20, 21, 22, 24, 28, 30, 32, 33, 34, 35, 36, 37, 41, 43, 44.

For the remaining instances this model formulation becomes infeasible meaning that either there exists a solution with a lower bound greater than zero or that the instance itself is infeasible (as is the case of instance art25).

5.6.2 Estimating lower bounds

As discussed in section 5.4, a large number of faculty members in each instance is used only for their academic level. Using a variation of the main model, where these faculty members are removed from the problem and virtual faculty members of all academic levels in enough numbers are introduced to cover every committee without incurring any penalties, some lower bounds for most instances can be obtained. Table 5.5 presents the lower bounds computed. Instances where this lower bound is proven to be optimal are denoted with an asterisk.

Table 5.5: Lower Bounds obtained by the approximation method described in 5.6.2

Instance	Bound	Instance	Bound	Instance	Bound
real01	36*	real18	1	art15	11*
real03	1	real20	1	art16	1*
real04	1	real21	9*	art23	8*
real07	16*	art01	2*	art26	1*
real08	14*	art02	1*	art27	13*
real09	3*	art03	1*	art29	1
real10	12	art06	2	art31	14*
real11	11*	art07	1	art38	1*
real12	1*	art09	1*	art39	7*
real14	1*	art10	1*	art40	6*
real15	9*	art11	1	art42	3*
real16	4*	art13	1*	art45	7*
real17	21*	art14	1*		

* denotes optimality.

5.6.3 Results

Making use of the base model formulation described in section 5.5. Solutions for all instances where a zero cost solution was proven by the model of section 5.5.2 not to exist are gathered. These results are presented in Table 5.6 in contrast to results from Dimitsas et al. [1] and [2] regarding the problem. Lower bounds are the bounds that the solver, Google's OR-Tools CP-SAT [64], returns. Figure 5.1 contains a visualization of the comparison. For the real Dataset 5 instances are with zero cost, in total 17 out of the 21 instances are solved to optimality. For the artificial Dataset 22 instances are with zero cost, in total 39 out of the 46 instances are solved to optimality.

Table 5.6: Results comparison

Instance	Result	[1]	[2]	Instance	Result	[1]	[2]
real01	36*	36	36*	art13	78*	78	78*
real02	0*	0	0*	art14	1*	29	1*
real03	49*	49	49*	art15	162 (161)	181	169
real04	25*	25	25*	art16	44 (1)	62	63
real05	0*	0	0*	art17	0*	0	0*
real06	0*	0	0*	art18	0*	0	0*
real07	31 (20)	32	31*	art19	0*	0	0*
real08	17*	17	17*	art20	0*	0	12
real09	32 (3)	35	62	art21	0*	0	0*
real10	38*	41	40	art22	0*	0	0*
real11	11*	11	13	art23	171*	172	171*
real12	22 (1)	35	34	art24	0*	0	0*
real13	0*	0	0*	art25	infeasible	infeasible	infeasible
real14	1*	1	1*	art26	14*	14	14*
real15	219 (9)	291	306	art27	205 (196)	229	219
real16	7*	7	7*	art28	0*	0	0*
real17	21*	21	21*	art29	188*	196	203
real18	46*	46	46*	art30	0*	0	0*
real19	0*	0	0*	art31	17 (14)	20	26
real20	17*	17	17*	art32	0*	0	0*
real21	9*	9	9*	art33	0*	0	0*
art01	82*	87	82*	art34	0*	0	0*
art02	16*	16	16*	art35	0*	0	0*
art03	1*	1	1*	art36	0*	0	0*
art04	0*	0	0*	art37	0*	0	0*
art05	0*	0	0*	art38	1*	1	1*
art06	106*	117	110	art39	9 (7)	10	11
art07	182*	202	201	art40	8 (6)	12	9
art08	0*	0	0*	art41	0*	0	0*
art09	8*	8	8*	art42	4*	4	4*
art10	12*	12	12*	art43	0*	0	0*
art11	204*	212	215	art44	0*	0	0*
art12	0*	0	0*	art45	7*	7	7

* denotes optimality. Lower bounds in parentheses.

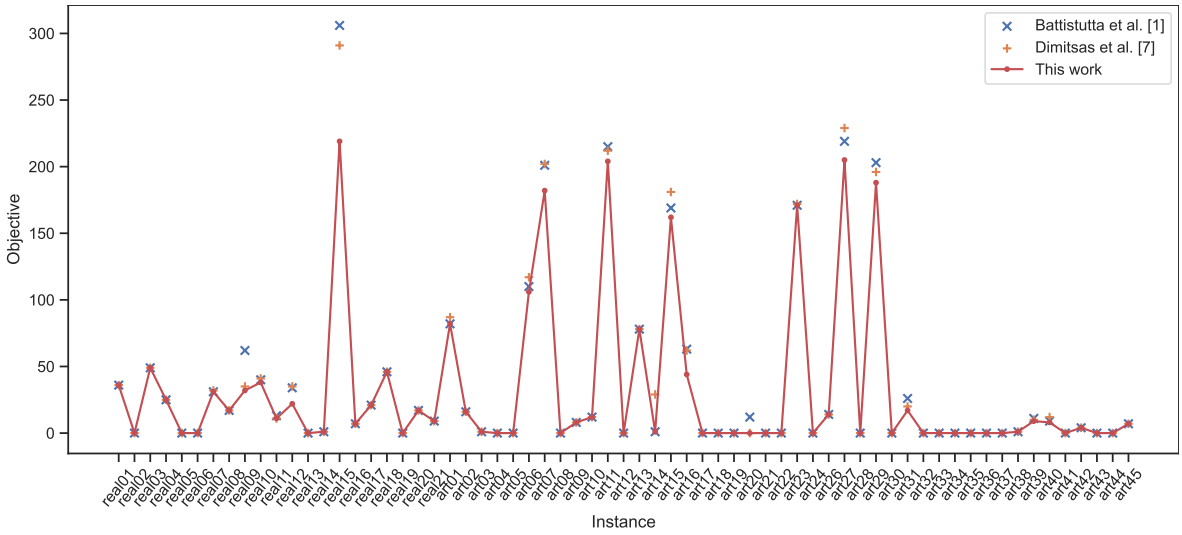

Figure 5.1: Result comparison of this work with [1], and [2].

This version of the Thesis Defense Timetabling Problem was found out to be especially vulnerable to a symmetry elimination process. Almost all instances where solved to optimality. This work resulted in the following publications:

A. Dimitsas, C. Gogos, and E. Pappa, "Better solutions for the Thesis Defense Timetabling problem using a three-phase approach," in *Proceedings of the 26th Pan-Hellenic Conference on Informatics*. Athens Greece: ACM, Nov. 2022, pp. 58–63

A. Dimitsas and C. Gogos, "Finding Near Optimal Solutions to the Thesis Defense Timetabling Problem by Exploiting Symmetries," *Operations Research Forum*, vol. 5, no. 3, p. 65, Jul. 2024

Chapter 6

Case Study: Sports Scheduling

Sports Scheduling is as old as sports themselves, in this Chapter symmetries in round robin tournaments are identified. An exact approach is employed to solve tough combinatorial optimization instances of double round robin tournaments. The participation and the solution process used in the International Timetabling Competition of 2021 is covered.

6.1 Problem Description

Sports Scheduling is the problem of constructing a tournament schedule consisting of matches among competing teams that form a league. The schedule should satisfy the constraints imposed by the tournament's rules and be 'invisible' in the sense that the various stakeholders such as organizers, teams, spectators, and others should not have legitimate reasons to question it.

Sports scheduling exists for as long as there are sports and teams willing to participate in tournaments with matches against each other. For some sports, like tennis, instead of teams, individual athletes compete. Furthermore, there are tournaments, like chess or other board games tournaments, where the actual matches would be hardly identified as sports, in the typical sense. E-sports (electronic sports), is another example of a competition for which its events should be scheduled according to a carefully crafted plan. The same principles regarding scheduling apply to all previously identified cases of tournaments and are special instances of the sports scheduling problem.

Several variations of tournaments exist including single round tournaments, double round tournaments, tournaments with elimination games, compact tournaments (all teams have matches in every time-slot), etc. Some heuristics for constructing sport schedules are known for many years, like the circle method first published by Reverend Thomas Kirkman in 1847, see Lambrechts et al. (2017) [91] and the Berger method introduced by Johann Berger in 1893, see Chen and Dong (2011) [92]. But when constraints are added the problem quickly becomes very hard to solve. Such constraints might involve the avoidance of consecutive away games for all or some teams, the enforcement of minimum distances (number of time slots) between a match and the rematch, and many others. In this work, an approach of generating high quality schedules for the compact, double round robin (2RR) type of tournament, is presented.

6.2 Symmetries in sports scheduling

Symmetries in sports scheduling can vary based on the competition type (tournament, single-elimination knockout etc.). In leagues where fairness is the most important factor symmetries are most prevalent. Let's take a simple example, imagine a single round robin tournament of 8 teams that have to face each other in 7 rounds.

We can construct a schedule using the Circle method, like this in Table 6.1:

Table 6.1: Single round robin tournament for 8 teams.

Round 1	**Round 2**	**Round 3**	**Round 4**	**Round 5**	**Round 6**	**Round 7**
1-8	1-7	1-6	1-5	1-4	1-3	1-2
2-7	8-6	7-5	6-4	5-3	4-2	3-8
3-6	2-5	8-4	7-3	6-2	5-8	4-7
4-5	3-4	2-3	8-2	7-8	6-7	5-6

Now let's examine the home-away pattern of this tournament, in Table 6.2 we can notice that teams have to play an unequal number of breaks (the same team plays two time in a row at home or away). In Table 6.3 there is a comparisson of the total breaks per team.

Table 6.2: Single round robin tournament for 8 teams Home Away patterns.

Team	**Round 1**	**Round 2**	**Round 3**	**Round 4**	**Round 5**	**Round 6**	**Round 7**
1	H	H	H	H	H	H	H
2	H	H	H	A	A	A	A
3	H	H	A	A	A	A	H
4	H	A	A	A	A	H	H
5	A	A	A	A	H	H	H
6	A	A	A	H	H	H	A
7	A	A	H	H	H	A	A
8	A	H	H	H	A	A	A

Table 6.3: Single round robin tournament for 8 teams Breaks.

Team	**Home Breaks**	**Away Breaks**	**Total Breaks**
1	6	0	6
2	2	3	5
3	1	3	4
4	1	3	4
5	2	3	5
6	2	2	4
7	2	2	4
8	2	2	4

Let's try to make this tournament more fair. We can create a tournament where each team has 1 home break and 1 away break, one such schedule is presented in Table 6.4.

Table 6.4: Single round robin tournament for 8 teams. All team have a home break and an away break.

Round 1	**Round 2**	**Round 3**	**Round 4**	**Round 5**	**Round 6**	**Round 7**
4-3	3-1	1-7	2-1	1-6	1-8	4-1
5-1	5-4	2-5	4-7	3-2	2-4	5-3
6-2	7-6	4-6	5-8	7-5	3-7	6-8
7-8	8-2	8-3	6-3	8-4	6-5	7-2

Because the requirement is the same among all teams. We could fix the first round for example team 1 plays home against 2 in round 1, team 3 plays home against 4 in round 1 etc. If we do this we can enumerate all feasible solutions: $173,568$. Otherwise, without restricting the teams, because we can choose any permutation of teams the total number of feasible solutions would be: $8! * 173,568$.

6.3 International Timetabling Competition 2021

The International Timetabling Competition 2021 (ITC2021) [1] was dedicated to automated sports timetabling. The competition's problem consists of constructing a compact double round-robin tournament with 16 to 20 teams while respecting various hard constraints and minimizing the penalties from violated soft constraints. The problem description for this specific version can be found at Van Bulck et al. (2021) [93] and it refers to tournaments categorized as time-constrained double round robin. Time-constrained or compact means that the timetable uses the minimum number of time slots, i.e. in each time slot all teams play in matches.

6.3.1 The Base Constraints

The base constraints for each tournament are the format of the tournament. All tournaments are in double round robin format, i.e. each team has two matches against every other team, one at home and one away. Some of the tournaments contain the Phase rule; the timetable is divided in half (two phases), a match and its rematch must be in a different phase. All tournaments are compact.

[1] https://robinxval.ugent.be/ITC2021/

6.3.2 The Hard and Soft Constraints of ITC2021

All type of constraints can be either hard or soft as of the type attribute. Hard constraints must be satisfied and soft constraints create deviations penalized in the objective function. There are 9 types of constraints in 5 different constraint categories.

Capacity Constraints Capacity constraints regulate the matches played by a team or a group of teams at home or away.

CA1 constraints regulate the number of matches a team plays at home or away in specific slots.

CA2 constraints regulate the number of matches a team plays at home or away in specific slots against specific teams.

CA3 constraints regulate the number of matches a team plays at home or away in a sequence of slots.

CA4 constraints regulate the number of matches a group of teams play at home or away in specific slots against specific teams.

Game Constraints Game constraints enforce or forbid specific matches in certain slots.

GA1 constraints deal with fixed or forbidden matches to slots assignments.

Break Constraints If a team plays a game with the same home-away status as its previous game, we say it has a break.

BR1 constraints limit the breaks a team has in specific slots.

BR2 constraints limit the breaks a group of teams has in specific slots.

Fairness Constraints Fairness constraints attempt to increase fairness and attractiveness of a tournament.

FA2 constraints limit the difference in played home games of set of teams.

Separation Constraints Separation constraints regulate the number of slots between matches involving the same pairs of teams.

SE1 limits the difference between matches and rematches of the same teams.

6.4 Related work

Several real life tournaments have been addressed using automated techniques involving mathematical programming, constraint programming, metaheuristics and heuristics; e.g., the Belgian soccer league by Goossens et al. (2009) [94], the Brazilian soccer tournament by Ribeiro and Urrutia (2012) [95], the Finnish national youth ice hockey league by Nurmi et al. (2014)e [96], the Chilean soccer leagues by Alarcón et al. (2017) [97], and the South American qualifiers for FIFA 2018 by Durán et al. (2017) [98].

Lewis and Thompson (2011) [99] present the association of the sports scheduling problem to the graph coloring problem. Moreover, an edge coloring presentation of the problem is available in Januario et al. (2016) [100].

Regarding the exploration of the solution space in Costa et al. (2012) [101] it is established that the solution space is not connected by the usually used neighborhood structures, i.e. it's impossible starting from a feasible timetable to reach all other possible timetables just by performing the usual heuristic moves proposed in the bibliography, and Januario and Urrutioa (2016) [102] proposed a new neighborhood operator to handle this issue.

Since sports timetabling usually results in problems of big sizes, decomposition approaches can be advantageous. In Trick (2000) [103] a first schedule then break approach was tried. First it was decided when teams would meet, and the home advantage is decided later. The opposite, first break then schedule approach can be seen at Riberio (2013) [104], first it is decided where each team plays at home and the teams are paired later. An effort on minimizing breaks is available by Miyashiro (2003) [105]. A research on feasible home-away patterns is presented by Briskorn (2008) [106].

6.5 Dataset

In table 6.5 some base characteristics of the ITC2021 competition are presented. 2RR is a double round robin tournament, C is for compact, P means there is a phase rule, and finally SC is the objective, minimize over soft constraints.

Table 6.5: Descriptive Statistics for the ITC 2021 dataset

Instance name	Teams	Slots	Classification
Early 1	16	30	2RR, C, P | BR1, BR2, CA1, CA2, CA4, FA2, GA1, SE1 | SC
Early 2	16	30	2RR, C, P | BR1, BR2, CA1, CA3, FA2, GA1 | SC
Early 3	16	30	2RR, C, P | BR1, BR2, CA1, CA2, CA3, FA2, GA1 | SC
Early 4	18	34	2RR, C, P | BR1, BR2, CA1, CA2, CA4, GA1, SE1 | SC
Early 5	18	34	2RR, C, P | BR1, BR2, CA1, CA2, CA3, CA4, GA1, SE1 | SC
Early 6	18	34	2RR, C, P | BR2, CA1, CA2, CA3, CA4, FA2, GA1, SE1 | SC
Early 7	18	34	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, CA4, GA1, SE1 | SC
Early 8	18	34	2RR, C, NULL | BR1, CA1, CA2, CA3, CA4, FA2, GA1 | SC
Early 9	18	34	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, FA2, GA1 | SC
Early 10	20	38	2RR, C, P | BR1, BR2, CA1, CA2, CA3, CA4, SE1 | SC
Early 11	20	38	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, CA4, GA1, SE1 | SC
Early 12	20	38	2RR, C, P | BR1, BR2, CA1, CA2, CA3, CA4, GA1 | SC
Early 13	20	38	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, GA1 | SC
Early 14	20	38	2RR, C, NULL | BR1, BR2, CA1, FA2, GA1 | SC
Early 15	20	38	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, CA4, FA2, GA1 | SC
Middle 1	16	30	2RR, C, P | BR1, BR2, CA1, CA2, CA4, SE1 | SC
Middle 2	16	30	2RR, C, P | BR1, BR2, CA1, CA2, CA3, CA4, GA1, SE1 | SC
Middle 3	16	30	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, CA4, FA2, GA1, SE1 | SC
Middle 4	18	34	2RR, C, P | BR1, CA1, CA2, CA3, CA4, GA1 | SC
Middle 5	18	34	2RR, C, P | BR1, BR2, CA1, CA2, CA3, FA2, GA1 | SC
Middle 6	18	34	2RR, C, P | BR1, BR2, CA1, CA2, CA3, CA4, GA1, SE1 | SC
Middle 7	18	34	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, CA4, GA1, SE1 | SC
Middle 8	18	34	2RR, C, NULL | BR1, CA1, CA2, CA3, CA4, GA1 | SC
Middle 9	18	34	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, CA4, FA2, GA1 | SC
Middle 10	20	38	2RR, C, P | BR1, BR2, CA1, CA2, CA4, GA1 | SC
Middle 11	20	38	2RR, C, P | BR1, CA1, CA2, CA3, CA4, FA2, GA1 | SC
Middle 12	20	38	2RR, C, P | BR1, BR2, CA1, CA2, CA3, FA2, GA1, SE1 | SC
Middle 13	20	38	2RR, C, NULL | BR1, CA1, CA2, CA3, CA4, GA1, SE1 | SC
Middle 14	20	38	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, CA4, FA2, GA1 | SC
Middle 15	20	38	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, GA1, SE1 | SC
Late 1	16	30	2RR, C, NULL | BR1, CA1, CA2, CA3, CA4, FA2, GA1 | SC
Late 2	16	30	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, CA4, GA1 | SC
Late 3	16	30	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, CA4, FA2, GA1, SE1 | SC
Late 4	18	34	2RR, C, P | BR1, CA1, CA4, GA1, SE1 | SC
Late 5	18	34	2RR, C, P | BR2, CA1, CA2, CA3, CA4, FA2, GA1 | SC
Late 6	18	34	2RR, C, P | BR1, BR2, CA1, CA2, CA4, GA1, SE1 | SC
Late 7	18	34	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, GA1, SE1 | SC
Late 8	18	34	2RR, C, P | BR1, BR2, CA1, CA2, CA3, GA1, SE1 | SC
Late 9	18	34	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, FA2, GA1 | SC
Late 10	20	38	2RR, C, P | BR1, BR2, CA1, CA2, CA3, CA4, GA1, SE1 | SC
Late 11	20	38	2RR, C, P | BR1, BR2, CA1, CA2, CA3, FA2, GA1 | SC
Late 12	20	38	2RR, C, NULL | BR1, BR2, CA1, CA2, CA3, CA4, SE1 | SC
Late 13	20	38	2RR, C, NULL | BR2, CA1, CA2, CA3, CA4, FA2, GA1, SE1 | SC
Late 14	20	38	2RR, C, NULL | BR1, CA1, CA2, CA3, CA4, FA2, GA1 | SC
Late 15	20	38	2RR, C, NULL | BR1, BR2, CA1, CA3, FA2, GA1 | SC

6.6 Constraint Programming Formulation

Decision Variables

For the set of teams $\mathbb{T}$, the set of slots $\mathbb{S}$, with S as the number of available slots and T as the number of teams we define the following binary decision variables.

$$x_{i,j,s} = \begin{cases} 1, & \text{If team i plays against team j in slot s} \\ 0, & \text{Otherwise} \end{cases} \quad \forall i,j \in \mathbb{T}, i \neq j, \forall s \in \mathbb{S} \tag{6.1}$$

To monitor the home away pattern we define:

$$y_{i,s} = \begin{cases} 1, & \text{If team i plays at home in slot s} \\ 0, & \text{Otherwise} \end{cases} \quad \forall i \in \mathbb{T}, \forall s \in \mathbb{S} \tag{6.2}$$

We enforce the home-away pattern to follow the timetable:

$$y_{i,s} = \sum_{j=1}^{T} x_{i,j,s} \quad \forall i \in \mathbb{T}, i \neq j, \forall s \in \mathbb{S} \tag{6.3}$$

In all instances, constraints regarding breaks do not take into consideration if the breaks occur at Home or Away, so we just have to keep track in which slots a general break occurs:

$$z_{i,s} = \begin{cases} 1, & \text{If team i has a break in slot s} \\ 0, & \text{Otherwise} \end{cases} \quad \forall i \in \mathbb{T}, \forall s \in \mathbb{S} \tag{6.4}$$

We enforce the break pattern to follow the home-away pattern:

$$z_{i,s} = \begin{cases} 1, & \text{if } y_{i,s} = y_{i,s-1}, s > 1 \\ 0, & \text{if } s = 1 \end{cases} \quad \forall i \in \mathbb{T}, \forall s \in \mathbb{S} \tag{6.5}$$

Base Constraints

Each team must play exactly one match at home against each other team:

$$\sum_{s=1}^{S} x_{i,j,s} = 1 \quad \forall i,j \in \mathbb{T}, i \neq j \tag{6.6}$$

To satisfy the compactness rule each team plays one match in each slot:

$$\sum_{j=1}^{T} (x_{i,j,s} + x_{j,i,s}) = 1 \quad \forall i \in \mathbb{T}, i \neq j, \forall s \in \mathbb{S} \tag{6.7}$$

For instances with the phase rule a match and its rematch must be in different

phases:

$$\sum_{s}^{S/2}(x_{i,j,s} + x_{j,i,s}) = 1 \quad \forall i,j \in \mathbb{T}, i < j, \forall s \in \mathbb{S} \tag{6.8}$$

CA1 Constraints

Each CA1 constraint with team t_c in "teams" field, with $\mathbb{S}_c$ the set of teams in "slots" field and max_c in "max" field, triggers a d_c deviation.

CA1 with mode="H":

$$d_c = \sum_{s \in \mathbb{S}_c} y_{t_c,s} - max_c \tag{6.9}$$

CA1 with mode="A" and S_c the size of $\mathbb{S}_c$:

$$d_c = S_c - \sum_{s \in \mathbb{S}_c} y_{t_c,s} - max_c \tag{6.10}$$

CA2 Constraints

Each CA2 with team t_1 in "teams1" field, with $\mathbb{S}_c$ the set of slot in "slots" field, with $\mathbb{T}_c$ the set of slots in "teams2" field, with max_c in "max" field triggers a deviation d_c.

CA2 with mode="H":

$$d_c = \sum_{t_2 \in \mathbb{T}_c} \sum_{s \in \mathbb{S}_c} x_{t_1,t_2,s} - max_c \tag{6.11}$$

CA2 with mode="A":

$$d_c = \sum_{t_2 \in \mathbb{T}_c} \sum_{s \in \mathbb{S}_c} x_{t_2,t_1,s} - max_c \tag{6.12}$$

CA2 with mode="HA":

$$d_c = \sum_{t_2 \in \mathbb{T}_c} \sum_{s \in \mathbb{S}_c} (x_{t_1,t_2,s} + x_{t_2,t_1,s}) - max_c \tag{6.13}$$

CA3 Constraints

Each CA3 with $\mathbb{T}_{c1}$ the set of teams in "teams1" field, with $\mathbb{S}_c$ as the slots in "slots" field, with $\mathbb{T}_{c2}$ the set of teams in "teams2" field and max_c in "max" field triggers deviations d_c for each team in $\mathbb{T}_{c1}$ and for all slot sequences $\mathbb{S}_c$ of size *intp* in "intp" field.

CA3 with mode="H":

$$d_c = \sum_{t_2 \in \mathbb{T}_{c2}} \sum_{s=k}^{k+intp} x_{t_1,t_2,s} - max_c \quad \forall t_1 \in \mathbb{T}_{c1}, t_1 \neq t_2, 1 \leq k \leq S_c - intp \tag{6.14}$$

CA3 with mode="A":

$$d_c = \sum_{t_2 \in \mathbb{T}_{c2}} \sum_{s=k}^{k+intp} x_{t_2,t_1,s} - max_c \quad \forall t_1 \in \mathbb{T}_{c1}, t_1 \neq t_2, 1 \leq k \leq S_c - intp \tag{6.15}$$

Special case: In all instances there are at most two Hard CA3 constraints, one with mode="H" and the other with mode="A", $\mathbb{T}_{c1} = \mathbb{T}_{c2} = \mathbb{T}$, $\mathbb{S}_c = \mathbb{S}$, max_c is always 2 and *intp* is always 3. If both rules exist then the home-away patterns "HHH" and "AAA" cannot appear, so for those instances a team cannot have two breaks in a row:

$$z_{i,s} + z_{i,s-1} \leq 1 \quad \forall i \in \mathbb{T}, \forall s \in \mathbb{S}, s > 2 \tag{6.16}$$

CA4 Constraints Each CA4 with mode2="GLOBAL" triggers a deviation d_c equal to the sum of the matches between the set of teams $\mathbb{T}_{c1}$ in "teams1" field and the set of teams $\mathbb{T}_{c2}$ in "teams2" field in all slots of the set $\mathbb{S}_c$ in "slots" field over max_c in "max" field.

CA4 with mode2="GLOBAL" and mode1="H":

$$d_c = \sum_{s \in \mathbb{S}_c} \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} x_{t_1,t_2,s} - max_c \quad t_1 \neq t_2 \tag{6.17}$$

CA4 with mode2="GLOBAL" and mode1="A":

$$d_c = \sum_{s \in \mathbb{S}_c} \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} x_{t_2,t_1,s} - max_c \quad t_1 \neq t_2 \tag{6.18}$$

CA4 with mode2="GLOBAL" and mode1="HA":

$$d_c = \sum_{s \in \mathbb{S}_c} \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} (x_{t_1,t_2,s} + x_{t_2,t_1,s}) - max_c \quad t_1 \neq t_2 \tag{6.19}$$

Each CA4 with mode2="EVERY" triggers a deviation d_c for each slot of the slots set $\mathbb{S}_c$ in "slots" field equal to the sum of the matches between the set of teams $\mathbb{T}_{c1}$ in "teams1" field and the set of teams $\mathbb{T}_{c2}$ in "teams2" field over max_c in "max" field.

CA4 with mode2="EVERY" and mode1="H":

$$d_c = \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} x_{t_1,t_2,s} - max_c \quad t_1 \neq t_2, \forall s \in \mathbb{S}_c \tag{6.20}$$

CA4 with mode2="EVERY" and mode1="A":

$$d_c = \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} x_{t_2,t_1,s} - max_c \quad t_1 \neq t_2, \forall s \in \mathbb{S}_c \tag{6.21}$$

CA4 with mode2="EVERY" and mode1="HA":

$$d_c = \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} (x_{t_1,t_2,s} + x_{t_2,t_1,s}) - max_c \quad t_1 \neq t_2, \forall s \in \mathbb{S}_c \tag{6.22}$$

GA1 Constraints Each GA1 triggers a deviation d_c calculated as the sum of matches of the set $\mathbb{M}_c$ in field "meetings" which occur in set of slots S_c in field "slots" under min_c in field "min" or over max_c in field "max".

$$d_c = \sum_{s \in \mathbb{S}_c} \sum_{t_1,t_2 \in \mathbb{T}_c} x_{t_1,t_2,s} - max_c \tag{6.23}$$

$$d_c = \sum_{s \in \mathbb{S}_c} \sum_{t_1,t_2 \in \mathbb{T}_c} x_{t_1,t_2,s} + min_c \tag{6.24}$$

BR1 Constraints Each BR1 with t_c the team in "teams" field, triggers a deviation d_c equal to the sum of teams t_c breaks in set of slots $\mathbb{S}_c$ in "slots" field over max_c in "max" field.

$$d_c = \sum_{s \in \mathbb{S}_c} z_{t_c,s} - max_c \tag{6.25}$$

BR2 Constraints In all instances where a BR2 constraint exists field "teams" contains all teams and field "slots" contains all slots except the first slot (as a team cannot have a break in the first slot). As such, a BR2 constraint triggers a deviation d_c equal to the sum of all breaks of all teams over *max* in "max" field.

$$d_c = \sum_{t \in \mathbb{T}} \sum_{s \in \mathbb{S}} z_{t,s} - max_c \tag{6.26}$$

FA2 Constraints In all instances where an FA2 constraint exists field "teams" contains all teams and field "slots" contains all slots. As such, an FA2 constraint triggers deviations d_c for each pair of teams equal to the largest difference in played home

games over all slots more than *intp* in "intp" field.

$$d_c = \max_{s \in \mathbb{S}} \left(\sum_{m=1}^{s} y_{i,m} - \sum_{m=1}^{s} y_{j,m} - intp; 0\right) \quad \forall i,j \in \mathbb{T}, i < j \tag{6.27}$$

SE1 Constraints For SE1 we need to keep track of the distance between matches and rematches for all combinations of the set of teams $\mathbb{T}_c$ in field "teams". Each combination of teams triggers a deviation d_c equal to the sum of the number of time slots less than *min* in "min" field between the match and the rematch.

$$d_c = \left| \sum_{s \in \mathbb{S}} s * x_{t_1,t_2,s} - \sum_{s \in \mathbb{S}} s * x_{t_2,t_1,s} \right| - min_c \quad t1 \neq t2, \forall t_1, t_2 \in \mathbb{T}_c \tag{6.28}$$

Objective Function Hard constraints must not generate any deviation. Soft constraints' deviations are multiplied by p_c denoted by the field "penalty" and summed. Deviations under zero are ignored.

$$min \quad \sum_{c \in \mathbb{C}} d_c * p_c \tag{6.29}$$

6.6.1 Results

The hybrid process was able to produce solutions for 37 out of 45 instances. The objective of the solutions can be seen in Table 6.6. Solution files are available at this github[2] repository. This process managed to solve 37 out of the 45 instances.

[2]https://github.com/AngelosDimitsas/papers/tree/main

Table 6.6: Results after three hours of execution time for each instance using the hybrid process. Objective is presented as the tuple (deviation of hard constraints, penalty of soft constraints).

Instance	Objective	Instance	Objective	Instance	Objective
Early 1	0, 512	Middle 1	17, -	Late 1	0, 2234
Early 2	0, 266	Middle 2	48, -	Late 2	0, 5680
Early 3	0, 1354	Middle 3	0, 12170	Late 3	0, 3004
Early 4	6, -	Middle 4	0, 7	Late 4	0, 0
Early 5	5, -	Middle 5	0, 732	Late 5	39, -
Early 6	0, 3957	Middle 6	0, 1900	Late 6	0, 1440
Early 7	0, 9644	Middle 7	0, 2792	Late 7	0, 3009
Early 8	0, 1614	Middle 8	0, 301	Late 8	0, 1375
Early 9	0, 448	Middle 9	0, 1015	Late 9	0, 1108
Early 10	32, -	Middle 10	1, -	Late 10	6, -
Early 11	0, 8189	Middle 11	0, 2956	Late 11	0, 511
Early 12	0, 1025	Middle 12	0, 1596	Late 12	0, 7218
Early 13	0, 380	Middle 13	0, 780	Late 13	0, 3576
Early 14	0, 63	Middle 14	0, 1619	Late 14	0, 1650
Early 15	0, 4470	Middle 15	0, 1833	Late 15	0, 80

Sports scheduling has several facets, mainly multiple stakeholders: teams, police, the board of directors, broadcasters to name a few, that make it an interesting and difficult problem. Sports scheduling problems are proved to be, in practice (and in theory), hard to solve. Sometimes even finding a feasible solution or proving that such a solution does not exist is extremely challenging. This work managed to receive an entry to the competition and a joint work by the competitions members. Related papers and conferences:

A. Dimitsas, C. Gogos, C. Valouxis, A. Tzallas, and P. Alefragis, "A pragmatic approach for solving the sports scheduling problem," in *Proc. 13th Int. Conf. Pract. Theory Autom. Timetabling, PATAT*, vol. 3, 2022, pp. 195–207

D. Van Bulck, D. Goossens, J.-P. Clarner, A. Dimitsas, G. H. Fonseca, C. Lamas-Fernandez, A. E. Phillips, and R. M. Rosati, "What algorithm to select to create your sports schedule?" in *MathSport International 2023*, 2023, pp. 48–48

D. Van Bulck, D. Goossens, J.-P. Clarner, A. Dimitsas, G. H. Fonseca, C. Lamas-Fernandez, M. M. Lester, J. Pedersen, A. E. Phillips, and R. M. Rosati, "Which algorithm to select in sports timetabling?" *European Journal of Operational Research*, vol. 318, no. 2, pp. 575–591, 2024

Chapter 7

Case Study: One-Machine Scheduling with Time-Dependent Capacity

The One-Machine Scheduling with Time-Dependent Capacity problem arose in scheduling the charging times of a fleet of electric vehicles. In this Chapter a multi-staged approach is presented. Constraint programming and Heuristics as-well as symmetry elimination is presented.

7.1 Problem Description

A detailed description of the problem exists in Mencía and Mencía (2021) [3], so only a brief description is provided. The problem involves n jobs and one machine with a certain capacity that varies over time. Each job i has duration P_i and due date D_i. All jobs are available from the start of time ($t = 0$) and consume one unit of the machine's capacity for the period that the job will eventually be scheduled. Once a job starts, it cannot be preempted and should continue executing until completion. It is imperative that the capacity of the machine cannot be exceeded at any time. Finally, the objective that should be minimized is the total tardiness of all jobs, which is computed based on the due dates of the jobs. If a job i completes execution before its due date, it does not affect the cost. Otherwise, it imposes a cost equal to $c_i - D_i$, where c_i is the completion time that job i assumes at the schedule. The mathematical formulation of the problem is presented in Section 7.6.

The problem $(1, Cap(t)||\sum t_i)$ is NP-hard, since $(1||\sum t_i)$ and $(P||\sum t_i)$ problems (P denotes a known number of identical machines), which are known to be NP-hard as shown by Koulamas (2010) [110] can be reduced to it.

7.1.1 Terminology

In line with the definition of terms in Mencía et al. (2021) [3] the same notation is used, S_i, p_i, d_i, C_i for start time, duration, due time, and completion time, respectively, of a job i in a given schedule. Then, T_i is the tardiness of job i which is $max\{0, C_i - d_i\}$.

A concrete example is presented below, which involves 12 jobs, and a capacity line that reaches a maximum of 4 units. This example is the one used as Example 1 in Mencía et al. (2019) [111]. Table 7.1 summarizes information related to it, alongside with values associated with an optimal schedule for this problem instance, achieving an optimal cost of 20. The schedule corresponding to the table's third line is presented graphically in Figure 7.1.

i	1	2	3	4	5	6	7	8	9	10	11	12
p_i, d_i	4,4	4,9	2,13	3,4	4,7	3,8	2,10	3,3	2,13	3,5	3,9	5,7
S_i, C_i, T_i	4,8,4	8,12,3	10,12,0	2,5,1	6,10,3	5,8,0	8,10,0	0,3,0	12,14,1	3,6,1	6,9,0	9,14,7

Table 7.1: A sample problem instance with 12 jobs. For each job i, the table shows its duration p_i and its due time d_i. Also, for a certain schedule, the table shows for each job i its start time (S_i), its completion time (C_i) and the penalty it incurs (T_i).

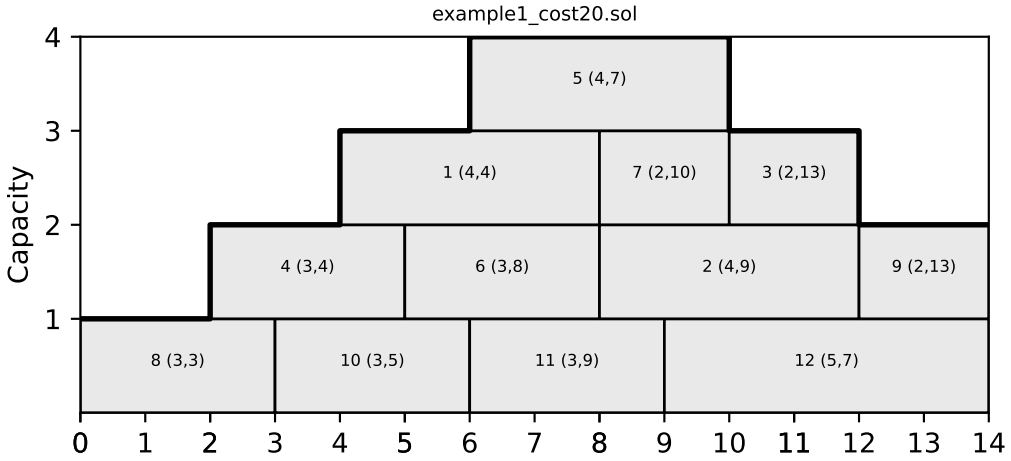

Figure 7.1: A graphical representation of the schedule in the last line of Table 7.1.

7.2 Related Work

Hard combinatorial optimization problems like the One-Machine Scheduling with Time-Dependent Capacity problem are approached using numerous solving methods Lenstra et al. (1977) [112], Gupta et al. (1987) [113]. For a work related to renewable energy consult Alefragkis et al. (2022) [114] The two basic categories of such approaches are the exact ones and the heuristics-metaheuristics ones. In the first category, one can identify Mathematical Programming (i.e., Linear Programming, Integer Programming, and others) Gogos et al. [115], Constraint Programming Baptiste et al. (2001) [116] and Valouxis et al. (2018) [117] and Gogos (2023) [118], approaches based on SAT (satisfiability) Großmann et al. (2012) [119] or SMT (Satisfiability Modulo Theory) solvers Ansótegui et al. (2011) [120] and in general methods that intelligently examine the complete search space while pruning parts of it during their quest for proven optimal solutions Brucker et al. (1998) [121]. In the second category the approaches are numerous, including Local Search methods Vaessens (1995) [122] and Matsuo et al. (1989) [123], Genetic Algorithms Lee et al. (1998) [124], Genetic Programming Gil-Gala et al. (2019) [125], Memetic Algorithms França et al. (2001) [126], Differential Evolution Wu and Che (2019) [127], Ant Colony Optimization Merkle and Middendorf (2003) [128], Particle Swarm Optimization Lin et al. (2010) [129], Bees Algorithms Yuce et al. (2017) [130], Hyper-

heuristics Gil-Gala et al. (2020) [131] and others. For industrial task scheduling you are referred to Gogos et al. (2025) [132] and Alefragis et al. (2025) [133].

7.2.1 Heuristically Constructed Schedules

In Mencía and Mencía (2021) [3], authors present the schedule builder algorithm, where jobs are ordered in an arbitrary sequence, and each job is scheduled to start at the earliest possible time. After positioning each job, the capacity of the machine is updated in accordance with the partial schedule. When all jobs are scheduled, the algorithm finishes and returns a feasible schedule. This search space is guaranteed to contain an optimal solution to any problem instance Mencía et al. (2019) [111]. So, each possible solution can be represented as a sequence of jobs to the schedule builder. Certain metaheuristic algorithms like Genetic Algorithms may be benefited by the idea of representing each possible schedule as a sequence of jobs and search through the space of all possible permutations.

Algorithm 1 Schedule Builder using lanes

```
Input: A problem instance P
Output: A feasible schedule S
  demand ← [0,...]
  left ← 0                ▷ Leftmost time point where capacity is not yet fully utilized
  for all jobs job do
      t ← left
      while True do
          flag = True
          for all capacity[t:t+job.duration], demand[t:t+job.duration] c, d do
              if d > c then
                  flag ← False
                  break
              end if
          end for
          if flag then
              lane = find_lowest_available_lane(job, t)
              S[job.id] ← lane, t
              for all [t:t+job.duration] t' do
                  demand[t'] ← demand[t'] + 1
                  if demand[t'] = capacity[t'] then
                      left ← t' + 1
                  end if
              end for
              t ← t + 1
              break
          end if
      end while
  end for
```

Here, algorithm 1 is presented a modification of the schedule builder algorithm [3] that introduces lanes, which are levels formed by the capacity of the problem. So, for example, a problem with a maximum capacity of four has four lanes, the first that is always available during the time horizon and three more that, based on the capacity, have periods of availability and unavailability. Lanes help plot schedules, disambiguate solutions with identical costs, and quickly identify jobs that form sequences of consecutive jobs.

In algorithm 1, the function `find_lowest_available_lane` is called this finds the lowest available lane that can accommodate a job starting at period `t`.

7.3 C-Paths

A fundamental concept that was introduced by Mencía et al. in (2017) [134] is the concept of a C-Path. A C-Path is a sequence of consecutive jobs (i.e., in a C-Path, the finish time of each previous job coincides with the start time of the following job) in a schedule. The importance of C-Paths stems from the fact that jobs in each C-Path can easily swap places and keep the schedule feasible. We can consider a graph view of a schedule, where each job is a node, and directed edges connect nodes that correspond to consecutive jobs. Then, each path from a source node of the graph (i.e., a node with no incoming edges) to a sink node (i.e., a node with no outgoing edges) is a C-Path. This is demonstrated in Figure 7.2 for a sample schedule of cost 35, for the toy problem instance of Table 7.1, alongside with its corresponding graph in Figure 7.3. The list of C-Paths are identified in this graph are the following 6 ones, (3,12,4), (3,10,1,6), (3,10,1,2), (7,9,8,5), (7,11,2) and (7,11,6).

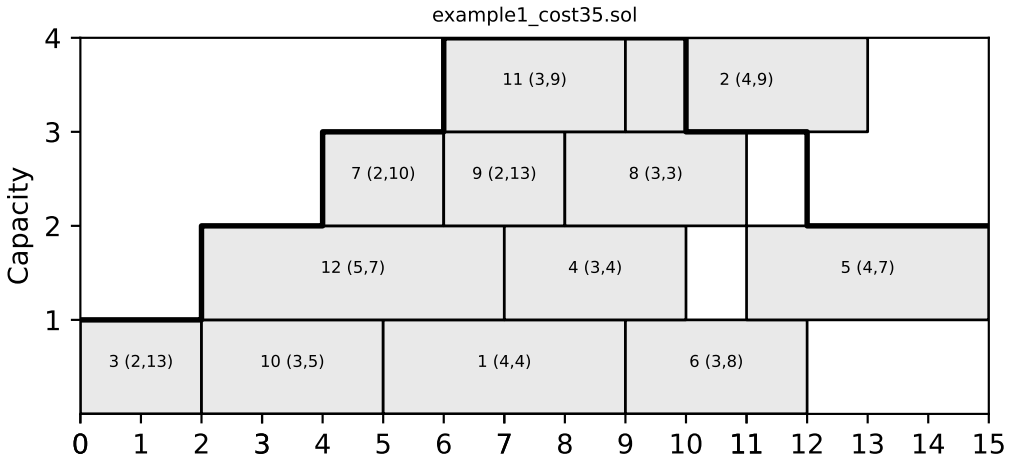

Figure 7.2: A suboptimal schedule of cost 35 for the toy problem of Table 7.3. Each job is depicted with a box, annotated with a label of the form $x(y, z)$, where x is the job identification number, y is the duration of the job, and z is its due time.

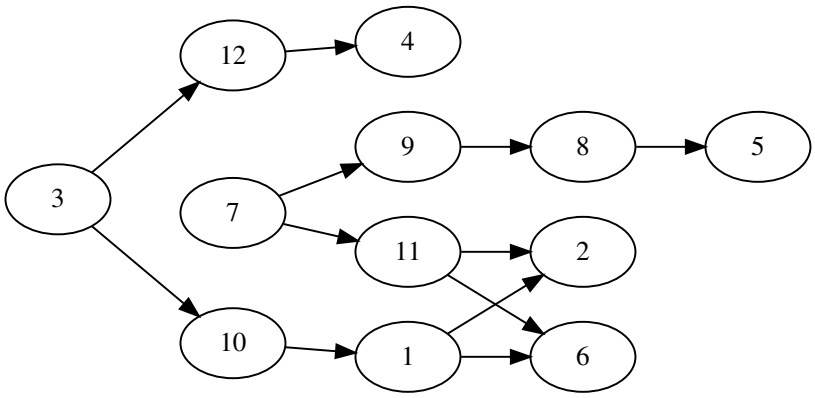

Figure 7.3: The graph that corresponds to the schedule of Figure 7.2

The number of C-Paths might be very large, especially for schedules of big-size problems. This is demonstrated in Table 7.2, which shows the number of C-Paths for specific schedules of selected problem instances. Note that the number of C-Paths might change dramatically for different schedules of the same problem instance and that larger problem instances might have fewer C-Paths than smaller problem instances for some schedules.

Problem instance	Schedule cost	Number of C-Paths
i120_3_1	848	3
i120_10_1	749	71
i250_10_1	4094	48
i250_30_1	3013	276
i500_10_1	4614	204
i500_30_1	2670	4528
i750_10_1	4409	22302
i750_50_1	5134	206022
i1000_10_1	641	903826
i1000_100_1	71012	216694

Table 7.2: Number of C-Paths for schedules of selected problem instances, which can be found at https://github.com/chgogos/1MSTDC.

7.3.1 Fast computation of C-Paths

Since complete enumeration of all C-Paths is out of the question for problems of large sizes, and a faster method of generating a single C-Path each time it is needed a new method was developed. This method starts by picking a random job, followed by two processes that find the right and the left part of a C-Path, having the selected job as a pivot element. The right side part of the C-path is formed by choosing a next job that starts at the finish time of the current job. If more than one such jobs exist, one of them is randomly selected and becomes the new current job. This process

continues until no more subsequent jobs are found. The left side of the C-Path is formed by setting once again as the current job the initially selected job and finding previous jobs that end at the start time of the current job. Similarly, if more than one exists, one is chosen at random and becomes the new current job. The process ends when no more suitable prior jobs can be found.

7.4 Dataset

A dataset consisting of a relatively large number of artificially generated problem instances is publicly available in github repo[1]. The procedure of generating these instances is described in [3] and special care has been taken so as the problems' structures to resemble the structure manifested during the process of electric vehicles charging Hernandez et al. [135]. In total, 190 problem instances exist, as seen in Table 7.3. The naming of each problem instance is `i<n>_<MC>_<k>`, where `n` is the number of jobs, `MC` is the Maximum Capacity and `k` is the sequence number of each problem instance for this `n`, `MC` pair.

Number of jobs (n)	Maximum Capacity (MC)
120	3, 5, 7, 10
250	10, 20, 30
500	10, 20, 30
750	10, 20, 30, 50
1000	10, 20, 30, 50, 100

Table 7.3: Problem instances in the dataset. For each pair of number of jobs and maximum capacity, 10 individual problem instances exist.

Note that the capacity in all problem instances is a unimodal step function that grows until reaching a peak, then decreases and finally stabilizes at a positive value.

7.5 Symmetry and Due times rule

This work contributes to identifying a rule that involves due times of jobs with equal durations. The rule states that "Jobs with equal duration should be scheduled in the order of their due times". In other words, if two jobs have equal duration, they should swap places if the due time of the job that is scheduled later is sooner than the due time of the job that is scheduled sooner. This rule can be used to strengthen mathematical formulations or as a heuristic for reaching better solutions.

[1] https://github.com/raulmencia/One-Machine-Scheduling-with-Time-Dependent-Capacity-via-Efficient-Memetic-Algorithms

Proof. Suppose that two jobs, i and j, with the same duration, are scheduled such that job i comes first and job j follows. Also, suppose that job j has earlier due time than job i, i.e. $d_j < d_i$. Let t and t' be the finish times of jobs i and j, respectively, and since both jobs have the same duration, $t < t'$ holds. When both jobs have non-negative tardiness values, i.e. $t - d_i \geq 0$ and $t' - d_j \geq 0$, swapping the jobs results again in non negative tardiness values for both jobs. This holds because the tardiness of job j after swap will be $t - d_j > t - d_i \geq 0$. Moreover, $t' > t \geq d_i$, so the tardiness of job i after swap will be $t' - d_i \geq 0$. So, the total tardiness before swap is $(t - d_i) + (t' - d_j)$, which is equal to the total tardiness after swap which is $(t - d_j) + (t' - d_i)$. This situation is depicted in Figure 7.4(a). The top figure shows a possible configuration of two equal-duration jobs, that both incur tardiness. It can be easily seen that the total length of the gray bars that represent tardiness remains the same after swapping jobs i and j.

The benefit of positioning equal duration jobs, according to due times, occurs when job i completes execution before its due time, and job j incurs some positive value of tardiness. Since job i has no tardiness, the total tardiness of the initial configuration is $t' - d_j$. After the swap the total tardiness will be $(t - d_j) + (t' - d_i)$. In order to prove that the total tardiness is no greater after the swap, the following inequality must hold, $(t - d_j) + (t' - d_i) \leq t' - d_j$. The inequality leads to a true proposition as follows, $(t - d_j) + (t' - d_i) \leq t' - d_j \implies t - d_i \leq 0 \implies t \leq d_i$. The last inequality holds since it is the assumption made for this case, i.e. job i has no tardiness. A visual representation of such a situation is depicted in Figure7.4 (b). The upper part of the figure shows the situation where job i is scheduled first, while the lower part of the figure shows the situation after swapping the jobs. Again, the gray bars represent the tardiness of the jobs, and it can be seen that the total tardiness is decreased when jobs swap places. □

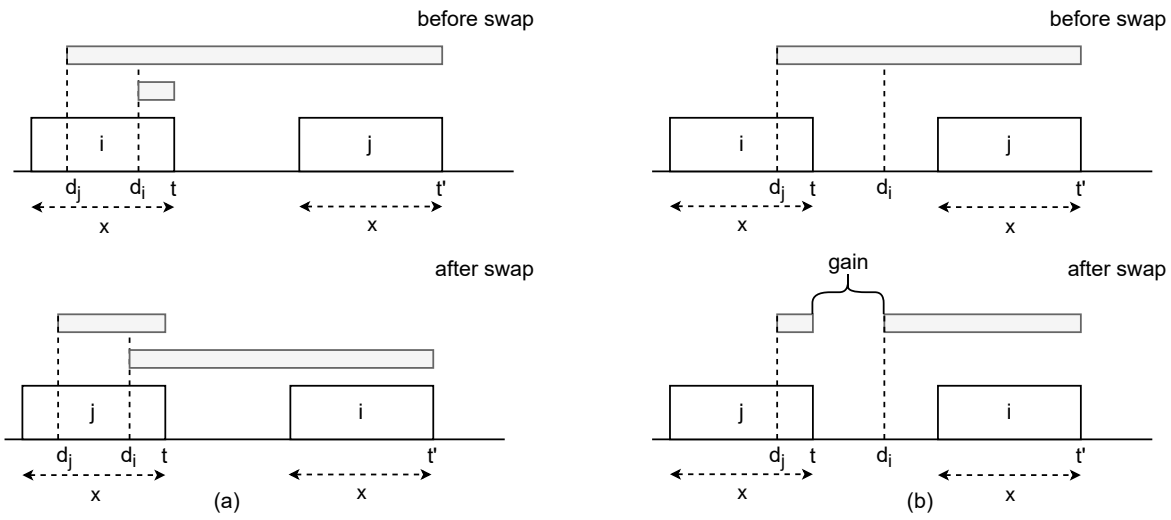

Figure 7.4: (a) Both job i and job j have non-negative tardiness values (b) Job i has no tardiness, but job j incurs tardiness

7.6 Formulation and Implementation

A formulation of the problem is presented below, which will be used to construct initial solutions to the problem. Then, the formulation is slightly modified and used for solving problems involving subsets of tasks in an effort to attain better schedules overall.

Let $\mathbb{J}$ be the set of jobs.

Let P_j be the duration of each job $j \in \mathbb{J}$.

Let D_j be the due date of each job $j \in \mathbb{J}$.

Let T be the number of time points. Note that the value of T is not given explicitly by the problem, but such a value can be computed by aggregating the duration of all jobs.

Let $Cap(t)$ be the capacity of the machine at each time point $t \in 0..T-1$.

We define integer decision variables $s_j \in 0..T-1-P_j$ which denote the start time of each job $j \in \mathbb{J}$.

Likewise, integer decision variables $f_j \in P_j..T-1$ are defined which denote the finish time of each job $j \in \mathbb{J}$.

We also define integer decision variables $z_j \geq 0$ which denote the tardiness of each job $j \in \mathbb{J}$.

Finally, the binary decision variables x_{jt} and y_{jt} are defined. Each one of the former

variables assumes value 1 if job j starts its execution at time point t or else it assumes value 0. Likewise, each y_{jt} variable marks the time point that job j finishes.

$$min \sum_{j \in \mathbb{J}} z_j \tag{7.1}$$

$$f_j = s_j + P_j \quad \forall j \in \mathbb{J} \tag{7.2}$$

$$z_j \geq f_j - D_j \quad \forall j \in \mathbb{J} \tag{7.3}$$

$$s_j \geq t * x_{jt} \quad \forall j \in \mathbb{J} \quad \forall t \in 0..T-1 \tag{7.4}$$

$$s_j \leq t + (M - t) * (1 - x_{jt}) \quad \forall j \in \mathbb{J} \quad \forall t \in 0..T-1 \tag{7.5}$$

$$\sum_{t \in 0..T-1} x_{jt} = 1 \quad \forall j \in \mathbb{J} \tag{7.6}$$

$$y_{jt+P_j} = x_{jt} \quad \forall j \in \mathbb{J} \quad \forall t \in 0..T-1-P_j \tag{7.7}$$

$$\sum_{j \in \mathbb{J}} \sum_{t' \in 0..t} x_{jt'} - \sum_{j \in \mathbb{J}} \sum_{t' \in 0..t} y_{jt'} \leq Cap(t) \quad \forall t \in 0..T-1 \tag{7.8}$$

A brief explanation of the above model follows.

The aim of the objective function in Equation 7.1 is to minimize the total tardiness of all jobs.

Equation 7.2 assigns the proper finish time value for each job given its start time and duration.

Equation 7.3 assigns tardiness values to jobs. In particular, when job j finishes before its due time, the right side of the inequality is a negative number, and variable z_j assumes the value 0 since its domain is of nonnegative integers. When job j finishes after its due time, z_j becomes $f_j - D_j$. This occurs because z_j is included in the minimized objective function and therefore forced to assume the smallest possible nonnegative value.

Equations 7.4 and 7.5 drive variables x_{jt} to proper values based on s_j values. This

occurs because when s_j assumes value t, then x_{jt} becomes 1. It should be noted that M in Equation 7.5 represents a big value, and $T - 1$ can be used for it. For the specific time point t that a job will be scheduled to begin, the right sides of both equalities will assume value t. For all other time points besides t, the right sides of the former and the latter equations become 0 and M, respectively.

Equation 7.6 enforces that only one among all x_{jt} variables of each job j will assume value 1.

Equation 7.7 dictates the following association rule. For each job j, when x_{jt} becomes 1 or 0, then the corresponding y variable of j having time offset P_j, which is y_{jt+P_j}, will also be 1 or 0 respectively.

Equation 7.8 guarantees that for each time point, the capacity of the machine will not be violated. The values that the left side of the equation assumes are the numbers of active jobs at each time point t. The first double summation counts the jobs that have started no later than t, while the second double summation counts the jobs that have also finished no later than t. Their difference is obviously the number of active jobs.

7.6.1 Constraint programming formulation

The IBM ILOG Constraint Programming (CP) Solver seems to be a good choice for solving scheduling problems involving jobs that occupy intervals of time and consume some types of resources that have time-varying availability Laborie et al. (2018) [63]. The one-machine scheduling problem can be easily formulated in IBM ILOG CP solver using one fixed size interval variable per job (`j`), and the constraint `always_in` that restricts all of them to assume values that collectively never exceed the maximum available capacity through time. This is possible by using a pulse cumulative function expression that represents the contribution of the fixed interval variables over time. Each job execution requires one capacity unit, which is occupied when the job starts, retained through its execution and released when the job finishes. In this case, variable `usage` aggregates all pulse requirements by all jobs. The objective function uses a set of integer variables `z[job.id]` that are stored in a dictionary having as keys the identifier of each job. Each `z[job.id]` variable assumes the value of the job's tardiness (i.e. the non negative difference of the job's due time (`job.due_time`) from its finish time (`end_of(j[job.id])`). An additional constraint is added that corresponds to the due time rule mentioned in Section 7.5. Jobs are grouped by duration, and a list ordered by due times is prepared for each group. Then, for all jobs in a list, the constraint enforces that the order of the jobs must be respected. This means that each job in a list should have earlier start time

than the start time of the job that follows it in the list. The model implementation using IBM ILOG CP solver's python API is presented below.

```
import docplex.cp.model as cpx

model = cpx.CpoModel()
x_ub = int(problem.ideal_duration() * 1.1)
j = {
    job.id: model.interval_var(
        start=[0, x_ub - job.duration - 1],
        end=[job.duration, x_ub - 1],
        size=job.duration
    )
    for job in problem.jobs
}

z = {
    job.id: model.integer_var(lb=0, ub=x_ub - 1)
    for job in problem.jobs
    }

usage = sum([model.pulse(j[job.id], 1) for job in problem.jobs])

for i in range(problem.nint):
    model.add(
        model.always_in(
            usage,
            [problem.capacities[i].start, problem.capacities[i].end],
            0,
            problem.capacities[i].capacity,
        )
    )

for job in problem.jobs:
    model.add(z[job.id] >= model.end_of(j[job.id]) - job.due_time)

for k in problem.size_jobs: # iterate over discrete job durations
    jobs_by_due_time = same_duration_jobs[k]
    for i in range(len(jobs_by_due_time)-1):
        j1, j2 = jobs_by_due_time[i][1], jobs_by_due_time[i+1][1]
        model.add(model.start_of(z[j1]) <= model.start_of(z[j2]))

model.minimize(sum([z[job.id] for job in problem.jobs]))
```

The object `problem` is supposed to be an instance of a class that has all relevant information for the problem instance under question (i.e. `jobs` is the list of all jobs,

each job besides `id` and `due_time` has also a `duration` property, `nint` is the number of capacity intervals, `capacities[i].start` and `capacities[i].end` are the start time and end time of the i^{th} capacity step, respectively). Finally, the problem object has the `ideal_duration` method that estimates a tight value for the makespan of the schedule, which is incremented by 10% to accommodate possible gaps that hinder the full exploitation of the available capacity. The "ideal duration" is computed by totaling the durations of all jobs and then filling the area under the capacity line from left to right and from bottom to top, with blocks of size 1×1 until the totaled durations quantity runs out. The rightmost point on the time axis of the filled area becomes the "ideal duration" and is clearly a relaxation of the actual completion time of the optimal solution since each job is decomposed in blocks of duration one and no gaps appear in the filled area.

An effort was undertaken to implement the above model using Google's ORTools CP-SAT solver. This solver has the cumulative constraint that can be used in place of `always_in` to describe the machine's varying capacity. A series of *FixedSizeIntervalVar* variables were used that transformed the pulse of the capacity to a flat line equal to the maximum capacity. Unfortunately, the solver under this specific model implementation could not approximate good results and was finally not used.

7.7 Local search improvement procedures

We have identified three local search procedures that have the potential to improve the cost of a given schedule. These local search procedures can be considered as "large" moves since they examine a significant number of neighboring schedules to the current one.

7.7.1 Local search Improve1

The first local search procedure starts by iterating over all jobs. For each job $j1$, each other consecutive job $j2$ is identified, and then each job $j3$ with a duration equal to the aggregated durations of $j1$ and $j2$ is found. Since $j1$ and $j2$ are consecutive, they can be swapped with job $j3$, and the schedule will still remain feasible, as seen in Figure 7.5. Moreover, the order of the two first jobs does not influence the feasibility. So, two alternatives are tested that compare the imposed penalties before and after the swap, and if an improvement is found, the swap occurs. The time complexity of this procedure is $O(|\mathbb{J}|)$ since the maximum number of consecutive jobs for each job is bounded by the maximum capacity, which is a constant number much smaller than the number of jobs. Moreover, identifying consecutive jobs and jobs of durations that are equal to the aggregated duration of two other consecutive jobs is performed

using Hash Maps that effectively contribute $O(1)$ to the above complexity. The first one uses times as keys and has a list of jobs starting at these times as values. By using as key the finish time of a job $j1$, the dictionary returns each job $j2$ that is consecutive to $j1$. The second Hash Map uses the jobs' durations as keys and value for each key x, the list of jobs with duration x. Note that the second Hash Map is computed only once and remains unchanged through the solution process.

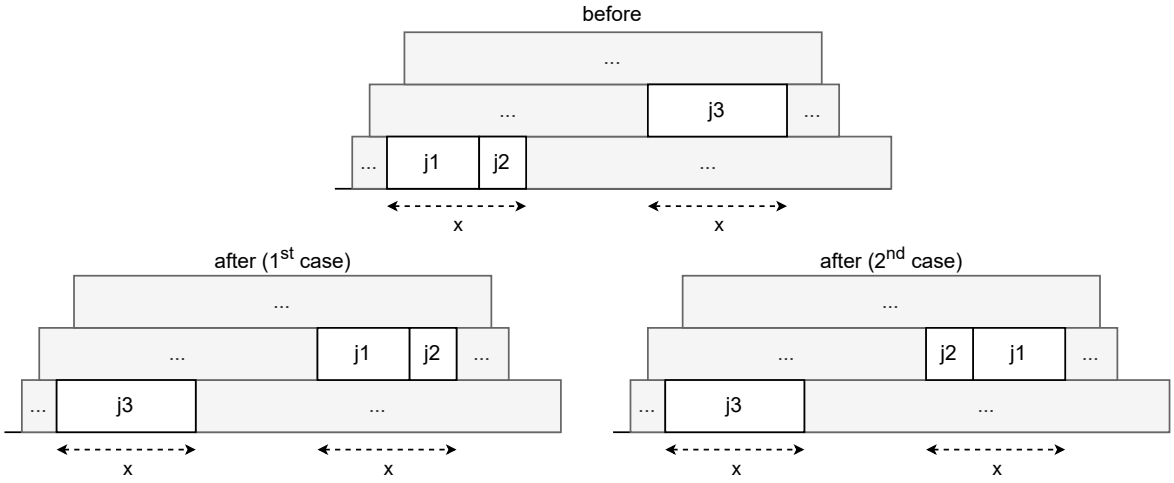

Figure 7.5: Assuming that job $j3$ has the same duration as the aggregated duration of jobs $j1$ and $j2$, two cases for swapping them become possible. The first one puts $j1$ first and $j2$ second and the other one puts $j2$ first and $j1$ second.

7.7.2 Local search Improve2

The second local search procedure uses C-Paths that are computed as described in subsection 7.3.1. Each C-Path is traversed from left to right until a job j is found that imposes cost to the schedule (i.e., has finish time greater than its due time). The only way that the penalty of a job j can be reduced is by moving it to the left side of the C-Path. So, all jobs that start earlier than job j are examined by swapping places with job j. If the total penalty imposed by job j and a sequence of jobs up to another job k is greater than the penalty after swapping jobs j and k, followed by shifts of jobs in between, then this set of moves occurs. The time complexity of this procedure is $O(|\mathbb{J}|^2)$. Since each C-Path has length that is $O(|\mathbb{J}|)$, and each C-Path is traversed once for identifying jobs with penalties, and then for each such job the C-path is again traversed, it follows that the complexity is quadratic. The construction of each C-Path costs $O(|\mathbb{J}|)$, which is added to the time of the above procedure and gives $O(|\mathbb{J}|) + O(|\mathbb{J}|^2)$. Since this occurs for every job, $|\mathbb{J}|$ C-Paths are generated, and this results in a total complexity of $O(|\mathbb{J}|^3)$ for the second local search procedure.

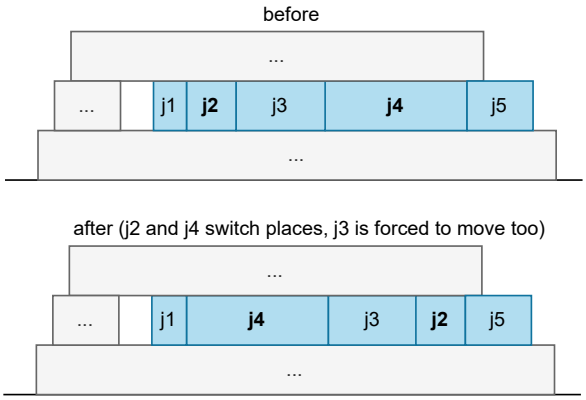

Figure 7.6: Given a C-Path, this local search procedure swaps two non-consecutive jobs ($j1$ and $j2$), and appropriately shifts the in-between jobs ($j3$) so as to keep the C-Path property for all involved jobs.

7.7.3 Local search Improve3

The third local search procedure starts by identifying periods where the capacity is not fully used. Given a capacity profile that has the form of a pulse, for each job, the pulse is lowered by one unit for the period that it is active. This is iterated for all jobs, and finally, it is possible to exist periods scattered across the horizon that have non-zero capacity remaining. So, jobs with finish times that fall inside these periods (gaps) can possibly be moved to the right, and the schedule should still be feasible. The main idea of this local search procedure is that it allows two jobs of marginally different durations to swap places. This occurs by firstly identifying two C-Paths, with no common jobs, that have as rightmost jobs, jobs with finish times falling inside gaps. Given two such C-Paths, each job of them can be swapped with a job of the other C-Path, provided that the slack that the gap provides is adequate for this move. This means that all jobs of a C-Path that are to the right of the smaller between the two swapping jobs should shift to the right, and all jobs of the other C-Path that are to the right of the bigger job should shift to the left, giving the opportunity of further penalty gains. In principle, the number of jobs that might have finish times that fall inside gaps is $O(|\mathbb{J}|)$, but the experiments showed that in practice, this number is a small fraction of $|\mathbb{J}|$. Since two C-Paths are involved, and each job of a C-Path has to be checked with each job of the other C-Path, this contributes $O(|\mathbb{J}|^2)$. Moreover, all possible pairs of jobs that fall in gaps are used as starting points in the construction of the corresponding C-Paths, resulting in another $O(|\mathbb{J}|^2)$ term. So, the time complexity of the overall procedure is $O(|\mathbb{J}|^4)$. It should be noted that shifts due to penalty reductions occur rarely, and their amortized contribution to the time complexity is neglected. In practice, the time needed for this move is comparable to the previous one due to the relatively small number of jobs that fall inside gaps.

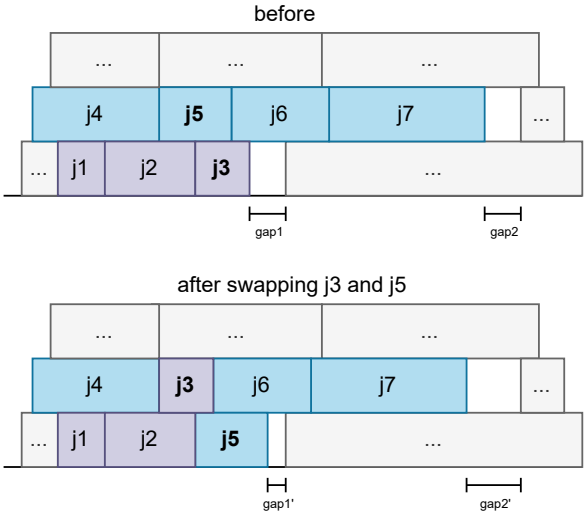

Figure 7.7: Jobs belonging to two C-Paths swap places to reduce the length or even remove gaps in the schedule.

7.8 A multi-staged approach

The approach employed for addressing the problem uses several stages that operate cyclically until the available time runs out.

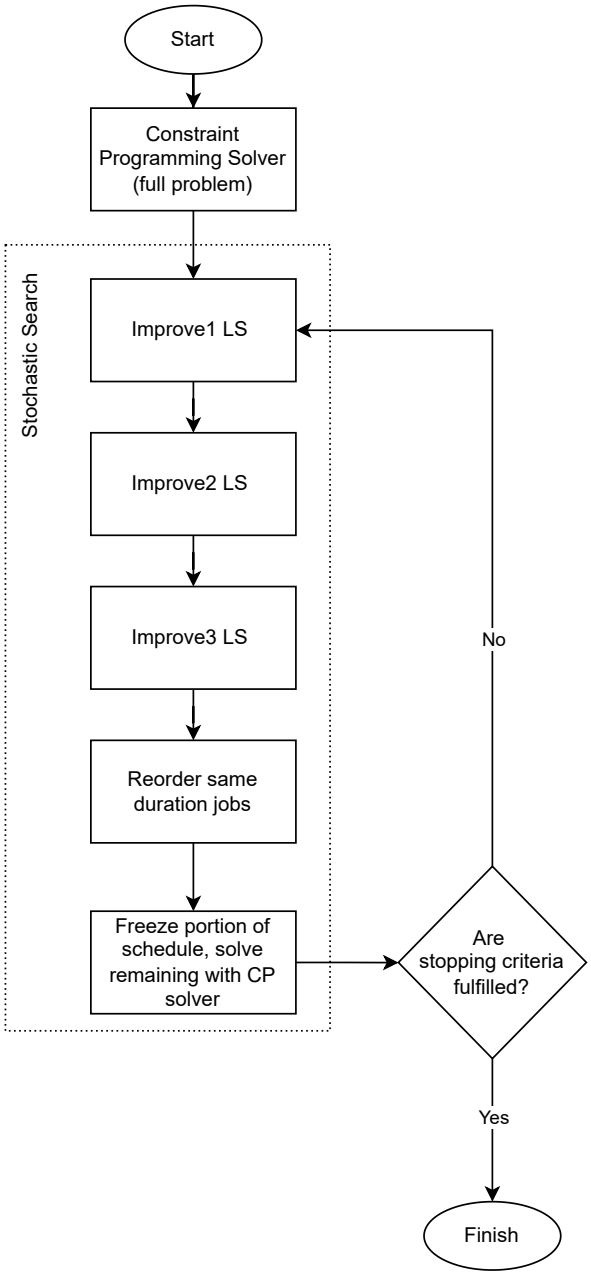

Figure 7.8: Hybrid Exact-Local Search approach.

7.9 Results

The experiments were run on a workstation with 32GB of RAM, and an Intel Core i7-7700K 4.2GHz CPU (4 cores, 8 threads), running Windows 10. The constraint programming solver IBM ILOG CP Optimizer Version 22 was used. The local search procedures, the implementation of the constraint programming model, and the driver program were all implemented in Python. The results are compared with the results of Mencía et al. in (2021) [3], which is a continuation of their previous works in (2017) [134] and (2019) [111]. In their most recent work they present and compare

six memetic algorithms termed MA_{SCP}, MA_{iSCP}, MA_{SCP+}, MA_{CB}, MA_{ICP} and MA_{HYB}. The last one gives the best results among all others and the previous approaches of the authors, and this is the algorithm this approach is compared with. MA_{HYB} combines CB and ICP procedures under a memetic algorithm. Both procedures use the concept of a cover. A cover is a disjoint set of C-Paths that covers all jobs. In CB once a cover is generated, C-Paths of the cover are examined in isolation for improvements. On the other hand, ICP swaps jobs between C-Paths, again using a cover to select the C-Paths participating in the procedure. In [3], no values of schedule costs are given, but the relative performance among the six approaches is recorded in tables and graphs instead. So, results about the actual schedule costs of MA_{HYB} and the other approaches that are used in the comparisons hereafter were taken from github repo[2], that authors cite in their paper.

7.9.1 CPO vs. CPO+

I call the constraint programming approach, briefly described in [3], CPO (Constraint Programming Optimizer), and the approach described in subsection 7.6.1 that exploits the "due rule", CPO+. Results about the performance of CPO were taken from the web repository cited at the end of the previous paragraph.

Table 7.4 depicts for each problem instance the total tardiness of all jobs for the schedule that CPO and CPO+ produced. It shows that CPO+ manages to find equal to the best-known solutions for 25 out of 190 problem instances, while CPO achieves this for only 2 problem instances (`i120_3_3` and `i120_5_4`). Best values are written in bold. An allotted time of $n/2$ seconds for each problem instance was given for each run, where n is the number of jobs. All available cores were used, which was the default setup for the IBM ILOG CP Optimizer. The results of CPO+ are the best among 10 runs for each problem instance, and random seeds were used to achieve diversity.

[2]https://github.com/raulmencia/One-Machine-Scheduling-with-Time-Dependent-Capacity-via-Efficient-Memetic-Algorithms

Problem	CPO										CPO+									
Set	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
i120_3	951	3834	**2410**	1059	3951	3258	730	4095	1690	1326	**867**	**3620**	**2410**	**1019**	**3859**	**3123**	**720**	**4084**	**1663**	**1268**
i120_5	1054	1673	999	**496**	3128	458	550	1227	3486	1245	**1030**	**1538**	**959**	**496**	**3062**	**455**	**519**	**1214**	**3332**	**1151**
i120_7	1201	2768	3798	4206	3220	2786	1712	3306	4694	546	**1133**	**2725**	**3551**	**4083**	**3095**	**2665**	**1655**	**3225**	**4510**	**503**
i120_10	764	1212	1554	935	1612	1157	2530	777	1056	881	**752**	**1129**	**1511**	**887**	**1467**	**1118**	**2464**	**721**	**1006**	**823**
i250_10	4311	552	1480	4871	6921	6563	4857	6108	5453	1405	**4144**	**506**	**1354**	**4755**	**6522**	**6288**	**4604**	**5940**	**5363**	**1320**
i250_20	5905	1974	2971	2774	1343	1786	1644	2408	1730	6632	**5674**	**1899**	**2825**	**2545**	**1097**	**1631**	**1594**	**2202**	**1561**	**6563**
i250_30	3381	3348	5273	5099	4560	5333	3990	803	1808	5600	**3085**	**3114**	**4999**	**4194**	**4304**	**5129**	**3726**	**695**	**1561**	**5418**
i500_10	4791	865	1109	2226	675	6312	3261	4688	497	2227	**4614**	**822**	**963**	**2159**	**610**	**6100**	**2864**	**4472**	**465**	**2130**
i500_20	8212	899	9440	1495	1859	9349	626	3536	6614	6030	**7705**	**790**	**9144**	**1273**	**1799**	**9232**	**525**	**3252**	**6420**	**5886**
i500_30	3064	613	6317	4787	6583	1838	4256	9360	1099	482	**2822**	**501**	**6048**	**4425**	**6114**	**1697**	**3892**	**9051**	**921**	**369**
i750_10	4670	8244	6051	5405	1655	8227	1039	8638	13594	4149	**4460**	**7820**	**5841**	**5110**	**1539**	**7909**	**970**	**8042**	**13103**	**3906**
i750_20	6100	889	1593	10151	9927	12396	5175	11910	4812	2866	**6046**	**703**	**1320**	**9633**	**9179**	**11652**	**4875**	**11414**	**4399**	**2690**
i750_30	5092	6045	3200	3240	733	3028	1243	3281	3118	2256	**4954**	**5314**	**2534**	**3030**	**687**	**2609**	**1114**	**2963**	**2812**	**2089**
i750_50	5989	5766	13945	2699	11096	12590	5748	11425	5689	6139	**5303**	**5098**	**12615**	**2433**	**10190**	**11998**	**5122**	**10898**	**5402**	**5242**
i1000_10	712	24629	1071	15711	16299	2188	779	20902	23213	4471	**641**	**23821**	**833**	**15342**	**15443**	**2025**	**743**	**20891**	**22495**	**4147**
i1000_20	10379	17525	20318	8214	15211	25044	10012	17482	11583	11316	**9470**	**16355**	**20265**	**8083**	**14510**	**24915**	**9912**	**17048**	**10436**	**10892**
i1000_30	7871	2074	20964	13991	4308	13630	10763	2713	15856	20959	**7054**	**1769**	**18580**	**12753**	**3800**	**12298**	**10232**	**2239**	**15631**	**19138**
i1000_50	3315	16491	13047	1520	1630	18216	21181	2227	15473	4682	**2963**	**16236**	**12130**	**1239**	**1478**	**17131**	**20133**	**2022**	**14454**	**4199**
i1000_100	**73637**	**81104**	28292	39057	**76080**	50541	47746	55973	26585	17803	78963	81638	**26446**	**37719**	77298	**49313**	**45036**	**53639**	**24823**	**16330**

Table 7.4: Best results (total tardiness) from the CPO approach and the CPO+ approach.

7.9.2 Hybrid Exact-Local Search

The HELS (Hybrid Exact-Local Search) approach is shown using the flowchart of Figure 7.8. Firstly, the constraint programming solver is employed for the full problem. A $n/2$ seconds period of time is given for executing this stage. Then, for $3\times n$ seconds a loop occurs that includes the 3 local search procedures, followed by an activation of the constraint programming solver again, but this time for subproblems. These subproblems might involve jobs that intersect with vertical ribbons on the time axis or groups of consecutive sequences of jobs (i.e., multiple C-Paths). Note, that reordering of jobs might be needed so as the current solution to conform with the "due rule", else the constraint programming solver might consider the fixed parts of the partial solution as infeasible. This is denoted by the extra stage "Reorder same duration jobs" after the third local search procedure of Figure 7.8.

Table 7.5 presents the best results that were achieved by the approach with the best known results that are all provided by MA_{HYB}.

Problem	MA_{HYB}										HELS									
Set	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
i120_3	**848**	**3568**	**2410**	**1019**	**3858**	**3120**	**720**	**4084**	**1663**	**1268**	**848**	3570	**2410**	**1019**	**3858**	**3120**	**720**	**4084**	**1663**	**1268**
i120_5	**1030**	**1511**	**959**	**496**	**3061**	**455**	**519**	**1214**	**3223**	**1131**	**1030**	1514	**959**	**496**	**3061**	**455**	**519**	**1214**	3231	**1131**
i120_7	**1120**	**2725**	**3493**	**4021**	**3059**	**2640**	**1655**	**3225**	**4470**	**493**	1121	**2725**	3505	4028	3078	**2640**	**1655**	**3225**	4474	**493**
i120_10	**746**	**1124**	**1442**	**887**	**1447**	**1118**	**2463**	**721**	**977**	**820**	749	1125	1453	**887**	**1447**	**1118**	**2463**	**721**	983	**820**
i250_10	**4094**	**506**	**1349**	**4731**	**6390**	**6280**	**4497**	**5881**	**5321**	**1293**	4103	**506**	**1349**	**4731**	6391	6284	4511	5887	5327	**1293**
i250_20	**5573**	**1882**	**2813**	**2525**	**1054**	**1583**	**1565**	**2190**	**1553**	**6531**	**5573**	1888	**2813**	**2525**	**1054**	1605	1570	**2190**	**1553**	6541
i250_30	**3013**	**3054**	4758	4098	4197	5034	**3641**	**686**	**1502**	5197	**3013**	**3054**	**4753**	**4093**	**4194**	**5019**	**3641**	**686**	**1502**	**5193**
i500_10	**4614**	**822**	**951**	**2102**	**610**	**5981**	**2768**	**4460**	**462**	**1998**	**4614**	**822**	953	2116	**610**	**5981**	2783	**4460**	**462**	2008
i500_20	7649	**790**	**8941**	**1272**	**1719**	9110	**523**	**3180**	**6291**	5661	**7569**	**790**	8970	**1272**	1744	**9097**	**523**	3188	6338	**5659**
i500_30	**2670**	**477**	5975	**4307**	**5869**	**1614**	**3795**	8946	**862**	**352**	**2670**	**477**	**5974**	**4307**	5873	1626	**3795**	**8888**	**862**	**352**
i750_10	4379	**7744**	**5819**	5086	1517	**7895**	952	7996	12837	**3840**	**4312**	**7744**	5821	**5082**	**1500**	**7895**	**943**	**7962**	**12814**	**3840**
i750_20	**5891**	**700**	**1314**	9674	**9073**	**11434**	**4855**	11353	**4393**	**2632**	5979	**700**	**1314**	**9562**	**9073**	**11434**	**4855**	**11284**	**4393**	**2632**
i750_30	**4713**	**5128**	**2364**	2951	**661**	**2577**	**1070**	**2911**	**2700**	**1994**	4767	**5128**	**2364**	**2945**	663	**2577**	**1070**	2912	**2700**	**1994**
i750_50	**5134**	4978	12601	**2356**	**9840**	**11483**	**4868**	**10580**	**5154**	5204	**5134**	**4792**	**12147**	**2356**	9847	11500	**4868**	10583	**5154**	**5028**
i1000_10	**641**	23729	815	**15204**	15393	**2025**	735	20686	22358	4028	**641**	**23521**	**812**	15206	**15180**	**2025**	**729**	**20574**	**22162**	**3982**
i1000_20	**9440**	16069	20168	**7962**	**14183**	24693	9816	16994	**10290**	**10781**	**9440**	**15971**	**19986**	7975	**14183**	**24507**	**9726**	**16789**	10301	**10781**
i1000_30	6902	1687	18433	**12399**	3768	**12090**	9795	**2085**	15625	18728	**6780**	**1668**	**18087**	12411	**3707**	**12090**	**9765**	**2085**	**15528**	**18483**
i1000_50	**2883**	16418	**11613**	**1142**	**1390**	**16656**	**19566**	**1886**	**13740**	**3989**	**2883**	**15977**	11618	**1142**	**1390**	16667	**19566**	1889	13747	**3989**
i1000_100	71034	**75101**	25977	**36374**	67109	**47540**	44545	**52266**	23704	15664	**71012**	75181	**25594**	36376	**66419**	47541	**43437**	**52266**	**23690**	**15230**

Table 7.5: Best previously known results (total tardiness) achieved from the MA_{HYB} approach, and results of the HELS approach.

Table 7.6 consolidates the relative performance of this approach when compared with the best-known results. Someone can observe that the approach manages to find new best results for 48 of the problem instances. It also equals best-known results for 91 problem instances. MA_{HYB} achieves better results than this approach for the remaining 51 problem instances. I also compare the solutions with how close it reaches the previously best-known solution as a percentage value, and this metric is called "distance%". Negative values mean that this approach sets a new best-known value. The average distance% metric for all problem instances assumes a negative value of −0.1727%, demonstrating its real good performance. This is further supported by Figure 7.9, which shows for each problem subset, consisting of 10 instances, a boxplot of the distance% values. Again, negative values are advantageous for this approach, and the graph clearly shows that this approach achieves excellent results, especially for large problem instances.

Best	Equal	Worse	Distance (%)
48	91	51	-0.1727

Table 7.6: HELS approach performance over best recorded results.

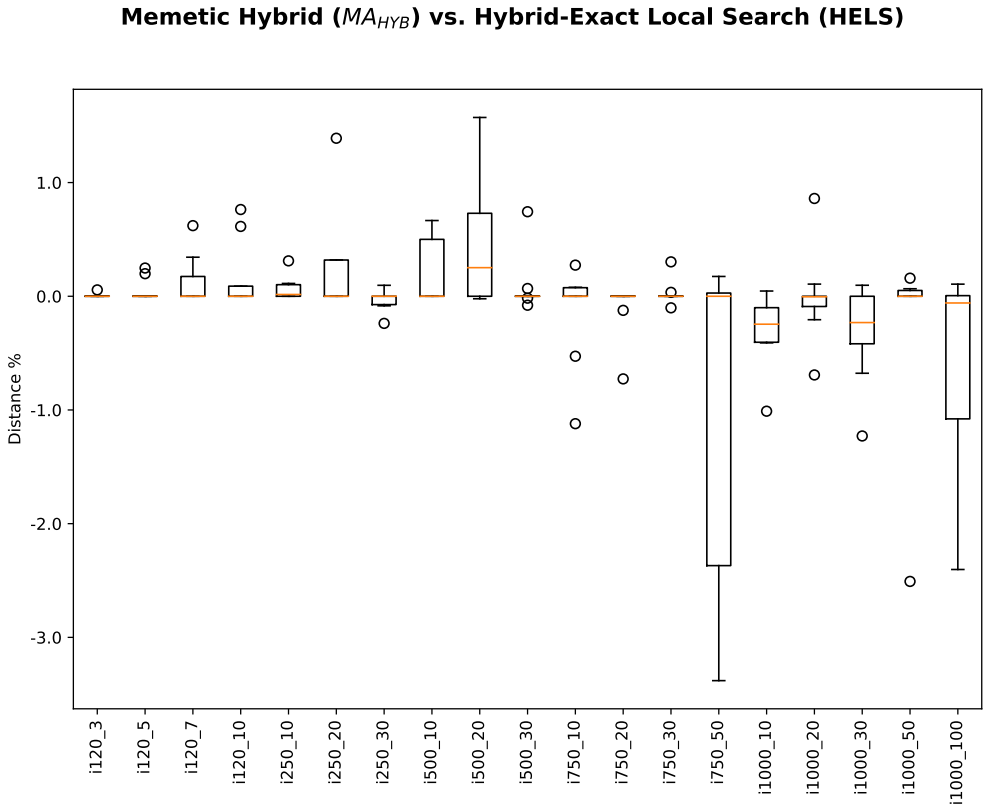

Figure 7.9: HELS approach compared with best known results derived from the MA_{HYB} approach in [3].

Figure 7.10 includes three graphs showing the cost of solutions during the allotted

execution time. Costs start from very high values but sharply fall to smaller values, not very far from the final ones. The long tails of the graphs indicate that less time is needed for achieving good quality schedules than the $3.5 \times n$ seconds (n is the number of jobs) that was used in the experiments. This is further demonstrated by values of CPO+ and HELS, in Tables 7.4 and 7.5, with the first ones being close to the second. In particular, for the set of all 190 problem instances, the percentage distance of CPO+ to HELS has a mean equal to 0.077 and a standard deviation equal to 0.069.

Regarding comparing the approach with the one by Mencía et al. [3], a few remarks can be made. Firstly, Mencía et al. do not report in their papers the exact values that their approaches returned but instead compare their results to their other less efficient approaches. So I retrieved the values I use in the comparisons from the site github repo[3] that the authors reference in their paper. Providing exact values alongside solution files that other researchers can download from the repository github repo[4] may attract more interest to the problem. Since the run-time environment used in this case and in Mencía et al. case are different, the focus is mainly on whether each approach can find the best possible result, given that limited processing power is exploited in both cases. Furthermore, in this case, Figure 7.10 clearly shows a trend observed in other problem instances: this approach reaches results very close to the final results using only about 1/5 of the allotted execution time.

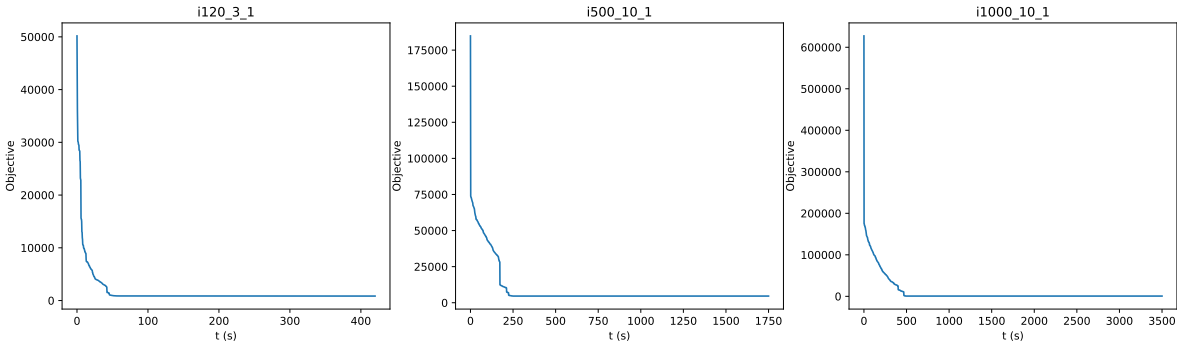

Figure 7.10: Cost values during the execution time, using the HELS approach.

The following paper was published based on this work:
C. Valouxis, C. Gogos, A. Dimitsas, P. Potikas, and A. Vittas, "A Hybrid Exact–Local Search Approach for One-Machine Scheduling with Time-Dependent Capacity," *Algorithms*, vol. 15, no. 12, p. 450, 2022

[3]https://github.com/raulmencia/One-Machine-Scheduling-with-Time-Dependent-Capacity-via-Efficient-Memetic-Algorithms
[4]https://github.com/chgogos/1MSTDC

Chapter 8

Conclusions

8.1 Research contributions

This thesis made contributions to the field of combinatorial optimization, particularly in the context of scheduling and timetabling problems. The key contributions include:

- Symmetry Identification and Elimination: A systematic approach was developed to identify and eliminate symmetries in various optimization problems, leading to reduced search spaces and improved solver performance.
- Modeling Innovations: New mathematical and constraint programming models were proposed for several real-world problems, including examination timetabling, course scheduling, thesis defense scheduling, sports scheduling, and one-machine scheduling with time-dependent capacity.
- Hybrid Approaches: The thesis introduced hybrid methods combining exact algorithms (e.g., CP, MIP) with local search and metaheuristics, achieving state-of-the-art results in multiple benchmark datasets.
- Quantum-Ready Formulations: QUBO models were developed and tested on quantum annealers, demonstrating the feasibility of solving real-world scheduling problems using emerging quantum technologies.

8.2 Results

The proposed methods were evaluated across five major case studies:

- The first proven optimal instance in a public available dataset from 1997 to the UETP.
- A sports scheduling approach able to generate solutions for double round robin tournaments in an exact way.
- Optimal solutions for most of the instances in two datasets of the TDTT problem.
- 48 new best known solutions for a dataset for the One-Machine Scheduling with Time-Dependent Capacity problem.

8.3 Future research directions

Several promising avenues for future work were identified:

- Automated Symmetry Detection: Developing tools that can automatically detect and exploit symmetries in arbitrary optimization models.
- Quantum Optimization: Further exploration of QUBO formulations and their deployment on quantum annealers and hybrid quantum-classical solvers.
- Generalized Scheduling Frameworks: Creating reusable, modular frameworks that can adapt to various scheduling domains with minimal customization.
- Real-Time and Dynamic Scheduling: Extending current models to handle dynamic environments where inputs and constraints evolve over time.
- Explainability and Fairness: Incorporating fairness metrics and explainable decision-making into optimization models, especially in educational and public-sector applications.

List of publications

Uncapacitated Examination Timetabling Problem

C. Gogos, A. Dimitsas, V. Nastos, and C. Valouxis, "Some insights about the Uncapacitated Examination Timetabling Problem," in *2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, Sep. 2021, pp. 1–7

A. Dimitsas, C. Gogos, C. Valouxis, V. Nastos, and P. Alefragis, "A proven optimal result for a benchmark instance of the uncapacitated examination timetabling problem," *Journal of Scheduling*, Mar. 2024

A. Dimitsas, P. Alefragis, C. Valouxis, and C. Gogos, "An unconstrained binary model for the Uncapacitated Examination Timetabling Problem," in *PATAT Conference 2024 proceedings of the 14th International Conference of the Practice and Theory of Automated Timetabling*, 2024

Post Enrollment Course Timetabling Problem

A. Dimitsas, V. Nastos, C. Valouxis, and C. Gogos, "A mathematical formulation for constructing feasible solutions for the Post Enrollment Course Timetabling Problem," in *2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE, 2022, pp. 1–5

A. Dimitsas, V. Nastos, C. Gogos, and C. Valouxis, "An exact based approach for the Post Enrollment Course Timetabling Problem," in *Proceedings of the 26th Pan-Hellenic Conference on Informatics*. Athens Greece: ACM, Nov. 2022, pp. 77–82

Thesis Defense Timetabling Problem

A. Dimitsas, C. Gogos, and E. Pappa, "Better solutions for the Thesis Defense Timetabling problem using a three-phase approach," in *Proceedings of the 26th Pan-Hellenic Conference on Informatics*. Athens Greece: ACM, Nov. 2022, pp. 58–63

A. Dimitsas and C. Gogos, "Finding Near Optimal Solutions to the Thesis Defense Timetabling Problem by Exploiting Symmetries," *Operations Research Forum*, vol. 5, no. 3, p. 65, Jul. 2024

Sports Scheduling

A. Dimitsas, C. Gogos, C. Valouxis, A. Tzallas, and P. Alefragis, "A pragmatic approach for solving the sports scheduling problem," in *Proc. 13th Int. Conf. Pract. Theory Autom. Timetabling, PATAT*, vol. 3, 2022, pp. 195–207

D. Van Bulck, D. Goossens, J.-P. Clarner, A. Dimitsas, G. H. Fonseca, C. Lamas-Fernandez, A. E. Phillips, and R. M. Rosati, "What algorithm to select to create your sports schedule?" in *MathSport International 2023*, 2023, pp. 48–48

D. Van Bulck, D. Goossens, J.-P. Clarner, A. Dimitsas, G. H. Fonseca, C. Lamas-Fernandez, M. M. Lester, J. Pedersen, A. E. Phillips, and R. M. Rosati, "Which algorithm to select in sports timetabling?" *European Journal of Operational Research*, vol. 318, no. 2, pp. 575–591, 2024

One-Machine Scheduling with Time-Dependent Capacity

C. Valouxis, C. Gogos, A. Dimitsas, P. Potikas, and A. Vittas, "A Hybrid Exact–Local Search Approach for One-Machine Scheduling with Time-Dependent Capacity," *Algorithms*, vol. 15, no. 12, p. 450, 2022

Various

C. Gogos, A. Dimitsas, C. Valouxis, and P. Alefragis, "Modeling a balanced commute educational timetabling problem in the context of teaching integer programming," in *2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE, 2022, pp. 1–5

E. Hytis, V. Nastos, C. Gogos, and A. Dimitsas, "Automated identification of fraudulent financial statements by analyzing data traces," in *2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE, 2022, pp. 1–7

Bibliography

[1] A. Dimitsas, C. Gogos, and E. Pappa, "Better solutions for the Thesis Defense Timetabling problem using a three-phase approach," in *Proceedings of the 26th Pan-Hellenic Conference on Informatics*. Athens Greece: ACM, Nov. 2022, pp. 58–63.

[2] M. Battistutta, S. Ceschia, F. De Cesco, L. Di Gaspero, and A. Schaerf, "Modelling and solving the thesis defense timetabling problem," vol. 70, pp. 1–12, 2019.

[3] R. Mencía and C. Mencía, "One-Machine Scheduling with Time-Dependent Capacity via Efficient Memetic Algorithms," *Mathematics*, vol. 9, no. 23, p. 3030, Jan. 2021.

[4] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[5] S. P. Adam, S.-A. N. Alexandropoulos, P. M. Pardalos, and M. N. Vrahatis, "No Free Lunch Theorem: A Review," in *Approximation and Optimization*, ser. Springer Optimization and Its Applications, I. C. Demetriou and P. M. Pardalos, Eds. Springer, December 2019, pp. 57–82.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[7] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Cham: Springer International Publishing, 2022.

[8] G. B. Dantzig, *Linear Programming and Extensions*. Princeton University Press, 1963.

[9] L. Khachiyan, "Polynomial algorithm in linear programming," *Sov Math Dokl*, vol. 20, Feb. 1979.

[10] M. Kojima, S. Mizuno, and A. Yoshise, "A primal-dual interior point algorithm for linear programming," *Progress in Mathematical Programming: Interior Point and Related Methods*, pp. 29–47, 1989.

[11] V. Chvátal, *Linear Programming*. W. H. Freeman and Company, 1983.

[12] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*. Athena Scientific, 1997.

[13] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.

[14] L. A. Wolsey, *Integer Programming*. Wiley-Interscience, 1998.

[15] T. Achterberg, "Scip: Solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009.

[16] A. Lodi, "Mixed integer programming computation," *In Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[17] M. Conforti, G. Cornuéjols, and G. Zambelli, *Integer Programming*. Springer, 2014.

[18] D. Bertsimas and R. Weismantel, *Optimization over Integers*. Dynamic Ideas, 2005.

[19] L. Liberti, "Symmetry in mathematical programming," in *Mixed Integer Nonlinear Programming*, J. Lee and S. Leyffer, Eds. New York, NY: Springer New York, 2012, pp. 263–283.

[20] F. Margot, "Symmetry in Integer Linear Programming," in *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, Eds. Berlin, Heidelberg: Springer, 2010, pp. 647–686.

[21] M. E. Pfetsch and T. Rehn, "A computational comparison of symmetry handling methods for mixed integer programs," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 37–93, Mar. 2019.

[22] V. Kaibel and M. E. Pfetsch, "Packing and partitioning orbitopes," *Mathematical Programming*, vol. 114, no. 1, pp. 1–36, 2008.

[23] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio, "Orbital branching," *Mathematical Programming*, vol. 126, no. 1, pp. 147–178, 2011.

[24] F. Margot, "Symmetry in integer linear programming," in *50 Years of Integer Programming 1958-2008*, M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, Eds. Berlin, Heidelberg: Springer, 2010, pp. 647–686.

[25] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006.

[26] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[27] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[28] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*. Elsevier, 2006.

[29] A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, vol. 8, no. 1, pp. 99–118, 1977.

[30] P. van Hentenryck, *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.

[31] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "Minizinc: Towards a standard cp modelling language," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2007, pp. 529–543.

[32] I. P. Gent and B. M. Smith, "Symmetry breaking in constraint programming," in *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000)*, Berlin, Germany, 2000, pp. 599–603.

[33] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, 1999.

[34] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, 1951, pp. 481–492.

[35] K. Sörensen, "Metaheuristics – the metaphor exposed," *International Transactions of Operations Research*, vol. In Press, Feb. 2013.

[36] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[37] F. Glover and G. A. Kochenberger, Eds., *Handbook of Metaheuristics*. Norwell, MA: Kluwer Academic Publishers, 2003.

[38] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. Chichester, UK: Wiley, 2009.

[39] C. Morris and C. Segura, "Symmetry breaking in evolutionary algorithms," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14)*. ACM, 2014, pp. 839–846.

[40] A. Lucas, "Ising formulations of many np problems," *Frontiers in Physics*, vol. 2, p. 5, 2014.

[41] T. Albash and D. A. Lidar, "Adiabatic quantum computation is equivalent to standard quantum computation," *Reviews of Modern Physics*, vol. 90, no. 1, p. 015002, 2018.

[42] F. Glover, G. Kochenberger, and Y. Du, "A tutorial on formulating and using qubo models," *arXiv preprint arXiv:1811.11538*, 2018.

[43] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang, "The unconstrained binary quadratic programming problem: a survey," *Journal of Combinatorial Optimization*, vol. 28, no. 1, pp. 58–81, 2014.

[44] F. Glover, G. Kochenberger, R. Hennig, and Y. Du, "Quantum bridge analytics I: A tutorial on formulating and using QUBO models," *Annals of Operations Research*, vol. 314, no. 1, pp. 141–183, Jul. 2022.

[45] T. Stollenwerk and A. Basermann, "Experiences with Scheduling Problems on Adiabatic Quantum Computers," in *Proceedings of the 1st International Workshop on Post-Moore Era Supercomputing*, 2016.

[46] K. Ikeda, Y. Nakamura, and T. S. Humble, "Application of Quantum Annealing to Nurse Scheduling Problem," *Scientific Reports*, vol. 9, no. 1, p. 12837, Sep. 2019.

[47] J. Ossorio-Castillo and F. Pena-Brage, "Optimization of a refinery scheduling process with column generation and a quantum annealer," *Optimization and Engineering*, vol. 23, no. 3, pp. 1471–1488, Sep. 2022.

[48] C.-Y. Huang, C.-N. Lee, and M.-T. Tsai, "Job Shop-Scheduling Based on Quantum Annealing," in *Proceedings of the 2023 International Conference on Intelligent Computing and Its Emerging Applications*, ser. ICEA '23. New York, NY, USA: Association for Computing Machinery, Dec. 2024, pp. 113–114.

[49] A. Bertuzzi, D. Ferrari, A. Manzalini, and M. Amoretti, "Evaluation of Quantum and Hybrid Solvers for Combinatorial Optimization," in *21th ACM International Conference on Computing Frontiers*, Mar. 2024.

[50] S. A. Cook, "The complexity of theorem-proving procedures," *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, pp. 151–158, 1971.

[51] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability*. Amsterdam: IOS Press, 2009.

[52] J. Marques-Silva and K. A. Sakallah, "Conflict-driven clause learning sat solvers," *Handbook of Satisfiability*, vol. 185, pp. 131–153, 2009.

[53] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing (SAT)*. Springer, 2003, pp. 502–518.

[54] G. Audemard and L. Simon, "On the glucose sat solver," *International Journal on Artificial Intelligence Tools*, vol. 27, no. 01, p. 1840001, 2018.

[55] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," *Handbook of Satisfiability*, vol. 185, pp. 825–885, 2009.

[56] M. W. Carter, G. Laporte, and S. Y. Lee, "Examination Timetabling: Algorithmic Strategies and Applications," *Journal of the Operational Research Society*, vol. 47, no. 3, pp. 373–383, Mar. 1996.

[57] E. Burke and J. Newall, "A multistage evolutionary algorithm for the timetable problem," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 1, pp. 63–74, Apr. 1999.

[58] K. Socha, J. Knowles, and M. Sampels, "A MAX-MIN ant system for the university course timetabling problem," in *Ant Algorithms*, ser. Ant Algorithms, vol. 1–13, Oct. 2002.

[59] L. Di Gaspero, "Tabu search techniques for examination timetabling," Feb. 2001.

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

[61] C. Gogos, A. Dimitsas, V. Nastos, and C. Valouxis, "Some insights about the Uncapacitated Examination Timetabling Problem," in *2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, Sep. 2021, pp. 1–7.

[62] R. Bellio, S. Ceschia, L. Di Gaspero, and A. Schaerf, "Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling," *Computers & Operations Research*, vol. 132, p. 105300, Aug. 2021.

[63] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím, "IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG," *Constraints : an international journal*, vol. 23, Mar. 2018.

[64] L. Perron and V. Furnon, "OR-tools," Google.

[65] A. Dimitsas, C. Gogos, C. Valouxis, V. Nastos, and P. Alefragis, "A proven optimal result for a benchmark instance of the uncapacitated examination timetabling problem," *Journal of Scheduling*, Mar. 2024.

[66] A. Dimitsas, P. Alefragis, C. Valouxis, and C. Gogos, "An unconstrained binary model for the Uncapacitated Examination Timetabling Problem," in *PATAT Conference 2024 proceedings of the 14th International Conference of the Practice and Theory of Automated Timetabling*, 2024.

[67] L. Di Gaspero, B. Mccollum, and A. Schaerf, "The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3)," Jan. 2007.

[68] T. Müller, "ITC2007 solver description: A hybrid approach," *Annals of Operations Research*, vol. 172, no. 1, pp. 429–446, Nov. 2009.

[69] Z. Lü and J.-K. Hao, "Adaptive Tabu Search for course timetabling," *European Journal of Operational Research*, vol. 200, no. 1, pp. 235–244, Jan. 2010.

[70] G. Lach and M. Lübbecke, "Curriculum based course timetabling: New solutions to Udine benchmark instances," *Annals of Operations Research*, vol. 194, pp. 255–272, Apr. 2012.

[71] S. L. Goh, K. , Graham, and N. R. and Sabar, "Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem," *Journal of the Operational Research Society*, vol. 70, no. 6, pp. 873–888, Jun. 2019.

[72] S. L. Goh, G. Kendall, N. Sabar, and S. Abdullah, "An effective hybrid local search approach for the post enrolment course timetabling problem," *OPSEARCH*, vol. 57, Jun. 2020.

[73] Y. Nagata, "Random partial neighborhood search for the post-enrollment course timetabling problem," *Computers & Operations Research*, vol. 90, pp. 84–96, Feb. 2018.

[74] S. Ceschia, L. Di Gaspero, and A. Schaerf, "Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment

course timetabling problem," *Computers & Operations Research*, vol. 39, no. 7, pp. 1615–1624, Jul. 2012.

[75] H. Cambazard, E. Hebrard, B. O'Sullivan, and A. Papadopoulos, "Local search and constraint programming for the post enrolment-based course timetabling problem," *Annals of Operations Research*, vol. 194, pp. 111–135, Apr. 2012.

[76] R. Lewis and J. Thompson, "Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem," *European Journal of Operational Research*, vol. 240, pp. 637–648, Feb. 2015.

[77] C. Valouxis, C. Gogos, P. Alefragis, and E. Housos, "Decomposing the high school timetable problem," *Practice and Theory of Automated Timetabling (PATAT 2012), Son, Norway*, vol. 61, 2012.

[78] R. Qu, E. Burke, B. Mccollum, L. Merlot, and S. Lee, "A survey of search methodologies and automated system development for examination timetabling," *J. Scheduling*, vol. 12, pp. 55–89, Feb. 2009.

[79] K. A. Dowsland and J. M. Thompson, "Simulated Annealing," in *Handbook of Natural Computing*, G. Rozenberg, T. Bäck, and J. N. Kok, Eds. Berlin, Heidelberg: Springer, 2012, pp. 1623–1655.

[80] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[81] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024.

[82] A. Dimitsas, V. Nastos, C. Valouxis, and C. Gogos, "A mathematical formulation for constructing feasible solutions for the Post Enrollment Course Timetabling Problem," in *2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE, 2022, pp. 1–5.

[83] A. Dimitsas, V. Nastos, C. Gogos, and C. Valouxis, "An exact based approach for the Post Enrollment Course Timetabling Problem," in *Proceedings of the 26th Pan-Hellenic Conference on Informatics*. Athens Greece: ACM, Nov. 2022, pp. 77–82.

[84] S. Limanto, N. Benarkah, and T. Adelia, "Thesis examination timetabling using genetic algorithm," in *2018 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC)*, Oct. 2018, pp. 6–10.

[85] T. T. B. Huynh, Q. D. Pham, and D. D. Pham, "Genetic algorithm for solving the master thesis timetabling problem with multiple objectives," in *2012*

Conference on Technologies and Applications of Artificial Intelligence, 2012, pp. 74–79.

[86] P. Su, B. Luo, F. Deng, A. Xia, and Y. Guo, "Group strategy of dissertation defense based on greedy retrospective hybrid algorithm," *Journal of Physics: Conference Series*, vol. 1634, p. 012077.

[87] M. I. Tawakkal and Suyanto, "Exploration-exploitation balanced krill herd algorithm for thesis examination timetabling," in *2020 International Conference on Data Science and Its Applications (ICoDSA)*, 2020, pp. 1–5.

[88] J. Almeida, D. Santos, J. R. Figueira, and A. P. Francisco, "A multi-objective mixed integer linear programming model for thesis defence scheduling," *European Journal of Operational Research*, vol. 312, no. 1, pp. 92–116, 2024.

[89] T. Stidsen, D. Pisinger, and D. Vigo, "Scheduling euro-k conferences," *European Journal of Operational Research*, vol. 270, no. 3, pp. 1138–1147, 2018.

[90] A. Dimitsas and C. Gogos, "Finding Near Optimal Solutions to the Thesis Defense Timetabling Problem by Exploiting Symmetries," *Operations Research Forum*, vol. 5, no. 3, p. 65, Jul. 2024.

[91] E. Lambrechts, A. Ficker, D. Goossens, and F. Spieksma, "Round-robin tournaments generated by the circle method have maximum carry-over," *Mathematical Programming*, vol. 172, Feb. 2017.

[92] J. Chen and D. Dong, "Research on the general method of round robin scheduling," in *Advances in Multimedia, Software Engineering and Computing Vol.2*, D. Jin and S. Lin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 393–399.

[93] D. Van Bulck and D. Goossens, "The international timetabling competition on sports timetabling (ITC2021)," *European Journal of Operational Research*, vol. 308, no. 3, pp. 1249–1267, 2023.

[94] D. Goossens and F. Spieksma, "Scheduling the belgian soccer league," *Interfaces*, vol. 39, pp. 109–118, Apr. 2009.

[95] C. Ribeiro and S. Urrutia, "Scheduling the brazilian soccer tournament: Solution approach and practice," *Interfaces*, vol. 42, pp. 260–272, Jun. 2012.

[96] K. Nurmi, D. Goossens, and J. Kyngäs, "Scheduling a triple round robin tournament with minitournaments fo. The Finnish national youth ice hockey league," *Journal of the Operational Research Society*, vol. 65, Nov. 2014.

[97] F. Alarcon, G. Duran, M. Guajardo, J. Miranda, H. Muñoz, L. Ramírez, M. Ramírez, D. Sauré, M. Siebert, S. Souyris, A. Weintraub, R. Wolf-Yadlin, and G. Zamorano, "Operations research transforms the scheduling of chilean soccer leagues and south american world cup qualifiers: 2016 franz edelman award finalists," *Interfaces*, vol. 47, Jan. 2017.

[98] G. Duran, M. Guajardo, and D. Sauré, "Scheduling the south american qualifiers to the 2018 FIFA world cup by integer programming," *European Journal of Operational Research*, vol. 262, Apr. 2017.

[99] R. Lewis and J. Thompson, "On the application of graph colouring techniques in round-robin sports scheduling," *Computers & Operations Research*, vol. 38, no. 1, pp. 190–204, 2011.

[100] T. Januario, S. Urrutia, C. C. Ribeiro, and D. de Werra, "Edge coloring: A natural model for sports scheduling," *European Journal of Operational Research*, vol. 254, no. 1, pp. 1–8, 2016.

[101] F. N. Costa, S. Urrutia, and C. C. Ribeiro, "An ILS heuristic for the traveling tournament problem with predefined venues," *Annals of Operations Research*, vol. 194, no. 1, pp. 137–150, Apr. 2012.

[102] T. Januario and S. Urrutia, "A new neighborhood structure for round robin scheduling problems," *Computers & Operations Research*, vol. 70, pp. 127–139, 2016.

[103] M. A. Trick, "A schedule-then-break approach to sports timetabling," in *Practice and Theory of Automated Timetabling III*, E. Burke and W. Erben, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 242–253.

[104] C. Ribeiro, "Sports scheduling: Problems and applications," *International Transactions in Operational Research*, vol. 19, pp. 201–226, Jan. 2012.

[105] R. Miyashiro and T. Matsui, "Round-robin tournaments with a small number of breaks," Oct. 2003.

[106] D. Briskorn, "Feasibility of home–away-pattern sets for round robin tournaments," *Operations Research Letters*, vol. 36, no. 3, pp. 283–284, 2008.

[107] A. Dimitsas, C. Gogos, C. Valouxis, A. Tzallas, and P. Alefragis, "A pragmatic approach for solving the sports scheduling problem," in *Proc. 13th Int. Conf. Pract. Theory Autom. Timetabling, PATAT*, vol. 3, 2022, pp. 195–207.

[108] D. Van Bulck, D. Goossens, J.-P. Clarner, A. Dimitsas, G. H. Fonseca, C. Lamas-Fernandez, A. E. Phillips, and R. M. Rosati, "What algorithm to

select to create your sports schedule?" in *MathSport International 2023*, 2023, pp. 48–48.

[109] D. Van Bulck, D. Goossens, J.-P. Clarner, A. Dimitsas, G. H. Fonseca, C. Lamas-Fernandez, M. M. Lester, J. Pedersen, A. E. Phillips, and R. M. Rosati, "Which algorithm to select in sports timetabling?" *European Journal of Operational Research*, vol. 318, no. 2, pp. 575–591, 2024.

[110] C. Koulamas, "The single-machine total tardiness scheduling problem: Review and extensions," *European Journal of Operational Research*, vol. 202, no. 1, pp. 1–7, Apr. 2010.

[111] C. Mencía, M. R. Sierra, R. Mencía, and R. Varela, "Evolutionary one-machine scheduling in the context of electric vehicles charging," *Integrated Computer-Aided Engineering*, vol. 26, no. 1, pp. 49–63, Feb. 2019.

[112] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, "Complexity of Machine Scheduling Problems," in *Annals of Discrete Mathematics*, ser. Studies in Integer Programming, P. L. Hammer, E. L. Johnson, B. H. Korte, and G. L. Nemhauser, Eds. Elsevier, Jan. 1977, vol. 1, pp. 343–362.

[113] S. K. Gupta and J. Kyparisis, "Single machine scheduling research," *Omega*, vol. 15, no. 3, pp. 207–227, Jan. 1987.

[114] P. Alefragis, K. Plakas, I. Karampinis, C. Valouxis, M. Birbas, A. Birbas, and C. Gogos, "Sustainable energy aware industrial production scheduling," in *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling*, 2022, pp. 256–264.

[115] C. Gogos, C. Valouxis, P. Alefragis, G. Goulas, N. Voros, and E. Housos, "Scheduling independent tasks on heterogeneous processors using heuristics and Column Pricing," *Future Generation Computer Systems*, vol. 60, pp. 48–66, Jul. 2016.

[116] P. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-Based Scheduling*, ser. International Series in Operations Research & Management Science, F. S. Hillier, Ed. Boston, MA: Springer US, 2001, vol. 39.

[117] C. Valouxis, C. Gogos, P. Alefragis, and N. Voros, "Constraint Programming Modeling for the Task Scheduling Problem with Data Storage at MPSoCs," 2018.

[118] C. Gogos, "Solving the Distributed Permutation Flow-Shop Scheduling Problem Using Constrained Programming," *Applied Sciences*, vol. 13, no. 23, p. 12562, 2023.

[119] P. Großmann, S. Hölldobler, N. Manthey, K. Nachtigall, J. Opitz, and P. Steinke, "Solving Periodic Event Scheduling Problems with SAT," in *Advanced Research in Applied Artificial Intelligence*, H. Jiang, W. Ding, M. Ali, and X. Wu, Eds. Berlin, Heidelberg: Springer, 2012, pp. 166–175.

[120] C. Ansótegui, M. Bofill, M. Palah'i, J. Suy, and M. Villaret, "Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem," ser. SARA 2011 - Proceedings of the 9th Symposium on Abstraction, Reformulation, and Approximation, Jan. 2011, pp. 2–9.

[121] P. Brucker, S. Knust, A. Schoo, and O. Thiele, "A branch and bound algorithm for the resource-constrained project scheduling problem1," *European Journal of Operational Research*, vol. 107, no. 2, pp. 272–288, Jun. 1998.

[122] R. R. Vaessens, "Generalized job shop scheduling : Complexity and local search," 1995.

[123] H. Matsuo, C. Juck SUH, and R. S. Sullivan, "A controlled search simulated annealing method for the single machine weighted tardiness problem," *Annals of Operations Research*, vol. 21, no. 1, pp. 85–108, Dec. 1989.

[124] K.-M. Lee, T. Yamakawa, and K.-M. Lee, "A genetic algorithm for general machine scheduling problems," in *1998 Second International Conference. Knowledge-Based Intelligent Electronic Systems. Proceedings KES'98 (Cat. No.98EX111)*, vol. 2, Apr. 1998, pp. 60–66 vol.2.

[125] F. J. Gil-Gala, C. Mencía, M. R. Sierra, and R. Varela, "Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity over time," *Applied Soft Computing*, vol. 85, p. 105782, Dec. 2019.

[126] P. M. França, A. Mendes, and P. Moscato, "A memetic algorithm for the total tardiness single machine scheduling problem," *European Journal of Operational Research*, vol. 132, no. 1, pp. 224–242, Jul. 2001.

[127] X. Wu and A. Che, "A memetic differential evolution algorithm for energy-efficient parallel machine scheduling," *Omega*, vol. 82, pp. 155–165, Jan. 2019.

[128] D. Merkle and M. Middendorf, "Ant Colony Optimization with Global Pheromone Evaluation for Scheduling a Single Machine," *Applied Intelligence*, vol. 18, no. 1, pp. 105–111, Jan. 2003.

[129] T.-L. Lin, S.-J. Horng, T.-W. Kao, Y.-H. Chen, R.-S. Run, R.-J. Chen, J.-L. Lai, and I.-H. Kuo, "An efficient job-shop scheduling algorithm based on particle

swarm optimization," *Expert Systems with Applications*, vol. 37, no. 3, pp. 2629–2636, Mar. 2010.

[130] B. Yuce, F. Fruggiero, M. S. Packianather, D. T. Pham, E. Mastrocinque, A. Lambiase, and M. Fera, "Hybrid Genetic Bees Algorithm applied to single machine scheduling with earliness and tardiness penalties," *Computers & Industrial Engineering*, vol. 113, pp. 842–858, Nov. 2017.

[131] F. Gil-Gala, M. Sierra, C. Mencía, and R. Arias, "Combining hyper-heuristics to evolve ensembles of priority rules for on-line scheduling," *Natural Computing*, vol. 21, Jun. 2020.

[132] C. Gogos, C. Valouxis, P. Alefragis, and A. Birbas, "Industrial chocolate production as an optimization problem," 2023.

[133] P. Alefragis, C. Gogos, C. Valouxis, M. Birbas, and A. Birbas, "Industrial production scheduling in the energy deregulation era," 2024.

[134] C. Mencía, M. Sierra, R. Mencía, and R. Arias, "Genetic algorithm for scheduling charging times of electric vehicles subject to time dependent power availability," May 2017, pp. 160–169.

[135] A. Hernández-Arauzo, J. Puente Peinador, R. Arias, and J. Sedano, "Electric vehicle charging under power and balance constraints as dynamic scheduling," *Computers & Industrial Engineering*, vol. 85, Apr. 2015.

[136] C. Valouxis, C. Gogos, A. Dimitsas, P. Potikas, and A. Vittas, "A Hybrid Exact–Local Search Approach for One-Machine Scheduling with Time-Dependent Capacity," *Algorithms*, vol. 15, no. 12, p. 450, 2022.

[137] C. Gogos, A. Dimitsas, C. Valouxis, and P. Alefragis, "Modeling a balanced commute educational timetabling problem in the context of teaching integer programming," in *2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE, 2022, pp. 1–5.

[138] E. Hytis, V. Nastos, C. Gogos, and A. Dimitsas, "Automated identification of fraudulent financial statements by analyzing data traces," in *2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE, 2022, pp. 1–7.