International Journal of Mathematics Trends and Technology
ISSN: 2231-5373/ https://doi.org/10.14445/22315373/IJMTT-V72I10P104

Volume 72 Issue 10, 45-52, October 2026
© 2026 Seventh Sense Research Group®

Original Article

Mixed-Integer Programming Optimisation for University Course Timetabling Problem

Jared Ongaro[1], Benjamin Kikwai[2]

[1]*Department of Mathematics, University of Nairobi, Nairobi, Kenya.*
[2]*Department of Mathematics and Statistics, Machakos University, Machakos, Kenya.*

[1]*Corresponding Author : ongaro@uonbi.ac.ke*

Received: 21 August 2026 Revised: 27 September 2026 Accepted: 14 October 2026 Published: 28 October 2026

Abstract *- This paper presents a comprehensive solution to the University Course Timetabling Problem (UCTP) through the lens of a flexible Mixed-Integer Programming (MIP) model. The model is designed to generate conflict-free schedules by assigning courses to instructors, rooms, and time slots, while minimising a weighted penalty function that rigorously quantifies soft objectives related to student conflicts, inefficient room utilisation, and instructor preferences. A key contribution is the provision of a practical, end-to-end implementation guide using an entirely open-source Python stack (Pandas, PuLP, CBC). This demonstration outlines the complete workflow, from simple CSV data inputs to a verifiable, human-readable timetable. The full implementation code is made publicly available for reproducibility. We conclude by analysing the inherent computational scalability challenges of this exact MIP approach and propose hybrid solution methodologies as a promising direction for future research on large-scale, real-world instances.*

Keywords *- Combinatorial Optimisation, Course Scheduling, Curriculum-Based Timetabling, Mixed-Integer Programming, Open-Source Solvers, Python, University Timetabling.*

1. Introduction

The construction of a weekly academic timetable is a multi-criteria optimisation problem of considerable complexity, governed by a multitude of variables, preferences, and operational constraints [1]. Factors such as core curriculum sequencing, faculty availability, room capacities, and diverse student enrolment patterns must be concurrently satisfied. While smaller institutions may manage this allocation manually, such methods become rapidly intractable and unsustainable as the scale and complexity of academic programmes grow [2]. This often results in inconsistent, suboptimal schedules that fail to meet the needs of the university community effectively. Consequently, automated approaches are not merely a matter of convenience but are essential for generating effective timetables and minimising administrative overhead.

This challenge is formally recognised in the operations research literature as the University Course Timetabling Problem (UCTP), a well-studied combinatorial optimisation problem. Researchers have applied a variety of sophisticated techniques to the UCTP, including metaheuristics (such as Tabu Search and Genetic Algorithms) [3, 4] and constraint programming [5]. Mixed-Integer Programming (MIP), however, offers a distinct and powerful advantage: the ability to model the problem with mathematical precision and find provably optimal solutions [6]. To fully explore the practical strengths and intrinsic limitations of this formal method, this work concentrates exclusively on solving the UCTP as an integer programme. Our contributions in this paper are threefold. First, we present a detailed MIP formulation designed for real-world applicability. This model captures critical hard constraints (e.g., no resource conflicts) and effectively manages the trade-offs among soft objectives, such as minimising student timetable clashes and respecting staff teaching preferences, through a flexible weighted penalty system. Second, we provide a practical, detailed roadmap for implementing this model using an accessible, open-source technology stack, thereby lowering the barrier to entry for powerful optimisation techniques. Third, we offer a critical assessment of the MIP methodology itself, with a particular focus on its computational scalability and performance limitations when applied to realistic problem sizes.

2. Literature Review

The university timetabling problem has been the subject of extensive research, yielding a diverse range of solution methodologies. Pioneering work by de Werra [7] and Carter and Laporte [1] established the problem's foundational

 This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

complexities, with comprehensive surveys by Schaerf [8] and later Lewis [4] providing structured overviews of the field. Solution techniques can be broadly categorised into three paradigms: exact methods (such as Integer Programming), heuristic and metaheuristic methods (which employ approximation-based search strategies), and constraint programming (which focuses on satisfying a set of logical rules).

2.1. Exact Methods: A Focus on Mixed-Integer Programming

Integer Programming (IP) and Mixed-Integer Programming (MIP) are fundamental exact methods for the UCTP. These approaches uniquely offer the advantage of delivering a provably optimal solution or, at a minimum, a feasible solution with a quantifiable optimality gap. The inherent computational difficulty of timetabling was established early on by Even, Itai, and Shamir [9], who classified such problems as NP-complete, a challenge further examined by Cooper and Kingston [10]. Despite this complexity, researchers have successfully applied MIP to complex, real-world instances.

Daskalaki, Birbas, and Housos [6], for example, presented a detailed IP formulation for a Greek university, effectively managing hard constraints such as resource conflicts and room capacities. More recently, Phillips et al. [11] developed an IP model to generate new timetables with minimal deviation from an existing schedule. The versatility of MIP has also been demonstrated in related domains, including patient admission scheduling [12] and urban transport networks [13]. Ultimately, the primary strength of MIP is its capacity to provide a rigorous, structured, and verifiable solution to a well-defined problem instance.

2.2. Alternative Paradigms for Large-Scale Instances

While powerful, MIP-based exact methods face significant scalability challenges. As the size and complexity of a timetabling problem increase, the time required to find an optimal solution can become computationally prohibitive. In response, metaheuristic algorithms have emerged as a popular and effective alternative. These general-purpose search algorithms are designed to find high-quality, feasible solutions within a reasonable timeframe, albeit without a guarantee of global optimality. The surveys by Blum and Roli [3] and Lewis [4] offer an excellent overview of metaheuristics applied to timetabling, such as genetic algorithms, simulated annealing, and tabu search, which are adept at navigating vast, complex solution spaces.

Constraint Programming (CP), detailed by Tsang [5], is another paradigm for tackling timetabling problems. CP models a problem as a set of variables and logical constraints, employing specialised propagation techniques to prune the search space efficiently. It is particularly effective at handling problems defined by a large number of intricate or non-linear logical rules. As no single method is universally superior, hybridisation has become a critical area of research. For instance, Shaw, De Backer, and Prosser [14] explored integrating CP with local search, combining the former's strength in rapid feasibility checking with the latter's solution-improving capabilities. Similarly, hybrid approaches that use MIP to solve a core, simplified subproblem before applying metaheuristics for refinement represent a promising direction for large-scale, real-world timetabling.

3. Mathematical Model of the Timetabling Problem

3.1. Problem Definition and Components

The University Course Timetabling Problem (UCTP) can be defined as a multi-dimensional assignment problem. The primary objective is to find an optimal mapping from the set of courses C to a Cartesian product of the sets of available time slots T, rooms R, and instructors I. This mapping must be feasible, satisfying a collection of hard constraints, and optimal, minimising a cost function derived from a set of desirable soft constraints. The core components that define the structure of the problem instance are as follows:

- Courses (C): The set of distinct academic courses to be scheduled. Each course $c \in C$ is a fundamental unit of the assignment.
- Instructors (I): The set of instructors available to teach the courses. We assume a pre-assignment where each course c is linked to a specific instructor $i \in I$.
- Rooms (R): The set of physical locations where courses can be held. Each room $r \in R$ is characterised by its seating capacity, Cap_r.
- Time Slots (T): The set of discrete time intervals for scheduling. This set is a Cartesian product of the set of working days, D, and the set of daily periods, P. Thus, a time slot $t \in T$ corresponds to an ordered pair (d, p), where $d \in D$ and $p \in P$.
- Students (S): To manage the complexity of student enrolments, we consider cohorts of students as single entities. A student group, or cohort, $s \in S$, represents a collection of students with identical course enrolment patterns. This abstraction allows for the efficient management of student-related constraints.

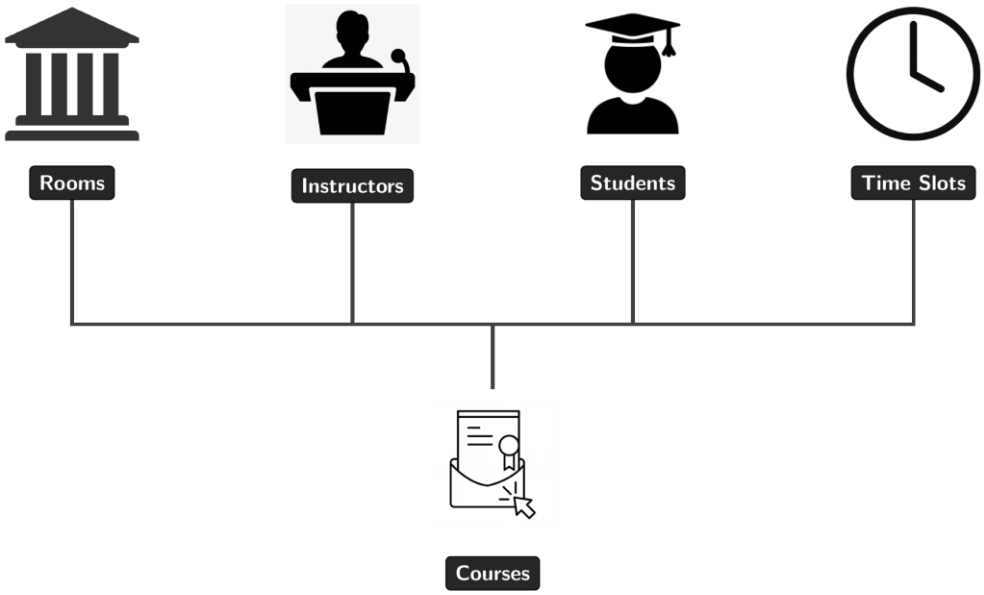


Fig. 1 A conceptual diagram of the timetabling optimisation problem, illustrating the core entities that must be assigned to courses

These entities and their associated properties form the basis of the mathematical model presented in the following subsection.

3.2. The Mixed-Integer Programming (MIP) Formulation

We now translate the problem definition into a formal MIP model, which consists of decision variables, an objective function, and a set of linear constraints [15, 16].

3.2.1. Sets and Indices

- c ∈ C: Index for courses.
- i ∈ I: Index for instructors.
- r ∈ R: Index for rooms.
- s ∈ S: Index for student groups.
- d ∈ D: Index for days.
- p ∈ P: Index for time slots within a day.

3.2.2. Parameters

- Size_c: The number of students enrolled in course *c*.
- Size_s: The total number of students in student group *s*.
- Cap_r: The seating capacity of room *r*.
- Teaches_ic: A binary parameter equal to 1 if instructor *i* teaches course *c*, and 0 otherwise.
- Enrolled_sc: A binary parameter equal to 1 if student group *s* is enrolled in course *c*, and 0 otherwise.
- I_special ⊂ I: A subset of instructors who prefer a compact teaching schedule.
- PrefCost_idp: A numerical penalty for assigning instructor *i* to day *d* and period *p*.
- W_capacity, W_days, W_clash, W_pref: Positive weights to control the trade-offs in the objective function.

3.2.3. Decision Variables

- X_cdrp ∈ {0, 1}: The primary binary variable; X_{cdrp} = 1 if course *c* is scheduled in room *r* on day *d* during period *p*, and 0 otherwise.
- Y_id ∈ {0, 1}: An auxiliary binary variable; Y_{id} = 1 if instructor *i* is required to teach on day *d*, and 0 otherwise.
- Clash_sdp ∈ $\mathbb{Z}^+$: A non-negative integer variable that counts the number of courses in excess of one assigned to student group *s* on day *d* at period *p*.

3.2.4. The Objective Function

The objective of the model is to minimise a total penalty score, Z, which is a weighted sum of four components representing different institutional goals. Each term in the function quantifies a specific type of 'cost' or undesirability associated with a potential timetable:

$$\min Z = W_{\text{capacity}} \sum_{c,d,r,p} X_{cdrp}\,(Cap_r - Size_c) + W_{\text{days}} \sum_{i \in I_{\text{special}}} \sum_{d \in D} Y_{id} + W_{\text{clash}} \sum_{s,d,p} C\,lash_{sdp} \cdot Size_s + W_{\text{pref}} \sum_{c,d,r,p} X_{cdrp} \left(\sum_{i \in I} T\,eaches_{ic} \cdot PrefCost_{idp} \right) \tag{1}$$

The first term penalises inefficient room usage by summing the 'wasted space'. The second penalises the total number of days that designated instructors must be on campus. The third term minimises the total number of students affected by scheduling conflicts. The final term penalises deviations from instructor preferences by summing the preference costs of each scheduled class.

3.2.5. Hard Constraints

These are inviolable rules that define a feasible timetable:

$$\sum_{d,r,p} X_{cdrp} = 1, \quad \forall c \in C \tag{2}$$

$$\sum_{c,r} (Teaches_{ic} \cdot X_{cdrp}) \leq 1, \quad \forall i \in I, d \in D, p \in P \tag{3}$$

$$\sum_{c} X_{cdrp} \leq 1, \quad \forall r \in R, d \in D, p \in P \tag{4}$$

$$Size_c \cdot X_{cdrp} \leq Cap_r, \quad \forall c \in C, d \in D, r \in R, p \in P \tag{5}$$

Constraint (2) ensures every course is scheduled exactly once. Constraints (3) and (4) are standard non-clash constraints, preventing an instructor or a room from being assigned to more than one course simultaneously. Constraint (5) enforces room capacity limits, ensuring that an assigned course's enrolment does not exceed the room's capacity.

3.2.6. Linking Constraints

These constraints enforce logical relationships between decision variables, primarily to correctly calculate the soft penalty costs:

$$\sum_{c,r} (Enrolled_{sc} \cdot X_{cdrp}) - 1 \leq Clash_{sdp}, \quad \forall s \in S, d \in D, p \in P \tag{6}$$

$$\sum_{c,r,p} (Teaches_{ic} \cdot X_{cdrp}) \leq |P| \cdot Y_{id}, \quad \forall i \in I_{\text{special}}, d \in D \tag{7}$$

Constraint (6) defines the student clash variable. If a student group s is assigned to k courses at a given time, the summation equals k, and Clash_sdp must be at least k - 1. Since the objective function minimises Clash_sdp, it will take on the exact value of k - 1 in an optimal solution. Constraint (7) is a 'big-M' constraint that links course assignments to instructor presence. Here, |P| (the number of periods in a day) serves as a valid, tight upper bound for the summation, ensuring Y_id is forced to 1 if instructor i has any classes on day d.

3.2.7. Variable Domains

Finally, the domains of the decision variables are formally defined:

$$X_{cdrp} \in \{0,1\}, \quad \forall c \in C, d \in D, r \in R, p \in P$$

$$Y_{id} \in \{0,1\}, \quad \forall i \in I, d \in D$$

$$Clash_{sdp} \in \mathbb{Z}^+, \quad \forall s \in S, d \in D, p \in P$$

4. Implementation with Open-Source Tools

While the mathematical formulation provides a complete and abstract representation of the timetabling problem, its practical value depends on its implementation. This section provides a guide for translating the MIP model into a functional software tool using exclusively open-source technologies. The goal is to demonstrate that a powerful optimisation system can be built without relying on expensive commercial licences, making it accessible to a wider range of institutions. We focus on a solution architecture based on a single, well-structured Python script.

4.1. The Proposed Technology Stack

The cornerstone of our implementation is the Python programming language, chosen for its simple syntax and rich ecosystem of packages for scientific computing. We leverage the Pandas library for efficient data manipulation from input files. To formulate and solve the mathematical model, we utilise PuLP, a popular open-source Python library that serves as a modeller [17]. PuLP allows a user to specify decision variables, constraints, and an objective function using a syntax that closely mirrors standard mathematical notation. It then acts as an interface to an underlying MIP solver. For this implementation, we propose using the COIN-OR Branch and Cut (CBC) solver, a robust and high-performance open-source MIP solver that is seamlessly integrated with PuLP.

4.2. Structuring Input Data

A well-defined data structure is critical for implementation. We recommend using Comma-Separated Values (CSV) files, as they are simple, human-readable, and can be easily edited with standard spreadsheet software. This approach decouples the data from the model's logic, allowing administrators to update timetabling requirements without modifying the source code. The following six CSV files are proposed to represent all necessary input parameters:

- courses.csv: Defines all courses to be scheduled, with columns: CourseID, InstructorID, and Size (student enrolment).
- instructors.csv: Provides details about instructors, with columns: InstructorID and IsSpecial (a binary flag for preference of a compact schedule).
- rooms.csv: Lists all available rooms, with columns: RoomID and Capacity.
- student_groups.csv: Defines student cohorts, with columns: GroupID and GroupSize.
- enrolments.csv: Defines the many-to-many relationship between students and courses, establishing the Enrolled_sc parameter. It contains the GroupID and CourseID columns.
- preferences.csv: Quantifies instructor preferences for teaching slots via the PrefCost_idp parameter. Columns are InstructorID, Day, TimeSlot, and Cost.

This modular file structure ensures that data is organised logically, is easy to manage, and can be loaded efficiently into Pandas DataFrames.

4.3. Solution Architecture and Workflow

The solution architecture is designed for simplicity and reproducibility, centred around a single Python script. The workflow is executed sequentially and can be broken down into four distinct phases.

4.3.1. Phase 1: Data Ingestion and Preprocessing

The process begins by loading the input data from the CSV files into Pandas DataFrames. This phase is critical for validating data and transforming it into Python data structures (primarily dictionaries and lists) that map directly to the MIP model's parameters.

```python
import pandas as pd
# Load data from CSV files
courses_df = pd.read_csv('courses.csv')
rooms_df = pd.read_csv('rooms.csv')
# ... load other files ...

# Create a parameter dictionary for room capacities
# e.g., {'R101': 50, 'R102': 30}
cap_r = dict(zip(rooms_df['RoomID'], rooms_df['Capacity']))
```

Listing 1. Loading CSV data and creating a parameter dictionary

4.3.2. Phase 2: MIP Model Formulation

With the data prepared, the script builds the MIP model programmatically using the PuLP library. This involves a direct translation of the mathematical formulation into Python code. The decision variables are created first.

```python
from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpBinary

# Initialise the model
model = LpProblem('UCTP', LpMinimize)

# Define sets from data
C = courses_df['CourseID'].tolist()
D = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']
R = rooms_df['RoomID'].tolist()
P = [1, 2, 3, 4]

# X_cdrp: 1 if course c is in room r on day d at period p
x = LpVariable.dicts('X', (C, D, R, P), cat=LpBinary)
```

Listing 2. Defining decision variables in PuLP

Following variable creation, the objective function and all constraints are added to the model object. For instance, the instructor non-clash constraint (Equation 3) is implemented by iterating through each instructor and time slot.

```python
# An instructor cannot teach more than one course at a time
for i in I:
    for d in D:
        for p in P:
            model += lpSum(x[c][d][r][p] * teaches_ic.get((i, c), 0)
                           for c in C for r in R) <= 1
```

Listing 3. Adding an instructor non-clash constraint to the model

4.3.3. Phase 3: Solving the Model

Once the model is fully formulated, a single command invokes the solver. PuLP passes the model to the underlying CBC solver, which employs optimisation algorithms such as branch-and-cut to find an optimal integer solution.

```python
# Solve the problem (uses CBC by default)
model.solve()
# Check the status (1 indicates Optimal)
if model.status == 1:
    print('Optimal solution found!')
```

Listing 4: Solving the formulated MIP model

4.3.4. Phase 4: Results Processing

If an optimal solution is found, the script iterates through the decision variables to identify which have a value of 1. Each such variable represents a successful assignment, which is then stored and formatted.

4.4. Code Repository and Accessibility

To ensure full reproducibility and encourage collaboration, the complete Python script, along with a set of sample CSV data files, is made publicly available in a GitHub repository. This allows other researchers and practitioners to download, inspect, and run the model. The repository can be accessed at: https://github.com/kikwaib/uctp-mip.git

4.5. Generating the Final Timetable

The final step is to transform the solver's raw output into a human-readable format. The script collects all assignments (where X_cdrp = 1) into a Pandas DataFrame. This structured format is ideal for analysis and can be exported to a CSV file named timetable_solution.csv, containing columns for Day, TimeSlot, CourseID, InstructorID, and RoomID. This format is universally accessible and can serve as a data source for generating polished PDF reports or web dashboards.

5. Discussion: Limitations of the MIP Approach

While the MIP model provides a powerful and rigorous framework for finding high-quality, provably optimal timetables, it is essential to acknowledge its inherent limitations. These challenges primarily relate to computational complexity and the practicalities of solving large-scale, real-world instances.

5.1. The Challenge of Computational Complexity and Scalability

The primary limitation of any MIP-based approach to the UCTP is its computational complexity. The problem belongs to the class of NP-hard problems [9, 10]. This implies that the time required to find a guaranteed optimal solution can grow exponentially with the size of the problem. The scale of this challenge becomes apparent when considering the number of binary variables. The main decision variable, X_cdrp, has a size of |C| × |D| × |R| × |P|. For a medium-sized department with 100 courses, 5 days, 4 time slots per day, and 15 rooms, the number of these binary variables would be 100 × 5 × 4 × 15 = 30,000. The theoretical search space is therefore on the order of 2^{30,000}, a number of astronomical proportions.

Although modern MIP solvers use sophisticated algorithms like the branch-and-cut method to prune this search space intelligently, the combinatorial explosion remains a fundamental barrier. For large universities, the model can become computationally intractable, meaning a solver may be unable to find an optimal solution -- or even a good feasible one -- within a practical timeframe. This trade-off between solution quality (optimality) and computation time is the central challenge of using exact methods for the UCTP.

5.2. Practical Considerations for Large and Complex Problems

The scalability challenges of MIP necessitate a pragmatic approach for institutions dealing with very large problem instances. In such scenarios, the quest for a mathematically perfect solution may be less important than the ability to generate a high-quality, feasible timetable quickly. This is where alternative solution paradigms, particularly metaheuristics, become highly relevant. As surveyed by Lewis [4] and Blum and Roli [3], methods like genetic algorithms and tabu search are designed to navigate vast solution spaces efficiently. They sacrifice the guarantee of optimality in exchange for speed and flexibility, making them well-suited for finding an initial, ‘good-enough’ solution that can be manually refined.

Furthermore, the utility of a perfectly optimal solution depends on the quality of the input data. In a real-world setting, data is rarely perfect; student enrolment numbers fluctuate, and instructor preferences may be qualitative or subject to change [2]. An MIP model will produce a solution that is optimal only with respect to the given data. If the data is imperfect, the resulting ‘optimal’ timetable may not be truly optimal in practice. Consequently, a fast heuristic that can quickly generate multiple good scenarios may be more valuable for decision-making than a slow, exact method tied to a single, rigid dataset. The choice between MIP and other methods is therefore not merely technical but strategic.

6. Conclusion

6.1. Summary of Work

This paper has presented a comprehensive treatment of the University Course Timetabling Problem using Mixed-Integer Programming. We developed a flexible and detailed MIP model capable of balancing multiple, often competing, institutional objectives -- including student conflict minimisation, instructor satisfaction, and efficient resource utilisation -- through a weighted, single-objective function. This approach provides administrators with the ability to tune the model according to their specific priorities.

Furthermore, we demonstrated a practical and accessible implementation path for this model. By leveraging Python and open-source libraries such as Pandas and PuLP, we outlined a complete workflow from data ingestion to final timetable generation. This implementation strategy underscores that powerful optimisation techniques need not be the exclusive domain of commercial software. We have shown that a robust, verifiable, and high-quality timetabling solution can be achieved with freely available tools.

6.2. Avenues for Further Research

Despite the strengths of the MIP approach, its computational limitations open several promising avenues for further research. One of the most practical directions is the development of hybrid solution methodologies. For instance, a two-stage approach could use a metaheuristic to quickly find an initial, high-quality feasible solution, which could then be used to warm-start an MIP solver, significantly reducing the search space and solution time. Alternatively, decomposition techniques, such as Benders or Dantzig-Wolfe decomposition (column generation), could be explored to break the large monolithic model into smaller, more manageable subproblems.

Another area for future work lies in enhancing the model’s fidelity to real-world complexities. The current model could be extended to include more nuanced constraints, such as minimising travel time for students and instructors between consecutive classes or ensuring fairness in the allocation of ‘prime’ teaching slots. Additionally, developing a user-friendly graphical interface would make the tool accessible to non-technical administrators, allowing them to adjust parameters, run the model, and visualise results without interacting directly with the code. Finally, a comparative study benchmarking this MIP model

against various state-of-the-art metaheuristics on a standardised set of UCTP instances would provide valuable insights into the trade-offs between solution quality and computational effort.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

References

[1] Michael W. Carter, and Gilbert Laporte, "Recent Developments in Practical Course Timetabling," *International Conference on the Practice and Theory of Automated Timetabling*, Brno, Czech Republic, vol. 1408, pp. 3-19, 1998. [CrossRef] [Google Scholar] [Publisher Link]

[2] Anthony Wren, "Scheduling, Timetabling and Rostering -- A Special Relationship?," *International Conference on the Practice and Theory of Automated Timetabling*, vol. 1153, pp. 46-75, 1996. [CrossRef] [Google Scholar] [Publisher Link]

[3] Christian Blum, and Andrea Roli, "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268-308, 2003. [CrossRef] [Google Scholar] [Publisher Link]

[4] Rhydian Lewis, "A Survey of Metaheuristic-based Techniques for University Timetabling Problems," *OR Spectrum*, vol. 30, no. 1, pp. 167-190, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[5] Edward Tsang, *Foundations of Constraint Satisfaction*, *Computation in Cognitive Science*, London, UK: Academic Press, 1993. [CrossRef] [Google Scholar] [Publisher Link]

[6] Sophia Daskalaki, Theodore Birbas, and Efthymios C. Housos, "An Integer Programming Formulation for a Case Study in University Timetabling," *European Journal of Operational Research*, vol. 153, no. 1, pp. 117-135, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[7] Dominique de Werra, "An Introduction to Timetabling," *European Journal of Operational Research*, vol. 19, no. 2, pp. 151-162, 1985. [CrossRef] [Google Scholar] [Publisher Link]

[8] Andrea Schaerf, "A Survey of Automated Timetabling," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 87-127, 1999. [CrossRef] [Google Scholar] [Publisher Link]

[9] Shimon Even, Alon Itai, and Adi Shamir, "On the Complexity of Timetable and Multi-Commodity Flow Problems," *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, USA, pp. 184-193, 1975. [CrossRef] [Google Scholar] [Publisher Link]

[10] Tim B. Cooper, and Jeffrey H. Kingston, "The Complexity of Timetable Construction Problems," *Practice and Theory of Automated Timetabling*, vol. 1153, pp. 281-295, 1996. [CrossRef] [Google Scholar] [Publisher Link]

[11] Antony E. Phillips et al., "Integer Programming for Minimal Perturbation Problems in University Course Timetabling," *Annals of Operations Research*, vol. 252, no. 2, pp. 283-304, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[12] Leonardo S.L. Bastos et al., "A Mixed Integer Programming Approach to the Patient Admission Scheduling Problem," *European Journal of Operational Research*, vol. 273, no. 3, pp. 831-840, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[13] James C. Chu, "Mixed-Integer Programming Model and Branch-and-Price-and-Cut Algorithm for Urban Bus Network Design and Timetabling," *Transportation Research Part B: Methodological*, vol. 108, pp. 188-216, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[14] Paul Shaw, "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems," *Principles and Practice of Constraint Programming*, vol. 1520, pp. 417–431, 1999. [CrossRef] [Google Scholar] [Publisher Link]

[15] Laurence A. Wolsey, *Integer Programming*, New York, NY: John Wiley & Sons, 2020. [Google Scholar] [Publisher Link]

[16] George L. Nemhauser, and Laurence A. Wolsey, "Integer and Combinatorial Optimization," New York, NY: Wiley-Interscience, 1988. [Google Scholar] [Publisher Link]

[17] Iain Dunning, Stuart Mitchell, and Michael O'Sullivan, "Pulp: A Linear Programming Toolkit for Python," 2011. [Google Scholar] [Publisher Link]