



**WEST UNIVERSITY OF TIMIȘOARA
FACULTY OF MATHEMATICS AND COMPUTER
SCIENCE
MASTER STUDY PROGRAM: Big Data - Data Science,
Analytics and Technologies**

Schedule for Examination Session

Coordinator:

Lect. Dr. Madalina Erascu

Team Members:

Hurloi Iulia

Tuns Adrian-Ioan

Voicu Bogdan Andrei

TIMIȘOARA

2022

Contents

1	Introduction	2
1.1	Problem Description	2
1.2	Project Motivation	2
2	Related Work	3
2.1	Related Work	3
2.2	Conclusions	3
3	Problem Constrains	4
3.1	Hard Constrains	4
3.2	Soft Constrains	5
3.3	Constrains Modelling	5
4	Data set	6
4.1	Data Set Structure	6
5	Solution	8
5.1	Technologies	8
5.2	Implementation	8
5.3	Graphical User Interface	12
5.4	Results	13
5.4.1	Problems Encountered	14
6	Conclusion and Future Work	15
	Bibliography	16

Chapter 1

Introduction

1.1 Problem Description

The semi-automatic generation of a proper examination schedule is a problem often encountered by many institutions of education.

The difficulty of achieving a proper semi-automatic generation is mainly due to the high number of both hard and soft constraints, as specified in the Problem Constrains chapter, that need to be satisfied, in order for both the students and the professors to attend the examinations for which they are enrolled in, respectively instruct.

1.2 Project Motivation

The importance of this project is linked to the problem previously presented, as this is a problem that has been studied for a fairly long period of time, and for which the existing solutions are open for improvements.

The main purpose of this project is the realization of a semi-automatic application that can satisfy the requirements of an examination scheduler, therefore reducing the time usually spent by secretaries and professors in creating the schedule.

Chapter 2

Related Work

2.1 Related Work

As noted in the work of Moreira José [4], several scientific works have approached the examination scheduling problem and the timetabling related problem.

In their work, Oliveira and Reis [1] presents the UniLang, a language used for representation of the timetabling problem, that intends to be a standard suitable as input language for any timetabling system. It offers a clear and natural way to represent data, constraints, quality measures and solutions for different timetabling problems (such as school and university timetabling or examination scheduling).

Gröbner, Wilke and Büttcher [3] present an approach to generalize all the timetabling problems, describing the basic structure of this problem, while also proposing a generic language that can be used to describe timetabling problems and its constraints.

Fernandes, Caldeira, Melicio and Rosa [2] classified the constraints of class-teacher timetabling problem in constraints strong and weak. Violations to strong constraints resulting in a invalid timetable, while violations to weak constraints result in valid timetable, but affect the quality of the solution.

2.2 Conclusions

By analysing the results obtained in the published works mentioned above, it can be concluded that the semi-automatic generation of an examination schedule is indeed achievable.

A similar approach as that of Fernandes [2] will be used for the solution in this project, by firstly establishing the hard and soft constraints (called strong and weak in the cited work) and then by working on meeting all of the hard constraints, while also taking into consideration the weak constraints.

Chapter 3

Problem Constrains

The constraints know so far for the semi-automatic examination scheduling can be further organized in hard and soft constraints. Hard constraints, or strong constraints, as mentioned above citing the related work studied, are to be all meet while developing the algorithm as they define the most important criteria to be satisfied in the resulting schedule. The soft constraints, or weak constraints, while they can be omitted, will be tried to be taken into consideration.

As the main goal of obtaining an fully functional semi-automatic examination scheduling is a hard constraint in our project, the main focus became adapting the problem to the current academic situation as follows.

3.1 Hard Constrains

The hard constraints known so far are the following:

- The exams should be scheduled at group level; as there is no classroom space constraint due to all courses and exams being online, the whole year could attend the exam at the same time. Furthermore, as some sections are enrolled in the same subject taught by the same professor, a grouping level could be represented by enrollment to the subject

For i and j in number of exam, $j \neq i$

$$\forall e_i == e_y, e_i \Rightarrow e_y$$

- The exams must be scheduled between start and end day (exams session period)

For i in number of exam

$$\wedge e_i(\vee ED[e])$$

- The exams must be scheduled between the hours 8:00 and 18:00
- Each exam must be scheduled in exactly one time slot where the duration of the exam is less than or equal to the duration of the time slot

For i in number of exam

$$\wedge e_i(\vee EP[e])$$

- A professor should have at most one exam in any given time slot
For i in number of professors

$$\wedge pr_i(\vee EP[pr])$$

- A student should have at most one exam at any given time slot

3.2 Soft Constrains

The soft constraints known so far are the following:

- Some professors have some preferences (e.g. exams should not be scheduled in the weekend, or a certain exam should be scheduled after 17:00, etc.)

3.3 Constrains Modelling

Notations used:

- $E = e_0, \dots, e_n$ - list of exams
- $Pr = pr_0, \dots, pr_n$ - list of professors
- $P = p_0, \dots, p_n$ - list of time-slots in the exam session (e.g. 2h/per exam \Rightarrow 5 different exams per day, provided the hour interval is between 8:00 and 18:00)
- $D = 0, \dots, x$ - number of days available for exams
- $EP[e]$ - denotes the time-slot in which E is scheduled, $EP[e] \in P$
- $ED[e]$ - denotes the day in which E is scheduled, $ED[e] \in D$
- Time-slot spread:
 - $\forall e \in E : ED[e] = \lceil EP[e]/5 \rceil$ - provided there are at most 5 time-slots in a day for exams

Chapter 4

Data set

The data set 4.1 used as a sample for the program is stored in a CSV file (Comma Separated Variables) by the consideration of easier access of the tabular data format.

4.1 Data Set Structure

Structured in four columns and 66 rows, the data set contains all exams of the first semester for computer science sections (Computer Science in Romanian, Applied Computer Science and Computer Science in English) of West University of Timișoara, for all years of studies (1, 2 and 3).

The columns are represented by the following:

- Subject - the name of subjects studied in the first semester by Computer Science students, that require an exam as form of grading their activity. The abbreviation form was used for each of them
- Section - corresponding section for the exam (Computer Science in Romanian = IR, Applied Computer Science = IA and Computer Science in English = IR)
- Teacher - lecture instructor corresponding to the subject and section
- Year - denotes the year a subject is studied in, having the values 1, 2 or 3

All the information about the exams was taken from the official website [7] of the university.

Figure depicting the first 20 entries of the data set:

	C1	C2	C3	C4
1	Subject	Section	Teacher	Year
2	IA	IR	V. Negru	3
3	TW	IR	G. Iuhasz	3
4	ED	IR	L. Tanasie	3
5	MPP/SP	IR	A. Popovici/ D. Pop	3
6	MPI/ITB/TSS	IR	L. Coroban/ C. Cira/R. Seculi	3
7	MLL	IR	L. Tanasie	3
8	DISCIPLINE COMPLEMENTARE	IR	PROFESOR-COMPLEMENTARE	3
9	SP	IA	D. Pop	3
10	TW	IA	G. Iuhasz	3
11	IA3	IA	V. Negru	3
12	PS/MPP	IA	M. Gaianu/ A. Popovici	3
13	MPI/TSS	IA	L. Coroban/R. Seculi	3
14	MLL	IA	V. Iordan	3
15	DISCIPLINE COMPLEMENTARE	IA	PROFESOR-COMPLEMENTARE	3
16	IA	IE	H. Popa Andreescu	3
17	TW	IE	G. Iuhasz	3
18	ED	IE	E. Kaslik	3
19	APP	IE	A. Copie	3
20	DP/IB/IT	IE	F. Micota/ C. Cira/ C. Bonchis	3

Figure 4.1: Data Set Sample

Chapter 5

Solution

The implementation of the solution can be found at the following Github repository:
<https://github.com/TunsAdrian/Examination-Schedule-Generation>

5.1 Technologies

In order to obtain the examination scheduler generator, python [5] programming language was used, along with Z3Prover [6].

Python was chosen both for its simplicity and for its usability in scientific computing, while Z3Prover was chosen because it is an high-performance Satisfiability Modulo Theories (SMT) solver, that can be easily integrated with python.

The resulting solution was printed using the package "tabulate" [8] in order to obtain a readable presentation of the mixed textual and numeric data as a table format.

The Graphical User Interface was created with the package "tkinter" [9], a lightweight GUI toolkit.

5.2 Implementation

The aim of the application is to obtain a schedule for the exams in the exam session for the Computer Science department of West University of Timișoara by formulating the hard constraints in a form that can be understood by the solver.

The first step for achieving this is represented by validating the input, and by checking the data to not contain characters that are reserved for later usage in the program, for another specific purpose.

```
def validate_input(row_data):
    if ' '.join(row_data).find('~') != -1:
        resultArea.insert(END, "Input data can't contain the following characters: '~")
        raise Exception("Input data can't contain the following characters: '~")
    if row_data[1].find('/') != -1 or row_data[3].find('/') != -1:
        resultArea.insert(END, "Section and Year can't contain the following characters: '/'")
        raise Exception("Section and Year can't contain the following characters: '/'")
```

Figure 5.1: Validate Input Function

Dictionaries were used to handle the data, having a structure generated under the form of key = **Subject+teacher+year**, followed by value = list of pairs **Specialisation+Year**, denoting which sections of each year are enrolled in the same course instructed by the same teacher. This type of data further minimises the effort of searching for the groups that need to take the online exam at the same time.

```
def create_courses_dict(subject, section, teacher, year):
    # strip leading/trailing spaces and replace the rest with "-"
    subject = ' '.join(subject.split()).replace(' ', '-')
    section = ' '.join(section.split()).replace(' ', '-')
    teacher = ' '.join(teacher.split()).replace(' ', '-')
    year = ' '.join(year.split()).replace(' ', '-')

    # create a dict with key made from exam subject name, teacher and year and value from a list containing the
    # section and year
    # e.g. key: Subject~Professor~Year, value: [Specialization1Year, Specialization2Year]
    if subject + '~' + teacher + '~' + year in courses_dict.keys():
        courses_dict[subject + '~' + teacher + '~' + year]["specializations"].append(section + year)
    else:
        courses_dict[subject + '~' + teacher + '~' + year] = {"specializations": [section + year]}
```

Figure 5.2: Create Dictionary Function

The CSV file format also includes packages of subjects that students need to enroll in, and in order to parse the situation properly, exams with the structure **Subject1/Subject2/Subject3**, together with professors mentioned in a similar way **Professor1/Professor2/Professor3** are needed, each of them being separated by "/" and considered as separate, unique entries.

```
with open(file_path, 'r') as file:
    csv_reader = csv.reader(file, delimiter=',')
    header = next(csv_reader)

    for row in csv_reader:
        validate_input(row)

        # check if subject and teacher fields have '/' (denoting packages of optional subjects) and register each
        # subject-teacher entry
        if row[0].find('/') != -1 and row[2].find('/') != -1:
            optional_subjects = row[0].split('/')
            optional_subjects_teachers = row[2].split('/')

            if len(optional_subjects) != len(optional_subjects_teachers):
                raise Exception(f"{row[0]} doesn't have teachers specified for all subjects.")

            for i in range(len(optional_subjects)):
                create_courses_dict(optional_subjects[i], row[1], optional_subjects_teachers[i], row[3])
        else:
            create_courses_dict(row[0], row[1], row[2], row[3])
```

Figure 5.3: Parse CSV With Package Courses

First step for the actual scheduling algorithm is to generate all the possibilities of placement of the exams in the time slots of every day of the exam session.

There are four variables that need to be given a value in the beginning of the application:

- `time_per_exam` - usually set to 2, as the vast majority of exams request a time slot of up to 2 hours
- `exams_starting_hour` - set to 8, as the hard constraint states all exams should be scheduled between 8 and 18
- `timeslots_number` - number of permitted time slots to be selected in a day depending on the time per exam and taking into consideration that the exam day ends at 18
- `days_number` - number of days in the exam session

```
# create a dict with key from exam name, teacher, year and values from list of same key with nr of days and of timeslots
for course in courses_dict:
    for specialization in courses_dict[course]["specializations"]:
        # e.g. key: Subject~Professor~Year/SpecializationYear, value: list with: [key + /day/timeslot]
        slot_dict[course + '/' + specialization] = [
            Bool(course + '/' + specialization + '/' + str(i) + '/' + str(j))
            for i in range(days_number)
            for j in range(timeslots_number)]
```

Figure 5.4: Function Generating All Possibilities

The next step is represented by adding the constraints with the help of loops that process the generated possibilities and take the necessary actions in order to fulfill the request.

First constraint is that there is at most one exam scheduled for every subject of any year or specialisation. The function feeds into the solver pairs of exams comparing the time slots they are assigned to and applying the function **Implies** with the second parameter negated in order to filter the exams scheduled in one time slot that are not the equal.

```
# At most one exam for each course and each specialization taking that course in the whole timetable
for slotEntry in slot_dict:
    for i in range(number_of_slots):
        for j in range(number_of_slots):
            if i != j:
                solver.add(Implies(slot_dict[slotEntry][i], Not(slot_dict[slotEntry][j])))
```

Figure 5.5: At Most One Exam Per Session

Second function ensures that there is at least one time slot for every exam.

```
# At least one exam for each course and each specialization taking that course for all days
for slotEntry in slot_dict:
    solver.add(Or([slots for slots in slot_dict[slotEntry]]))
```

Figure 5.6: At Least One Exam Per Session

For the subjects instructed by the same professor for more then one section, the exam should be in the same time slot. This is achieved by comparing the subject, teacher, year keys for all exams and put them in the same time slot by the function "Implies", as Exam1 implies Exam1, as long as subject, teacher and year are the same.

```
# For all specializations having a specific exam with the same teacher, the exam should be on the same day
for slotEntry in slot_dict:
    for i in range(number_of_slots):
        for remaining in slot_dict:
            if remaining.split('/')[0] == slotEntry.split('/')[0] and slotEntry != remaining:
                solver.add(Implies(slot_dict[slotEntry][i], slot_dict[remaining][i]))
```

Figure 5.7: Exam For All Sections Enrolled In Subject With Same Professor

To make sure that every specialization will have at most one exam in a day, a function of "Implies" will be applied on exams from the same specialisation and year, stating that every slot of one day should have at most one exam from specialisation x.

```
# At most one exam for a specialization in one day
for slotEntry in slot_dict:
    for i in range(number_of_slots):
        for remaining in slot_dict:
            # check specialization name
            if remaining.split('/')[1] == slotEntry.split('/')[1] and slotEntry != remaining:
                y = int(i / timeslots_number)
                y = y * timeslots_number

                for j in range(y, y + timeslots_number):
                    solver.add(Implies(slot_dict[slotEntry][i], Not(slot_dict[remaining][j])))
```

Figure 5.8: One Exam Per Day For Every Section

Lastly the constraint about professors having just one exam in an available time slot to avoid overlapping. The flow for this function is similar to the one previously mentioned.

```

# At most one exam for a teacher in one timeslot
for slotEntry in slot_dict:
    for i in range(number_of_slots):
        for remaining in slot_dict:
            # check teacher name
            if remaining.split('/')[0].split('~')[1] == slotEntry.split('/')[0].split('~')[1] and slotEntry != remaining:
                y = int(i / timeslots_number)
                y = y * timeslots_number

            for j in range(y, y + timeslots_number):
                # check if current entry has the same day and hour as the others, and that it doesn't have
                # the same exam name
                if slot_dict[slotEntry][i].__str__().split('/')[0].split('~')[0] != \
                    slot_dict[remaining][j].__str__().split('/')[0].split('~')[0] and \
                    slot_dict[slotEntry][i].__str__().split('/')[2] == \
                    slot_dict[remaining][j].__str__().split('/')[2] and \
                    slot_dict[slotEntry][i].__str__().split('/')[3] == \
                    slot_dict[remaining][j].__str__().split('/')[3]:
                    solver.add(Implies(slot_dict[slotEntry][i], Not(slot_dict[remaining][j])))

```

Figure 5.9: One Exam Per Time Slot For Professor

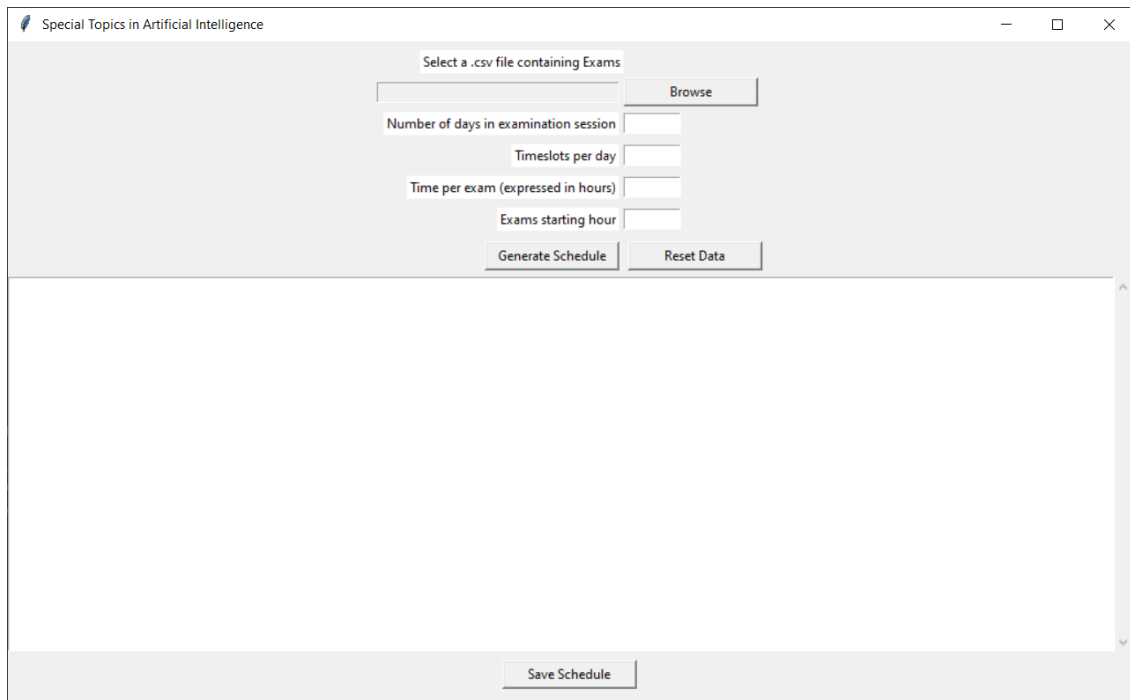
After writing the necessary constraints, the application checks if the solver result is SAT, and if the answer is affirmative it prints a message confirming the status and a generated table containing the solution, generated by the function `tabulate`. On the contrary if the solver is UNSAT, a corresponding message will be printed.

5.3 Graphical User Interface

A minimalist graphical user interface was also created in order to exemplify the way such an application would look and function in a real environment.

The interface is minimal, composed of four input variables with a designated space for values to be passed to the system. A button was implemented for uploading the files with the detailed data about the exams.

The resulting table is printed on the scrollable text section of the page, right above the button having the option to save the results to a text file.



The image shows a graphical user interface (GUI) for a scheduling application. The window title is "Special Topics in Artificial Intelligence". The interface is divided into two main sections. The top section contains a form for inputting exam details. It starts with a label "Select a .csv file containing Exams" above a text input field and a "Browse" button. Below this are four more input fields: "Number of days in examination session", "Timeslots per day", "Time per exam (expressed in hours)", and "Exams starting hour". At the bottom of this section are two buttons: "Generate Schedule" and "Reset Data". The bottom section of the window is a large, empty rectangular area, likely intended for displaying the generated schedule. At the very bottom of the window, centered, is a "Save Schedule" button.

Figure 5.10: Graphical User Interface

5.4 Results

The 5.11 figure depicts how the results look for the full list of exams, having 10 days set for the examination session, 3 timeslots per day, starting at 8:00 and having 2 hours per exam.

Exam-Name	Specialization	Year	Teacher	Day	Hour
AC	IR	1	L. Mafteiu Scail	1	02:00 - 10:00
AC	IA	1	A. Popovici	1	02:00 - 10:00
Etica	IE	1	A. Craciun	1	02:00 - 10:00
PI	IR	2	M. Iordan	1	02:00 - 10:00
PI	IA	2	I. Tarba	1	02:00 - 10:00
PI	IE	2	Stinga Oana	1	02:00 - 10:00
MLL	IR	3	L. Tanasie	1	02:00 - 10:00
MLL	IA	3	V. Iordan	1	02:00 - 10:00
MLL	IE	3	M. Erascu	1	02:00 - 10:00
SOI	IR	2	F. Fortis	10	02:00 - 10:00
SOI	IA	2	F. Fortis	10	02:00 - 10:00
IA	IR	3	V. Negru	10	02:00 - 10:00
ASDI	IR	1	D. Zaharie	2	02:00 - 10:00
LC	IA	1	A. Fortis	2	02:00 - 10:00

Figure 5.11: Valid Examination Schedule

5.4.1 Problems Encountered

The hard constraint of settings a certain number of days between two exams could not be met. The tried way of achieving this constraint was by iterating over each entry of the slots dictionary and by adding the constraint that the current entry implies that it is not represented by all the other entries that didn't have the number of days equal to the current entry day plus the number of desired days.

However, this approach proved to be highly inefficient, as it would have a way to large time complexity.

Chapter 6

Conclusion and Future Work

The purpose of the project to generate a semi-automatic scheduling that is correct from the standpoint of satisfying several hard constraint was fulfilled.

For future work the following are planned to be implemented:

- Refactor the code in order to obtain a better time for processing and schedule generation
- Implement a better looking graphic user interface that is more intuitive and user friendly
- Satisfy the hard constraint of having days set between exams
- Find a way to satisfy the soft constraint of taking into account a professor preference and add the solution in the GUI

Bibliography

- [1] Luís Reis and Eugénio Oliveira. “A Language for Specifying Complete Timetabling Problems”. In: Aug. 2000, pp. 322–341. ISBN: 978-3-540-42421-5. DOI: 10.1007/3-540-44629-X_20.
- [2] Carlos Fernandes, João Caldeira, Fernando Melicio, and Agostinho Rosa. “Infected Genes Evolutionary Algorithm for School Timetabling”. In: (Jan. 2002).
- [3] Matthias Gröbner, Peter Wilke, and Stefan Büttcher. “A Standard Framework for Timetabling Problems”. In: Aug. 2002, pp. 24–38. ISBN: 978-3-540-40699-0. DOI: 10.1007/978-3-540-45157-0_2.
- [4] José Moreira. “A System for Automatic Construction of Exam Timetable Using Genetic Algorithms”. In: *Tékhne - Revista de Estudos Politécnicos* 6 (June 2008).
- [5] *Welcome to Python.org*. [Online; accessed 3. Dec. 2021]. Dec. 2021. URL: <https://www.python.org>.
- [6] Z3prover. *z3*. [Online; accessed 3. Dec. 2021]. Dec. 2021. URL: <https://github.com/Z3Prover/z3#readme>.
- [7] *Facultatea de Matematică și Informatică – Universitatea de Vest din Timișoara*. [Online; accessed 20. Jan. 2022]. Jan. 2022. URL: <https://www.info.uvt.ro>.
- [8] *tabulate*. [Online; accessed 18. Feb. 2022]. 2022. URL: <https://pypi.org/project/tabulate>.
- [9] *tkinter — Python interface to Tcl/Tk — Python 3.10.2 documentation*. [Online; accessed 18. Feb. 2022]. 2022. URL: <https://docs.python.org/3/library/tkinter.html>.