A MIP and CP approach for International University Timetabling Problem

Viet Linh Vu, Ningzhen Rao, Thi Ngoc Anh Nguyen, Anni Huang, Guowei Yang, Jingxue Tang

School of Computing and Information Systems, Singapore Management University, Singapore 178902, Singapore

Abstract

The university timetabling is a classical combinatorial optimization problem that takes a large number of variables and constraints into account. Student allocating and class scheduling decisions need to be made jointly against the backdrop of classroom and timeslot penalty and subject to practical rules. We model this problem as a two-stage mixed integer program which search for a feasible solution at the start and graduately improve from it iteratively, which reduced the time to get optimal solution by 58%. Besides, we used a constraint programming method with the same objective function and constraints and compared the results for the two methods on 2019 ITC datasets. A comparison with results reported in this passage shows that the MIP approach outperformed the constraint programming approach in faster speed and better solution.

Keywords: University timetabling, Mixed integer linear programming, Constraint Programming.

1. Introduction

Educational timetabling is an ongoing challenging administrative task that is required in most academic institutions. This is mainly due to a large number of constraints and requirements that have to be satisfied. Educational timetabling problems have been classified as NP-hard problems and can be divided into three types: exam timetabling, course timetabling and high school timetabling[1]. The domain of high school timetabling is not well developed when compared to other fields of educational timetabling such exam timetabling and high-school timetabling.

The biggest challenge is to generate a good solution within reasonable time. In previous work, their MIP approach is very slow for instances with more than 606300 constraints.

Thereby we put forward a method where we don't put objective function for the first round, and find a better solution each time iterative until we meet the stopping criteria to save the time.

First, we developed our decision variables and objective function for the two models. Then, we defined our constraints with three assumptions to simply this problem. After that, we build the two models, whose data structure and general structure will be described briefly. Finally, we build functions to check the conflicts of constraints and turn our solution into XML format and submitted on ITC homepage to validate our result.

2. Problem Definition and Assumptions

2.1. Problem Definition

The problem consists of university courses and student course requests. A course may have one or more classes such as one lecture and several seminars. A hierarchical course structure is defined to specify how students attend classes below in Figure 5. One class can have several meetings a week

Email addresses: `vietlinh.vu.2021@mitb.smu.edu.sg` (Viet Linh Vu), `nzrao.2021@mitb.smu.edu.sg` (Ningzhen Rao), `nanguyen.2021@mitb.smu.edu.sg` (Thi Ngoc Anh Nguyen), `annihuang.2021@mitb.smu.edu.sg` (Anni Huang), `guowei.yang.2020@mitb.smu.edu.sg` (Guowei Yang), `jxtang.2020@mitb.smu.edu.sg` (Jingxue Tang)

and must be scheduled at the same time and in the same room. It can be taught at either all or selected weeks of the semester. For example, a course may have two seminars of up to 20 students, and each seminar can take place on Mondays and Wednesdays at 9 am every other week, one seminar on odd weeks, the other on even weeks.

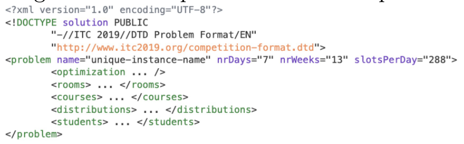
The aim is to find a proper time, a room and students for all the classes. Each class has a set of acceptable time and room assignments, together with non-negative penalties that define their suitability. All students must be correctly assigned to classes of their requested courses. Distribution constraints express relations among a set of classes, e.g., seminars must be taught after the lecture each week.

A feasible solution exists for each data instance and only feasible solutions are accepted in the competition. The overall solution cost should be minimized. It is computed as a weighted sum of all the time and room assignment penalties, penalties of broken soft distribution constraints, and the total number of student conflicts.

As an example, each class has a list of available time and room assignments given in the specific problem, each with a penalty. This means, for example, that only rooms that are big enough for the class and that meet all the other requirements (room type, equipment, building, etc.) are listed. Similarly, all the preferences have been already combined into a single penalty that is incurred when the time or room is assigned (see also details about problem transformations in the paper[2]).

Each problem has a certain number of weeks nrWeeks, a number of days nrDays in each week, and a number of time slots per day slotsPerDay. These parameters, together with the instance name, are defined in the root element of the XML file as shown in Figure 1.

Figure 1: XML specification for this problem

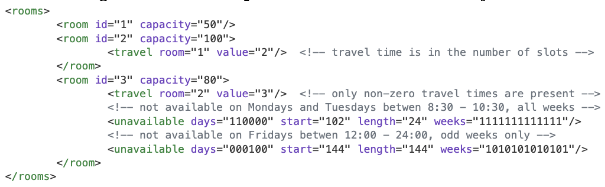


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE solution PUBLIC
        "-//ITC 2019//DTD Problem Format/EN"
        "http://www.itc2019.org/competition-format.dtd">
<problem name="unique-instance-name" nrDays="7" nrWeeks="13" slotsPerDay="288">
        <optimization ... />
        <rooms> ... </rooms>
        <courses> ... </courses>
        <distributions> ... </distributions>
        <students> ... </students>
</problem>
```

Having this representation of time, we define what it means when two classes overlap. Two classes overlap when they share at least one week, at least one day of the week and they overlap within this day in their time slots using the start and length of the classes.

Each room is specified by its id and capacity. A room may not be available at certain times, which are defined by unavailable elements using the days of the week, the start time slot, and a length in slots for a set of weeks during the semester. Non-zero travel times from other rooms are specified by the travel elements, which express the number of time slots it takes to get from one room to another. Travel times are expected to be symmetric and are listed only on one of the rooms. See Figure 2 for an example.

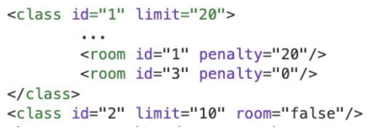
Figure 2: XML specification for room object



```
<rooms>
        <room id="1" capacity="50"/>
        <room id="2" capacity="100">
                <travel room="1" value="2"/>  <!-- travel time is in the number of slots -->
        </room>
        <room id="3" capacity="80">
                <travel room="2" value="3"/>  <!-- only non-zero travel times are present -->
                <!-- not available on Mondays and Tuesdays betwen 8:30 - 10:30, all weeks -->
                <unavailable days="110000" start="102" length="24" weeks="1111111111111"/>
                <!-- not available on Fridays betwen 12:00 - 24:00, odd weeks only -->
                <unavailable days="000100" start="144" length="144" weeks="1010101010101"/>
        </room>
</rooms>
```

A class cannot be placed in a room when its assigned time overlaps with an unavailability of the room or when there is some other class placed in the room at an overlapping time. Besides available times, each class also has a list of rooms where it can be placed. Each class needs one time and one room assignment from its list of possible assignments. It is possible for a class to only need a time assignment and no room. In this case, there are no rooms listed in Figure 3 and the room attribute of the class is set to false.

Figure 3: XML specification of possible rooms where a class can be placed



```
<class id="1" limit="20">
        ...
        <room id="1" penalty="20"/>
        <room id="3" penalty="0"/>
</class>
<class id="2" limit="10" room="false"/>
```

The example in Figure 1 specifies that the semester has 13 weeks and 7 days a week, e.g., from Monday to Sunday. While the data format allows for variations, in all the competition instances one time slot takes 5 minutes, which allows us to model travel times as well as classes that meet at irregular times. There are 288 time slots covering the whole day, from midnight to midnight.

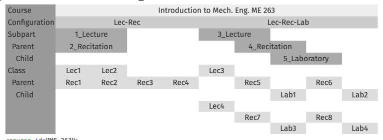
A class can meet several times a week during some or all weeks of the semester. All meetings of a class start at the same time, run for the same number of slots, and are placed in the same room. Examples of possible time assignments of a class are listed in Figure 4. Each time is specified by its starting time slot start within a day and have a duration length given as a number of time slots. Days and weeks with a meeting are specified using the days and weeks binary strings. For example, days 1010100 means that the class meets three times a week (on Monday, Wednesday, and Friday) each week when it is meeting. Similarly, weeks 0101010101010 specifies that the class would only meet during even weeks of the semester (during the 2nd, 4th, . . . , and 12th weeks of the semester).

Figure 4: XML specification of possible times when a class can meet

```
<class id="1" limit="20">
        <!-- Monday-Wednesday-Friday 7:30 - 8:20 All Weeks -->
        <time days="1010100" start="90" length="10" weeks="1111111111111" penalty="20"/>
        <!-- Tuesday-Thursday 9:00 - 10:15 All Weeks -->
        <time days="0101000" start="108" length="15" weeks="1111111111111" penalty="0"/>
        <!--Thursday 8:00 - 9:50 Even Weeks-->
        <time days="0001000" start="96" length="22" weeks="0101010101010" penalty="2"/>
</class>
```

Courses may have a very complex hierarchical structure of classes, i.e., events to be scheduled. An example of one course, ME 263, with the corresponding XML specification is shown in Figure 5.

Figure 5: XML specification for course object



```
<course id="ME 263">
        <config id="1"> <!-- Lec-Rec configuration, not linked (any Lec with any Lab) -->
                <subpart id="1_Lecture"> <!-- Lecture subpart -->
                        <class id="Lec1" limit="100"/>
                        <class id="Lec2" limit="100"/>
                </subpart>
                <subpart id="2_Recitation"> <!-- Recitation subpart -->
                        <class id="Rec1" limit="50"/>
                        <class id="Rec2" limit="50"/>
                        <class id="Rec3" limit="50"/>
                        <class id="Rec4" limit="50"/>
                </subpart>
        </config>
        <config id="2"> <!-- Lec-Rec-Lab configuration, linked -->
                <subpart id="3_Lecture"> <!-- Lecture subpart -->
                        <class id="Lec3" limit="100"/>
                        <class id="Lec4" limit="100"/>
                </subpart>
                <subpart id="4_Recitation"> <!-- Recitation subpart -->
                        <class id="Rec5" parent="Lec3" limit="50"/>
                        <class id="Rec6" parent="Lec3" limit="50"/>
                        <class id="Rec7" parent="Lec4" limit="50"/>
                        <class id="Rec8" parent="Lec4" limit="50"/>
                </subpart>
                <subpart id="5_Laboratory"> <!-- Laboratory subpart -->
                        <class id="Lab1" parent="Rec5" limit="50"/>
                        <class id="Lab2" parent="Rec6" limit="50"/>
                        <class id="Lab3" parent="Rec7" limit="50"/>
                        <class id="Lab4" parent="Rec8" limit="50"/>
                </subpart>
        </config>
</course>
```

Each course (ME 263 in Figure 5) has a unique id and consists of one or more configurations named config in the XML (Lec-Rec and Lec-Rec-Lab) and identified by their unique ids such that each student attends (some) classes in one configuration only. Each configuration consists of one or more subparts with their unique ids (Lec-Rec-Lab configuration has three subparts: 3 Lecture, 4 Recitation and 5 Laboratory). Each student must attend one class from each subpart of a single configuration. All students in the course configuration must be sectioned into classes of each subpart such that their limit is not exceeded (one student attending configuration Lec-Rec must take one class from each of its subparts 1 Lecture and 2 Recitation, e.g., Lec1 and Rec3). Each class has a unique id and belongs to one subpart (classes Rec5, Rec6, Rec7, and Rec8 belong to subpart 4 Recitation).

A class may have a parent class defined which means that a student who attends the class must also attend its parent class. For example, Lab3 has the parent Rec7 which has the parent Lec4. This means that a student attending Lab3 must also attend Rec7 and Lec4 and no other combination including Lab3 is allowed. On the other hand, there is no parent-child relationship between classes of subparts 1 Lecture and 2 Recitation, so a student may take any lecture Lec1 or Lec2 and any recitation Rec1, Rec2, Rec3 or Rec4.

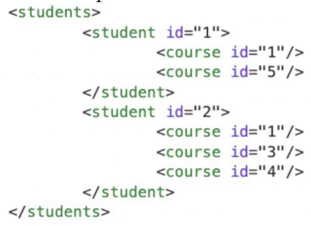
In the described problem, the imposed course structure is needed only for student sectioning, to be able to evaluate the possible combinations of classes that a student needs to take. All the other constraints that could be derived from the structure are already included in the distribution constraints given in the problem (see their description in Section Distribution Constraints) since each institution may use a different set of constraints. These typically include:

Classes that are in a parent-child relation cannot overlap in time (SameAttendees constraint is required, e.g., between any valid Lecture–Laboratory–Seminar combination that a student can take), classes of a subpart need to be spread in time as much as possible (NotOverlap constraint is placed between these classes imposing a penalty for each pair of classes that overlap in time) the lecture may (or must) be placed before all the seminars (Precedence constraint is placed between any pair of a lecture and a seminar that a student can take). Each class has a defined set of possible times when it can meet. Each eligible time has a specified penalty which must be included in the overall time penalization when the time is selected for the class (see Section Solution). Valid time specifications were described in Section Times.

Each class also has a defined a set of possible rooms where it can meet (if a room is needed). Each eligible room has a specified penalty to be included in the overall time penalization when selected (see Section Solution). Valid room specifications were given in Section Rooms.

Each student has a unique id and a list of courses that he or she needs to attend. Each course is specified by its course id. See Figure 6 for an example.

Figure 6: XML specification for student object



```
<students>
        <student id="1">
                <course id="1"/>
                <course id="5"/>
        </student>
        <student id="2">
                <course id="1"/>
                <course id="3"/>
                <course id="4"/>
        </student>
</students>
```

A student needs to be sectioned into one class of every subpart of a single configuration for each course from his or her list of courses. If a parent-child relation between classes of a course is specified, this relation must be respected as well (see also Section Courses). Also, the number of students that can attend each class is constrained by the limit that is part of the class definition.

A student conflict occurs when a student is enrolled in two classes that overlap in time (they share at least one week and one day of the week and they overlap in time of a day) or they are one after the other in rooms that are too far apart. This means that the number of time slots between the two classes is smaller than the travel time value between the two rooms. Student conflicts are allowed and penalized. The same penalty of one student conflict occurs for any pair of classes that a student cannot attend, regardless of the number of actual meetings that are in conflict or the length of the overlapping time.

2.2. Assumptions

The original university timetabling challenge is very complicated. Since we only have 3 weeks to tackle it, we simplify it with two assumptions and kept the others.

- There's no travel time between two classes.
- The soft constraints are computational heavy, so we will only consider the hard constraints.
- We only consider SameAttendees in the distributed constraints
- There is only one configuration for each course

3. Models and Algorithms

The goal of the Univesity Timetabling Problem is to build a semester timetable. The semester is organized as a set of weeks, and each week has a set of days. And each day has a set of time periods. Each time period has a starting time, length. Class $c \in C$ can be scheduled in some rooms and some time slots. And rooms can be unavailable in some time slots.

The inputs for the problem are a set of courses that should be scheduled at a specific times, and a set of students that should be assigned to a list of courses that they need to learn. In addition, courses that consists a set of subparts. Each subpart consists a set of classes. Student need to choose one of the classes in a subpart[2].

In the competitions, they generally used three ways to solve it. First one is by using mixed integer programming to obtain a feasible solution, then use local search algorithm [3] or Fix-and-Optimize[4] to optimise from it. Second one is using MaxSAT method by decomposing the problem into two sub-problems, namely Course timetabling and Student sectioning, then use local search to optimise the conflicts[5, 6]. Thirdly is by using Metaheuristics[7], such as Simulated Annealing[8], Tabu Search, iterative forward search [9], Genetic Algorithms, and hybrid approaches[10].

Constraint list

H1 Every class must be assigned a time

H2 Every class must be assigned a room, where applicable

H3 Every student must attend exactly one class for each subpart that he/she must attend

H4 For two classes with a parent-child relationship, if a class is assigned to a student, the parent class must also be assigned

H5 The capacity of each class in terms of the number of students must be satisfied

H6 A room cannot be used when it is unavailable

H7 These pairs of classes in SameAttendees cannot overlap in time because there are students who are studying both of them

H8 Two classes can't be at the same time and the same room

3.1. Mixed Integer Programming Model

In this subsection, we present our MIP formulation for this problem considering all the hard and soft constraints. The notation used in the problem formulation is presented in table 1. Firstly, there're some sets from the original data and supporting tables. They are a class set C, room set R, time period set T, and student set S, in which each element is denoted in its corresponding lower case. For each class, it has three attributes, namely time options denoted as CT_c, room options denoted as CR_c, capacity denoted as M_c, prerequisite denoted as CP_c (this attribute is optional). For each student, it has a list of required courses. And each course has a list of subparts. We extract this to be each student has a set of subparts which is denoted as SP_s. A single subpart is denoted as *sbp* which contains a list of classes. We flatten them and formed a set of classes for each student denoted as SC_s, containing all the classes each student can choose from. Besides, we have some parameters. For penalty, we have two kinds of penalty when choosing room and time slot, denoted as $pr_{c,r}$ and $pt_{c,t}$ respectively. And there's a limitation for each class denoted as M_c which indicates the maximum registration number of it. For decision variables, we have three, namely x, y, z which are shown below.

$$x_{c,t} = \begin{cases} 1 & \text{The class c takes place at time t} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{c,r} = \begin{cases} 1 & \text{The class c takes place in room r} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{s,c} = \begin{cases} 1 & \text{The student s is assigned to class c} \\ 0 & \text{otherwise} \end{cases}$$

The objective function in equation (1) of the problem consists of two weighted penalization part related to some time and room options for each class. Constraint eqs. (2) to (6), (9) and (10) ensures the hard constraint H1-H7 respectively.

$$\begin{aligned} \text{Min} \quad & \sum_{c\in C}\sum_{r\in CR_c} y_{c,r}pr_{c,r} \\ & + \sum_{c\in C}\sum_{t\in CT_c} x_{c,t}pt_{c,t} \end{aligned} \tag{1}$$

H1: Every class c must have a time assignment t. class $c \in C$:

$$\sum_{t\in CT_c} x_{c,t} = 1 \ for \ c \in C \tag{2}$$

H2: Every class c must be assigned a room r, where applicable.

$$\sum_{r\in CR_c} y_{c,r} = 1 \ for \ c \in C \tag{3}$$

H3: Every student s must attend exactly one class c from each subpart Pf of the selected course configuration f for each course o that he must attend.

$$\sum_{c\in SC_c} z_{s,c} = 1 \ for \ s \in S \tag{4}$$

H4: If a class c has a parent class c' defined, whenever the class c is assigned to a student s then the parent class c' must also be assigned to that student.

$$z_{s,c} <= z_{s,c'} \ for \ s \in S, for \ c \in C, if \ c' \ is \ CP_c \tag{5}$$

H5: The capacity M_c of each class in terms of the number of students must be satisfied.

$$\sum_{s\in SC_c} z_{s,c} <= \ M_c \ for \ c \in C \tag{6}$$

H6: A room cannot be used when it is unavailable. For every class $c \in C$ and for every $t \in CT_c$ and $r \in CR_c$, if the time assignment t overlaps with a period of unavailability of the assigned room r, then both these assignments cannot be made

$$x_{c,t} + y_{c,r} <= 1 \ for \ r \in CR_c, for \ t \in CT_c, for \ c \in C \tag{7}$$

H7: SameAttendee requirements. This part includes two parts: SameAttendee constraints provided by the problem (an extracted list of class pairs

Table 1: Notation

Symbol	Definition
Sets	
$t \in T$	set of time periods
$r \in R$	set of rooms
$c \in C$	set of classes
$s \in S$	set of students
$r \in CR_c$	set of rooms each class can use
$t \in CT_c$	set of time slots each class can use
$sbp \in SP_s$	set of subparts student s must attend
$c \in sbp$	set of class options each subpart has
$c \in CP_c$	prerequisite for class c (optional)
$c \in SC_s$	set of classes that each student need to learn
$(c_i, c_j) \in SA$	set of class pairs c_i, c_j that can't overlap
Parameters	
$pr_{c,r}$	penalty of choosing room r of class c
$pt_{c,t}$	penalty of choosing time t of class c
M_c	capacity of each class c
Decision Variables	
$x_{c,t}$	binary variable that indicates whether class c is scheduled to time slot t
$y_{c,r}$	binary variable that indicates whether class c takes place at room r
$z_{s,c}$	binary variable that indicates whether student s choose class c

that cannot be overlap in time - due to same professor/instructor attending...)

$$x_{c_1,t_1} + x_{c_2,t_2} <= 1 \; for \; c_1, c_2 \; in \; SA, \quad for \; t_1 \; in \; CT_{c1}, for \; t_2 \; in \; CT_{c2} \tag{8}$$

Second part is the SameAttendee constraints saying that class pairs that are having similar students attending them cannot be overlap in time

$$z_{s,c_1} + z_{s,c_2} + x_{c_1,t_1} + x_{c_2,t_2} <= 3 \; for \; s \in S, \quad for \; c_1, c_2 \; in \; SC_s, \quad for \; t_1 \; in \; CT_{c1}, for \; t_2 \; in \; CT_{c2} \tag{9}$$

H8: Two classes cannot happen at the same time in the same room. For every pair of classes $c_1, c_2 \in C$ and every potential common room assignment $r \in R_{c_1} \cap R_{c_2}$, and every time assignment $t_1 \in T_{c_1}$ and $t_2 \in T_{c_2}$ where t_1 and t_2 overlap, not all four assignments can be made simultaneously

$$x_{c_1,t_1} + x_{c_2,t_2} + y_{c_1,r} + y_{c_2,r} <= 3 \tag{10}$$

We add the objective function at the beginning but find that the model can't solve it in a reasonable time. So we come up with an architecture for our MIP model which generates a feasible solution, calculates its objective function, and compares it with the previous one. We will only accept a better solution in each iteration. The general structure of our MIP approach is shown in below algorithm 1.

Algorithm 1 MIP approach general structure

```
$sol_{pre} \leftarrow None$
while not meet the stopping criteria do
    $sol \leftarrow model.solve()$
    if $sol$ then
        $sol_{pre} \leftarrow sol$
    end if
    $penalty \leftarrow objective(sol_{pre})$
    $p_{time} \leftarrow m.sum(pt_{c,t} * x_{c,t} \; for \; c \in C, t \in CTime_c)$
    $p_{room} \leftarrow m.sum(pr_{c,r} * y_{c,r} \; for \; c \in C, r \in CRoom_c)$
    $model.add\_constraint(p_{time} + p_{room} <= penalty)$    ▷ New solution must be better than current one
end while
```

3.2. Constraint programming

The constraints for constraint programming are the similar to the ones in mathematical programming. But it has different decision variables. In mixed integer programming, the decision variables are all bool variables while in constraint programming, they are integers. For example, x_c denotes the time options for each class c, while in mixed integer programming, we use a bool variable $x_{c,t}$ to do that.

$$x_c = t, for\ c \in C$$

$$y_c = r, for\ c \in C$$

$$z_{s,sbp} = c, for\ s \in S, for\ sbp \in SP_s$$

For the constraints, H1, H2, H3 are automatically satisfied. So we do not need to include them. For H4, if a class has prerequisite class c', student also need to choose that class.

$$if\ z_{s,sbp} = c,\ z_{s,sbp} = c'$$

For H5, the capacity of each class in terms of the number of students must be satisfed

$$count(z, c)M_c$$

For H6, A room cannot be used when it is unavailable

$$If\ x_c = t,\ then\ y_c \neq r\ for\ t \in r.available()$$

For H7, same attendee requirement. If t_1 and t_2 overlap and $r_1 r_2$, we will add another constraint

$$If\ x_{c_1} = t_1\ then\ x_{c_2} \neq t_2\ if\ overlap(t_1, t_2)$$

$$If\ z_{s,sbp_1} = c_1\ and\ z_{s,sbp_2} = c_2\ and\ x_{c_1} = t_1$$

$$then\ x_{c_2} \neq t_2\ if\ overlap(t_1, t_2)$$

For H8, two class cannot be at the same time and in the same room

$$If(x_{c_1} = t_1)(x_{c_2} = t_2)(y_{c_1} = r_1), then\ y_{c_2} \neq r_2$$

4. Experiment result

The implementation was done in Python using the commercial software CPLEX. We extracted 7 datasets to test our models but some don't have students, some are too large to run in a reasonable amount of time. So we finally decide to use two datasets to compare our models. One is wbg-fal10, a small dataset with 150 students, 19 students, and 7 rooms. The other one is pu-cs-fal07, with 174 classes, 2002 students, and 13 rooms. We compared the model in general two aspects: the speed and quality of the model's result. To examine the speed of our model, we have two matrices. The first one is the execution time to get the first feasible solution, the second one is the execution time to get an optimal solution before meeting the stopping criteria. To examine the quality of the two models, we consider two matrices. First is the objective value of the first feasible solution, and the second is the objective value of the best solution.

Comparing the speed to get a feasible solution, the MIP method performs better on both of the datasets[1] as shown in Table 2

Table 2: Comparison of Speed to Get a Feasible Solution of Different Methods on ITC2019 comp datasets

Dataset Name	Execution time(s)	
	MIP	CP
wbg-fal10	**16.19**	739.84
pu-cs-fal07	**5.03**	18.68

Comparing the speed to get the optimal solution, the MIP method performs better at small dataset as shown in Table 3, while CP has better performance in large datasets. Besides, the time for MIP is steady while CP's speed highly depends on the specific configuration of the dataset.

Table 3: Comparison of Speed to Get the Optimal Solution of Different Methods on ITC2019 comp datasets

Dataset Name	Execution time(s)	
	MIP	CP
wbg-fal10	**327.68**	928.49
pu-cs-fal07	325.53	**260.8**

[1]https://www.itc2019.org/instances/all

In Table 4, we can see that MIP can get better optimal solution on small dataset. But for large dataset, they are the same.

Table 4: Comparison of the quality of the Optimal Result of Different Methods on ITC2019 comp datasets

Dataset Name	Best result (min. penalty)	
	MIP	CP
wbg-fal10	**16**	23
pu-cs-fal07	60	60

As we can see from Table 5, MIP model can generate a better initial solution for both datasets.

Table 5: Comparison of the quality of the First Feasible Solution of Different Methods on ITC2019 comp datasets

Dataset Name	Best result (min. penalty)	
	MIP	CP
wbg-fal10	**182**	439
pu-cs-fal07	**554**	686

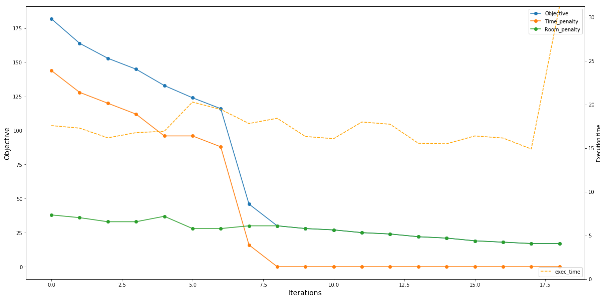
As we can see from Table 6, CP model has much less variables than MIP model. Since in CP, the decision varibles are IntVar, but in MIP we need much more Boolen variables to implement that.

Table 6: Comparison of Variable Number of Different Methods on ITC2019 comp datasets

Dataset Name	Variable number	
	MIP	**CP**
wbg-fal10	6272	**799**
pu-cs-fal07	38624	**3837**

The change of objective value in iterations on dataset wbg-fal10 is shown in Figure 7. As we can see, the penalty for room, time, and the total penalty is decreasing along with iterations which means our strategy for the MIP model is working. After 17 iterations, the execution time incerased dramatically. So for this dataset, we can stop after 17 iterations.

Figure 7: MIP Result



First, we compare performance of our MP and CP solution. For finding first feasible result, MP is outperform to CP in both 2 datasets. MP get faster and better objective to CP. While fore finding optimal solution, while in first dataset, MP still get faster result and better object, in second large dataset, performance of MP and CP seem to be the same. In general, MP work faster, It may because we have remove all soft constraint so the feasible solution must satify all the constraint Next is our MIP approach result by iteration. So you can see, the objective is the blue line, going down sharply in first 17 iteration and decrease slowly until last iteration and get optimal result at 100

Lastly, to validate our solution, we have convert our solution to xml competition format. So the format is each solution have a list of classes, each class is assigned in a time configuration and has a list of student. Here is the result we got from competition validator in Table 7. While our total penalty is much higher than showed before as we have simplified problem (our objective = total time penalty+ total room penalty)

Table 7: The gap with best known result for our two models

Dataset Name		wbg-fal10	pu-cs-fal07
Our Best Result	MP	537	740
	CP	633	1086
Best Known Result		36	200
Gap		1391.67%	270.00%

5. Findings and Insights

This article presented a mixed integer programming approach and a constraint programming approach for solving the timetabling problems of the

2019 International Timetabling Competition. Although the problem size for a typical timetable is very large to be solved exactly using traditional mixed-integer programming tools, several improvements can significantly reduce the size to manageable levels. Talking about speed, we find that MP is always quicker than CP to find a feasible solution. And it can find the optimal solution quicker on the small dataset, while CP can find the optimal solution quicker on the large dataset since it will not compute all the constraints like MP does. Considering about quality of solutions, we find that MIP can get better optimal solutions on small datasets, while on the large dataset, the quality of optimal solutions using CP and MIP are the same. If we just want to get a feasible solution, MIP is a better choice since it can get a better solution on both the small and large datasets.

References

[1] Mei Ching Chen et al. "A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities". In: *IEEE Access* 9 (2021), pp. 106515–106529. DOI: 10.1109/ACCESS.2021.3100613.

[2] Tomáš Müller, Hana Rudová, Zuzana Müllerová, et al. "University course timetabling and international timetabling competition 2019". In: *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018)*. Vol. 1. 2018, pp. 5–31.

[3] Efstratios Rappos et al. "A mixed-integer programming approach for solving university course timetabling problems". In: *Journal of Scheduling* (2022), pp. 1–14.

[4] Dennis S Holm et al. "A mip based approach for international timetabling competation 2019". In: *Proceedings of the International Timetabling Competition* 2020 (2019).

[5] Alexandre Lemos, Pedro T Monteiro, and Inês Lynce. "Introducing UniCorT: an iterative university course timetabling tool with MaxSAT". In: *Journal of Scheduling* (2021), pp. 1–20.

[6] Alexandre Lemos, Pedro T Monteiro, and Inês Lynce. "Minimal perturbation in university timetabling with maximum satisfiability". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research.* Springer. 2020, pp. 317–333.

[7] Abeer Bashab et al. "A systematic mapping study on solving university timetabling problems using meta-heuristic algorithms". In: *Neural Computing and Applications* 32.23 (2020), pp. 17397–17432.

[8] Nuno Leite, Fernando Melıcio, and Agostinho C Rosa. "A fast simulated annealing algorithm for the examination timetabling problem". In: *Expert Systems with Applications* 122 (2019), pp. 137–151.

[9] Tomás Müller, Roman Barták, and Hana Rudová. "Iterative Forward Search : Combining Local Search with Maintaining Arc Consistency and a Conflict-based Statistics". In: 2004.

[10] Amin Rezaeipanah, Samaneh Sechin Matoori, and Gholamreza Ahmadi. "A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search". In: *Applied Intelligence* 51.1 (2021), pp. 467–492.