Noname manuscript No.
(will be inserted by the editor)

A MIP based approach for International Timetabling Competation 2019

Dennis S. Holm* · **Rasmus Ø. Mikkelsen*** ·
Matias Sørensen · **Thomas R. Stidsen**

Received: date / Accepted: date

Keywords Mixed Integer Programming · Matheuristics · Fix and Optimize · Conflict graphs · University Timetabling · International Timetabling Competition 2019

1 Introduction

This summary paper was written as a part of the submission for the International Timetabling Competition 2019 (ITC2019). It aims to give an overview description of the algorithm used to solve the ITC2019 problem instances. Since the paper is limited to 4 pages, the description cannot be very comprehensive. The algorithm is divided into different parts. First part is a reduction algorithm where unnecessary information in the data is removed. It is followed up by two initial solution algorithms and a Fix-and-Optimize matheuristic. The initial solution algorithms and Fix-and-Optimize algorithms all depend on a Mixed Integer Programming (MIP) formulation, which will also be described briefly. Finally the computational setup is presented as it defines the resulting algorithm.

* First Author

Dennis S. Holm*
Akademivej
Building 358
2800 Kgs. Lyngby
E-mail: dsho@dtu.dk

Rasmus Ø. Mikkelsen*
Akademivej
Building 358
2800 Kgs. Lyngby
E-mail: rasmi@dtu.dk

Matias Sørensen
E-mail: sorensen.matias@gmail.com

Thomas R. Stidsen
Akademivej
Building 358
2800 Kgs. Lyngby
Tel.: +45 45254449
E-mail: thst@dtu.dk

2 Reducing the problems

The reduction of the problems concern two parts. One part considers the removal of times/rooms of classes that are never allowed in a feasible solution. The other part considers the removal of distribution constraints that are dominated by other distribution constraints.

2.1 Reducing times/rooms

This part is very useful for the MIP because it reduces the number of variables. Consider a graph where each vertex corresponds to a class-time pair. To get a valid class-time assignment in the problem, a vertex must be chosen for each of the classes. Now add an edge between two vertices if the two pairs cannot both be chosen in a valid solution. That could happen if a hard distribution constraint states, that the times of the two classes are not allowed simultaneously or if the two vertices represent the same class. Likewise a conflict graph of class-room pairs can be constructed.

Now consider a conflict graph G described as above and consider the sets of vertices $V(c_i)$ that represents class c_i. If $|V(c_i)| = 1$ for a c_i we denote the vertex as *fixed*. This means that any neighbour of a fixed vertex cannot be chosen in a valid solution, thus such vertices can be removed from G.

Consider a clique C in the graph. C describes that only one of the vertices $V(C)$ can be chosen. If $V(c_i) \subset V(C)$ for a specific c_i then the vertices of $V(C) \setminus V(c_i)$ can be removed from G. By reducing the graph G to G' with the above methods one might find vertices that are fixed in G' but not in G. It is therefore important to keep reducing G' until no more reductions can be made.

2.2 Reducing distribution constraints

Redundant distribution constraints: A constraint that consider at most one class, a soft constraint with 0 penalty, or cannot be violated by the classes it consider.

Dominated distribution constraints: A distribution constraint (hard or soft) d_1 is said to be dominated by a hard distribution constraint of equal type d_2 if the classes of d_1 is a subset of the classes of d_2.

Redundant and dominated distribution constraints are removed from the problem.

3 MIP

The Mixed Integer Programming formulation consider a binary decision variable $x_{c,t,r}$ that is equal to 1 if a class c is scheduled at time t in room r and 0 otherwise. If the problem includes student sectioning the MIP formulation also consider the binary variable $E_{s,c}$ which is equal to 1 if student s is attending class c and 0 otherwise.

The decision variable $x_{c,t,r}$ leads to auxiliary variables $y_{c,t}$, $z_{c,d}$ and $w_{c,r}$, which respectively consider the assignment of time t, day d or room r for a class c. Note that the variables $y_{c,t}$ and $w_{c,r}$ are represented by vertices in the conflict graphs presented in section 2. The distribution constraints used to define the edges of a conflict graph are modelled by a clique cover of the conflicts graphs. This is the modelling of most of the hard distribution constraints. Note that *SameAttendees* requires an additional conflict graph on $x_{c,t,r}$ when the times by themselves are not overlapping but the room assignments violate the constraint. Conflict graph are created for the soft distribution constraints as well. Here the edges have a cost corresponding to the distribution constraint(s) that created the edge. Each edge can be used as a constraint in the model. But to lower the amount of constraints

it is better to divide the graph into subgraphs where all edges have equal cost and then find a star cover of each graph. Each star is added as a constraint to the model.
The distribution constraints *MaxDays*, *MaxDayLoad*, *MaxBreaks* and *MaxBlock* are modelled in a more advanced way.
Student sectioning is performed like *SameAttendees* except that the cost of overlap between courses c_1 and c_2 is related to E_{s,c_1} and E_{s,c_2}.

4 Initial solution

An initial solution is constructed by two simple constructive matheuristics. The constructive heuristics split the problem into two or three parts respectively, where the parts are solved one followed by the other.

Algorithm 1 Two-Stage Constructive Algorithm (2SCA)

```
1: assign times and rooms to classes
2: assign students to classes
3: return assignments
```

For the 2SCA algorithm a MIP is defined with the only objective being the number of unassigned classes. When a feasible schedule is found, the schedule is given to the original MIP which is solved to assign students to classes.

Algorithm 2 Three-Stage Constructive Algorithm (3SCA)

```
1: assign times to classes
2: try to assign rooms to classes
3: while assignment of rooms was not possible do
4:     find a new assignment of times to classes
5:     try to assign rooms to classes
6: end while
7: assign students to classes
8: return assignments
```

In the 3SCA algorithm a MIP that considers only the assignment of times is solved first. The time-assignment is then given to another MIP that considers the assignment of rooms. If there is no feasible room-assignment to the given time-assignment another time-assignment will be found. When a feasible time and room assignment has been found the schedule is given to a MIP that considers student sectioning.

5 Fix-and-Optimize

To improve the solutions found in section 4 we use a Fix-and-Optimize matheuristic. The Fix-and-Optimize splits the decision variables into two sets F and U. We then consider the *subproblem* where the variables of F are fixed and we optimize the subproblem. The results are strongly dependent on the way the sets are chosen. If a large set U is chosen, the model will be too complex, on the other hand if U is too small there will be no improvement.

When choosing U we consider a neighbourhood of courses. That is the decision variable $x_{c,t,r}$ (and related auxiliary variables) for all classes that are part of a

given set of courses. As the instances vary greatly in difficulty, we choose the size of U to be 25% of class assignments as a base line.

Algorithm 3 Pseudo code for Fix-and-Optimize

```
MIP: Set solution sol*
MIP: Fix all variables to current value
while time do
    U = GetVariablesToUnfix()
    MIP: Unfix all U
    sol_new = Solve MIP
    if sol_new is improving then
        MIP: Set soltuion sol_new
    end if
    MIP: Fix all variables to current value
end while
```

The pseudo code of Fix-and-Optimize is shown in algorithm 3. The solution sol^* is the warm start solution, that could be the initial, best known or any other solution. The set of variables U is determined by function `GetVariablesToUnfix()`.

5.1 Dynamically updating parameters

The goal of Fix-and-Optimize is to find a balance between the MIP complexity and the availability and ease of finding improving solutions. On smaller and easier instances it is preferable to unfix in a more aggressive manner, while a more conservative strategy should be used for more difficult cases. The correct strategy is difficult to gauge a priori and therefore the parameters of Fix-and-Optimize are updated dynamically through the search.

6 Computational setup

When a data instance is received we start by reducing the file as described in section 2, this gives a reduced data instance that is used to construct the MIP described in section 3. The 3SCA algorithm described in section 4 is run without considering soft distribution constraints to find a pool of initial solutions. Additionally the 2SCA is also run. The MIP and a number of Fix-and-Optimize algorithms are run in parallel. The MIP has focus on improving the lower bound while the Fix-and-Optimize algorithms produce new solutions that are passed to the MIP to help reduce the branch and bound tree. The Fix-and-Optimize algorithms focus on separate neighbourhoods and regularly reset to the best known solution, such that none are "left behind". If enough time passes with no improvement in best known solution, the Fix-and-Optimize algorithms begin to diversify; each search starts from a new initial solution (from the 3SCA algorithm), no longer resets to the best known solution and considers all available neighbourhoods. This continues until the best known solution is improved, where after the Fix-and-Optimize algorithms revert to their normal strategy.

For instances where the number of students exceeds 30.000, we start an additional process where a specialized MIP is defined that applies student sectioning to fixed timetables. The timetables are produced by the 3SCA algorithm and a variant of the Fix-and-Optimize algorithm that is set up to produce timetables without considering the students.