Towards a Deterministic Model for Course Timetabling

PILAR POZOS PARRA  OSCAR CHAVEZ BOSQUEZ  JOSE LUIS GOMES RAMOS
University of Tabasco
Department of Informatics and Systems
Carretera Cunduacán - Jalpa Km. 1, Cunduacán, Tabasco
MEXICO
pilar.pozos@dais.ujat.mx  oscar.chavez@dais.ujat.mx  jose.gomez@dais.ujat.mx

Abstract: The Course Timetabling Problem consists in the weekly scheduling for all the lectures (events) of a set of university courses, subject to certain constraints. Unfortunately, course timetabling problems vary from university to university, and as far as we know, no standard formulation has been proposed from the community. Nevertheless, the International Timetabling Competitions, ITC-2002 and ITC-2007, have been organized with the aim of creating the common formulation for comparison. The formulation has become quite standard, and many solutions have been proposed in the literature. Most of the provided solutions utilize non-deterministic techniques. In this paper, we introduce a new deterministic algorithm for the solution of timetabling problems. The instances upon which the algorithm was tested are the official ones of the ITC-2002 web page. Almost all solutions run in less than 10% of the ITC-2002 benchmark time. The analysis is still ongoing, and it includes suitable extensions for tackling problems of the ITC-2007.

Key–Words: Course Timetabling Problem, International Timetabling Competition, STF Algorithm, Scheduling Problem, Deterministic Search, Implementation.

1 Introduction

The Course Timetabling Problem consists of fixing a sequence of events (lectures) of a set of university courses within a number of rooms and time periods, usually weekly, satisfying some constraints. During the recent years a large number of works has been directed to automated timetabling. Unfortunately, course timetabling problems vary from university to university, and as far as we know, no standard formulation has been proposed from the community. Nevertheless, throughout the years it has been possible to characterize common underlying formulations that could be used for comparing algorithms.

In particular the International Timetabling Competitions, ITC-2002 and ITC-2007, have been organized with the aim of creating the common formulation for comparison. The formulations have become quite standard, and many solutions have been proposed in the literature [14, 6]. Even when the competition is open to stochastic and deterministic approaches, all the proposed solutions appearing in the competition web pages are stochastic [4], they utilize non-deterministic techniques, such as tabu search, genetic algorithms, simulated annealing, ant colony optimization, etc., thus the competitions missed of deterministic approaches; as far as we know there is no record of some deterministic approach to find feasible solutions and effectively participate in the contest.

The motivation of this work was to develop a deterministic algorithm that solves the hard constraints of the 20 instances of the ITC-2002 in a timely manner, so we introduce a new deterministic algorithm, called the Sort Then Fix (STF) algorithm, for solving the timetabling problems. The instances upon which the algorithm was tested are the official ones of the ITC-2002. Even when the proposal solves only instances from ITC-2002, the analysis is still ongoing, and we expect suitable extension of the algorithm for ITC-2007. One of the major advantages of the proposal is that the best and worst cases are the same; this means that the running time does not need to be averaged over some number of runs.

Although there is evidence of the implementation of an initial algorithm that preprocess the information problem in order to find a feasible solution [6, 7], authors do not provide the overall penalty nor the time spent in this initial phase. This deterministic algorithm may create standard feasible inputs for other metaheuristics in order to measure their real performance.

This paper is organized as follows, the next section introduces the target problem; the third section presents the deterministic algorithm; the fourth section provides the results in detail, and finally we conclude and discuss some possible research directions.
2 Target Problem

We consider the problem of weekly scheduling a set of single events (or lectures). The problem has been discussed in [1] and it was the topic of ITC-2002 [2], where twenty artificial instances were proposed. The instances are available from the ITC-2002 web page. In addition, some instances have been proposed and made available via web in [5]. The data format used is an ad hoc fixed-structure text-only one. The input data comes in a single file containing the scalar values (number of events, rooms, room features, and students), followed by the elements of the input arrays, one per line. The output format must be done in a single file containing two scalar values (room and timeslot) per line, which indicate the schedule for events.

Formally, the problem consists of finding an optimal timetable within the following framework: there is a set of events $E = \{E_1, E_2, \ldots, E_{n_E}\}$ to be scheduled in a set of rooms $R = \{R_1, R_2, \ldots, R_{n_R}\}$, where each room has 45 available timeslots, nine for each day in a five day week. There is a set of students $S = \{S_1, S_2, \ldots, S_{n_S}\}$ who attend the events, and a set of features $F = \{F_1, F_2, \ldots, F_{n_F}\}$ satisfied by rooms and required by events. Each event is attended by a number of students, and each room has a given size, which is the maximum number of students the room can accommodate. A feasible timetable is one in which all events have been assigned a timeslot and a room so that the following hard constraints are satisfied:

1. no student attends more than one event at the same time;
2. the room is big enough for all the attending students and satisfies all the features required by the event; and
3. only one event is scheduled in each room at any timeslot.

In contest instance files there were typically 10-11 rooms, hence there are 450-495 available places. There were typically 350-400 events, 5-10 features and 200-300 students.

The problem proposes to penalize a timetable for each occurrence of some soft constraint violations, which are the followings:

1. a student has to attend an event in the last timeslot on a day;
2. a student has more than two classes in a row; and
3. a student has to attend solely an event in a day.

The problem may be precisely formulated as:

- let $F = \{F_1, F_2, \ldots, F_{n_F}\}$ be a set of symbols representing the features;
- $R_j = \{F'_1, F'_2, \ldots, F'_{n_{R_j}}\}$ where $F'_j \in F$ for $j = 1, \ldots, n_{R_j}$ and $n_{R_j}$ is the number of features satisfied by room $R_j$;
- $N = \{N_1, \ldots, N_{n_R}\}$ be a set of integer numbers indicating the maximum of students each room can accommodate;
- $E_i = \{F''_1, F''_2, \ldots, F''_{n_{E_i}}\}$ where $F''_j \in F$ for $j = 1, \ldots, n_{E_i}$ and $n_{E_i}$ is the number of features required by event $E_i$;
- $S_i = \{E'_1, E'_2, \ldots, E'_{n_{S_i}}\}$ where $E'_j \in E$ for $j = 1, \ldots, n_{S_i}$ and $n_{S_i}$ is the number of events student $S_i$ attends; and
- $T = \{T_1, \ldots, T_{45}\}$ be a set of timeslots.

Find a feasible solution, i.e. a set of pairs $\{(T'_1, R'_1), \ldots, (T''_{n_R}, R''_{n_R})\}$ such that:

- $T'_i \in T$ and $R'_i \in R$;
- $\neg(T'_i = T'_j)$ if $E_i, E_j \in S_k$ and $\neg(i = j)$;
- $E_i \subseteq R'_i$ and $\{|\{S_j | j = 1, \ldots, n_S \text{ and } E_i \in S_j\}| \leq N_k$ where $R'_i = R_k$;
- $\exists i \exists j (\neg(i = j) \land T'_i = T'_j \land R'_i = R'_j)$.

The competition adds a constraint over execution time, i.e. given the information about rooms, events, features, and students, find the best possible feasible solution within a given time limit. The time limit is given by a benchmark tool provided by the organizers.

3 Algorithm STF

We propose the Algorithm 1 to solve timetabling problems, it is based on events and rooms sorting.

The first step in the algorithm, apart from the obvious actions such as reading the problem file, is to find a binary matrix that represents the available rooms for every event. The following actions are performed:

- the number of students for each event is calculated and stored, $n_i = |\{S_j | j = 1, \ldots, n_S \text{ and } E_i \in S_j\}|$
- a list of available rooms is created for each event, $e_i = \{R_j | E_i \subseteq R_j \text{ and } n_i \leq N_j\}$

The problem may be precisely formulated as:
This first step allows us to reduce the problem by eliminating the information concerning features and room capacity, defining a new event set \( \{ e_1, \ldots, e_{nk} \} \), called in the algorithm \textit{Event.Room}, that includes the eliminated information. The new event set will be used for defining the most constrained event.

Our approach differs from direct heuristics [13], which usually fill up the complete timetable with one event at a time as far as no-conflicts arise, and then at that point they start making some swapping so as to accommodate other events. Our approach does not consider swapping; if an event cannot be scheduled then the next most constrained event is scheduled letting the event in question without room and timeslot. Thus we focus in the definition of the “most constrained event” as precise as possible in order to avoid letting an event without schedule.

The core strategy of the algorithm is to assign the most constrained event to the least constrained timeslot found for that event. In order to avoid strategies of movement that provide a limited form of backtracking to recover from mistakes, our aim is not to make mistakes of selection of placement and event.

Algorithm 2 finds the most constrained event, the intuitive ideas is as follows: sort \( IX_{Rest} \) (indexes of the unassigned events) in ascending order for the number of free slots, then for any index with equal number of free slots, sort them in descending order for the number students attending the event, then for any index with equal number of student attending the associated event, sort them in ascending order for the number of available rooms for the event, then for any index with equal number of suitable rooms associated to the event, sort them in ascending order for the number of potential accommodation of events in its available rooms, finally the most constrained event is the one appearing in the first place of the sorted \( IX_{Rest} \).

5 Conclusion

STF algorithm has been proposed and appears to solve timetabling problems in a natural way. The idea is intended to sort the events before scheduling. Unlike other approaches STF is defined in a deterministic manner. The STF algorithm solves effectively the 20 instances of the ITC-2002 within the time established by the benchmark program provided by the organizers of the contest. The approach utilized in the algorithm focuses only in the hard constraints, so the algorithm finds feasible solutions to the problem regardless the soft constraints.

As a main contribution, the STF algorithm may be used to find an initial solution to another optimization algorithm, so it can be combined with a non-deterministic approach, such as a metaheuristic, to build a hybrid algorithm that can find a better solution faster than a metaheuristic by itself.

The metaheuristics used by te other participants of the contest do a preprocessing of the problem data before starting the search process, so all of the algorithms start with a different initial solution. To tackle this issue, the timetables generated by the STF algorithm can be used as an initial solution for all of these metaheuristics, in order to measure the real effectiveness of each metaheuristic with respect to the other ones.

The STF algorithm finds a feasible timetable in much less than the time available, so the next version of the algorithm intend to solve both hard and soft constraints to find perfect solutions to the 20 problem instances. Also, the STF algorithm will be extended for solving instances of the ITC-2007, in order to manage the new constraints proposed in the competition and thus close the gap between the real problems and theoretical solutions.

\(^{1}\)The STF implementation is available under request via e-mail.
Data:
Data_file

Result:
Solution_file

begin
Timeslots ← 45
Read from Data_file:
Events : Number of events
Rooms : Number of rooms
Features : Number of features
Students : Number of students
RoomCapacity Vector of number of available places of rooms
Event_Student Matrix of attendance to events by students
Feature_Room Matrix of satisfied features by rooms
Feature_Event Matrix of required features by events
EventCapacity ← find number of places required by every event
Event_Room ← find binary matrix of suitable rooms for events
Events4Room ← find for every room the number of potential events to be allocated in
Room_Slot ← create a matrix of ones
Event_Slot ← create a matrix of ones
R_S ← create a matrix of zeros
IX_Rest ← indexes of events
for i=1 . . . Events do
    IX_i ← find the index of the most constrained event in IX_Rest
    IX_Room ← find the indexes of suitable rooms for event IX_i
    sort IX_Room from the least to the most constrained room
    Events4Room ← Events4Room − Event_RoomIX_i
    for r ∈ IX_Room do
        for s = 0 . . . Timeslots − 1 do
            OK ← 1, k ← 1
            IX_Student ← find indexes of students associated to event IX_i
            while k < i and OK do
                if R_S2,IX_k = r and R_S1,IX_k = s then
                    OK ← 0, break
                for j ∈ IX_Student do
                    if Event_StudentIX_k,j and R_S1,IX_k = s then
                        OK ← 0, break
                        k ← k + 1
                if OK then
                    R_S1,IX_i = s
                    Room_Slotr,s ← 0
                    for j = 1 . . . Events do
                        if any(Event_Studentj,IX_Student) then
                            Event_Slotj,s ← 0
                            break
                    if OK then
                        R_S2,IX_i ← r, break
            IX_Rest ← IX_Rest − IX_i
write to Solution_file:
end

Algorithm 1: STF
**Data:**

- IX_Rest
- Event_Room
- Room_Slot
- Event_Slot
- EventCapacity
- Events4Room

**Result:**

```plaintext
IX_j 
begin 
    sort IX_Rest in ascending order for Event_Room*IX_Rest * Room_Slot * Event_Slot 
    for in case of tie do 
        sort indexes with equal values in descending order for EventCapacity*IX_Rest 
    for in case of tie do 
        sort indexes with equal values in ascending order for \sum_{Room} Event_Room*IX_Rest 
    for in case of tie do 
        sort indexes with equal values in ascending order for Event_Room*IX_Rest * Events4Room 
    IX_j ← first index of the sorted IX_Rest 
end 
```

**Algorithm 2:** find the index of the most constrained event

**Table 1:** Results of the STF algorithm in the 20 problem instances. Time in seconds

<table>
<thead>
<tr>
<th>Instance</th>
<th>Hard constraints</th>
<th>Soft constraints</th>
<th>GNU Octave time</th>
<th>Matlab time</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0</td>
<td>940</td>
<td>139.491</td>
<td>35.775</td>
</tr>
<tr>
<td>02</td>
<td>0</td>
<td>870</td>
<td>129.923</td>
<td>33.507</td>
</tr>
<tr>
<td>03</td>
<td>0</td>
<td>926</td>
<td>132.475</td>
<td>34.354</td>
</tr>
<tr>
<td>04</td>
<td>0</td>
<td>1191</td>
<td>144.784</td>
<td>40.401</td>
</tr>
<tr>
<td>05</td>
<td>0</td>
<td>1395</td>
<td>130.249</td>
<td>33.199</td>
</tr>
<tr>
<td>06</td>
<td>0</td>
<td>1387</td>
<td>128.457</td>
<td>32.857</td>
</tr>
<tr>
<td>07</td>
<td>0</td>
<td>1736</td>
<td>104.068</td>
<td>27.671</td>
</tr>
<tr>
<td>08</td>
<td>0</td>
<td>1135</td>
<td>135.687</td>
<td>35.811</td>
</tr>
<tr>
<td>09</td>
<td>0</td>
<td>937</td>
<td>154.627</td>
<td>40.409</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>952</td>
<td>145.488</td>
<td>36.641</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>958</td>
<td>138.342</td>
<td>35.398</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>829</td>
<td>126.545</td>
<td>32.471</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1097</td>
<td>141.385</td>
<td>36.905</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>1605</td>
<td>125.638</td>
<td>30.660</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1448</td>
<td>131.486</td>
<td>31.743</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1032</td>
<td>163.606</td>
<td>42.263</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>1491</td>
<td>134.854</td>
<td>35.476</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>926</td>
<td>138.835</td>
<td>35.992</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>1325</td>
<td>156.347</td>
<td>40.449</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>1450</td>
<td>141.738</td>
<td>34.448</td>
</tr>
</tbody>
</table>
References:


