Efficient Timetabling Solution with Tabu Search

Jean-François Cordeau[1], Brigitte Jaumard[2,3], Rodrigo Morales[2]

[1]Ecole des Hautes Etudes Commerciales
Department of Production and Operations Management

[2]Université de Montréal
Department of Computer Science and Operations Research

[3]Canada Research Chair on
Optimization of Communication Networks

March 31, 2003

1. Introduction

We propose a Tabu Search metheuristic to solve the timetabling problem designed by Ben Paechter for the International Timetabling Competition organized by the Metaheuristic Network and sponsored by PATAT (Practice and Theory of Automated Timetabling).

We present the Tabu Search algorithm that we have developed in the next sections. The description is organized as follows. First we propose some preprocessing tests that we perform at the very beginning of the algorithm in order to reduce the solution space. Next we describe the first phase of the algorithm aimed at finding a first feasible solution, i.e., a solution satisfying all the hard constraints. We then proceed to the second phase with the goal of improving the value of the first feasible solution. We conclude the presentation of the algorithm with the results that have been obtained on the 20 instances provided by the challenge organizers.

2. Notations

Set of n_R rooms: $\mathsf{R} = \{R_j : j \in J\}$, with $J = \{1, 2, \ldots, n_R\}$.
Set of n_E events: $\mathsf{E} = \{E_i : i \in I\}$, with $I = \{1, 2, \ldots, n_E\}$.
Set of n_S students: $\mathsf{S} = \{S_k : k \in K\}$, with $K = \{1, 2, \ldots, n_S\}$.
Set of n_F features: $\mathsf{F} = \{F_\ell : \ell \in L\}$, with L = $\{1, 2, \ldots, n_F\}$.

Set of $n_T=45$ periods: $\mathsf{P} = \{P_t : t \in T\}$ with $T = \{1, 2, \ldots, 45\}$.

3. Preprocessing

We eliminate the room/feature matrix and define a room/event matrix Room_Event that takes the features, including the capacity requirement, into account:

$$\text{Room\_Event}[R_j, E_i] = \begin{cases} 1 & \text{if room } R_j \text{ has the features that are required by event } E_i \text{ and} \\ & \text{the capacity required for all the students attending event } E_i \\ 0 & \text{otherwise.} \end{cases}$$

In other words,
Room_Event[R_j, E_i] = 1 if
for all $\ell \in L$, Room_Feature[R_j, $F\ell$] ≥ Event_Feature[E_j, $F\ell$]
and Room_Event[R_j, E_i] = 0 otherwise.

4. Finding a first feasible solution

4.1 *An initial solution that satisfies the second and third hard constraints*

Three hard constraints :

- hard Constraint 1 : No student attends more than one event at the same time,
- hard Constraint 2 : The room is big enough for all the attending students and satisfies all the features required by the event,
- hard Constraint 3 : Only one event is in each room at any time.

The aim is first to generate an initial solution that satisfies the second and third hard constraints. In order to reach this goal, we solve a first assignment problem that is defined as follows (see Figure 1 for an illustration).

The first set is the event set, the second set is a room-period set $\mathsf{R\text{-}P} = \{(R_j, P_t): i \in I, t \in T\}$, and the assignment problem consists in assigning a room and a period to each event. Not all events can be assigned to any room or period: an event E_j can be assigned to room R_j and period P_t if and only if room R_i has the features and the student capacity required by event E_i, i.e., if Room_Event[R_j, E_i] =1. Therefore, we consider only the potential assignments ($E_i \leftrightarrow R_j$, P_t) such that Room_Event[R_j, E_i] =1.
This first assignment, i.e. a even/room-period assignment, has always a solution as all timetabling instances provided for the competition are assumed to have a solution.

We use the program of Jonker and A. Volgenant [1] to solve this first assignment problem. This program requires both sets of the assignment problem to have the same cardinality: in order to satisfy this constraint, we have added some dummy elements in the event set. With respect to the objective function of the assignment problem, all relations ($E_i \leftrightarrow R_j$, P_t) have

the same weight, and therefore the objective is only to find a feasible solution for the assignment problem.

After solving this first assignment problem, we obtain a first timetabling solution that satisfies the second and third hard constraints, but not necessarily the first one.

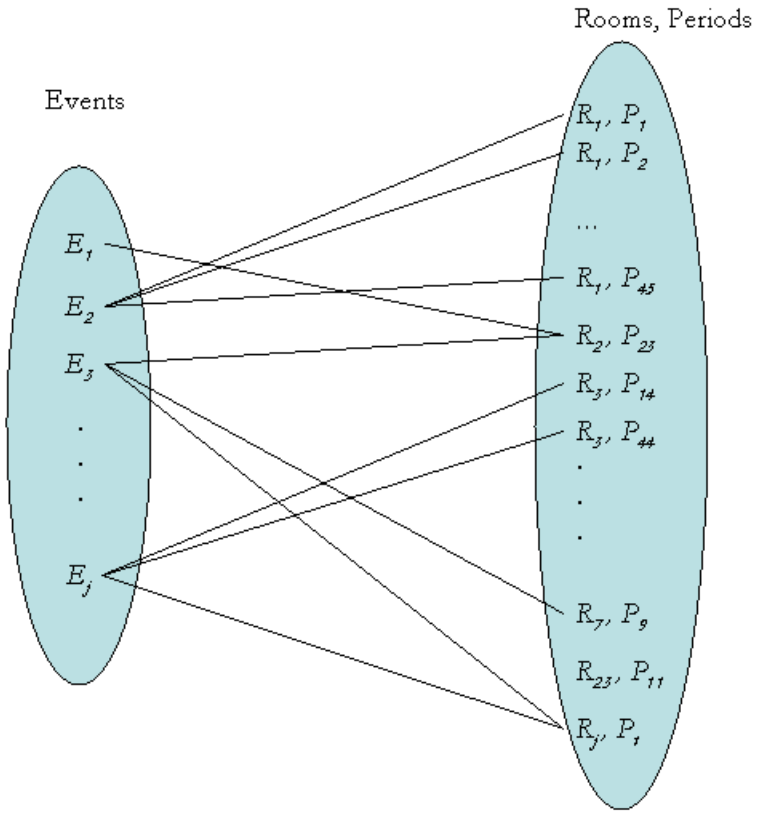

Figure 1. Event/Room-period assignment problem.

4.2 *A solution that satisfies all hard constraints*

In order to get a solution that satisfies all hard constraints, we next solve a sequence of assignment problems for each room. For a given room R_j , let us now define the R_j-assignment problem.

R_j-assignment problem.
The first set is the set of events that have been assigned to room R_j by the solution of the first assignment problem. The second set is the set of 45 periods. We weight each pair (E_i, P_j) as follows:

$w(E_i, P_t)$ = number of students with more than one event at period P_t if event E_i takes place at period P_t (considering the other events $E_{i'}$ ($i' \neq i$) that are taking place in the other rooms $R_{j'}, j \neq j'$).
We consider the R_j-assignment problem with the objective of finding the room/period assignment with minimum weight, i.e., with the least number of students attending more than one event at the same time.

The procedure, called *Obtain_a_feasible_timetabling_solution*, that iteratively solves the R_j-assignment problems in order to find a feasible solution is described below.

Procedure Obtain_a_feasible_timetabling_solution
$j \leftarrow 0$;
While the timetabling solution is not feasible ***do***
$j \leftarrow$ mod $(j+1, n_R)$;
solve the R_j-assignment problem
EndWhile.

In practice, for most timetabling instances, we need to revisit each room 3 times on average before obtaining a feasible solution.

4.3 *Phase 1: building a feasible timetabling solution*

The first phase of the algorithm is summarized below.

Algorithm MOUSTIK - Phase 1

Solve the event\room-period assignment problem ;
Apply the procedure *Obtain_a_feasible_timetabling_solution.*

---

5. Improving the first feasible solution

5.1 A First Basic Tabu Search Algorithm

In order to improve the quality of the first feasible solution, we design a Tabu Search algorithm. The basic move consists in, for a given triplet (R_j, P_t, E_i) to be selected, changing either the period, or the room, or both the period and the room. In other words, the set of moves consists in all the pairs of triplets such that:

$\big((R_j, P_t, E_i), (R_j, P'_t, E_i)\big)$ i.e. move $(R_j, P_t, E_i) \xrightarrow{\text{change of period}} (R_j, P'_t, E_i)$;

$\big((R_j, P_t, E_i), (R'_j, P_t, E_i)\big)$ i.e. move $(R_j, P_t, E_i) \xrightarrow{\text{change of room}} (R'_j, P_t, E_i)$;

$\big((R_j, P_t, E_i), (R'_j, P'_t, E_i)\big)$ i.e. move $(R_j, P_t, E_i) \xrightarrow{\text{change of both room and period}} (R'_j, P'_t, E_i)$.

We consider all possible triplets $j \in J$, $t \in T$, $i \in I$, and build all the pairs of triplets without restricting a priori to the moves leading to a feasible solution.

The main optimization criterion that drives the Tabu Search algorithm is the number of violated soft constraints. However, in order to give more flexibility to the search for better solutions, we allow infeasible solutions from time to time. Indeed, the feasibility domain is possibly highly nonconvex, so restricting the Tabu Search algorithm going from one feasible solution to a new feasible solution may increase the number of moves needed in order to reach a better solution, but may also prevent the algorithm from reaching a better solution because the path to permit it, is too sinuous.

Several ways to temporarily go through infeasible solutions have been proposed in the Tabu Search literature. We consider a scheme in which the value of the objective function controls the infeasibility of the current timetabling solution.

For a given timetabling solution $\mathbf{X}$, the Tabu Search evaluation function f is defined as follows:

$$f(\mathbf{X}) = \alpha \times N_{HC}(\mathbf{X}) + N_{SC1}(\mathbf{X}) + N_{SC2}(\mathbf{X}) + N_{SC3}(\mathbf{X}),$$

where

α is a parameter that varies from one iteration to the next,

N_{HC} = number of violated hard constraints in solution $\mathbf{X}$,

N_{SC1} = number of violated soft constraint #1 in solution $\mathbf{X}$,

N_{SC2} = number of violated soft constraint #2 in solution $\mathbf{X}$,

N_{SC3} = number of violated soft constraint #3 in solution $\mathbf{X}$.

N_{HC} is computed as follows :

- hard constraint 1: for each student S_k such that this constraint is violated (once or several times), count 1 in N_{HC} ;
- hard constraint 2: count 1 for each room that does not satisfy either the requirements or the cardinality for a given event E_i ;
- hard constraint 3: count 1 for each event such that this constraint is violated.

All numbers N_{SC1}, N_{SC2} and N_{SC3} are evaluated following the soft constraint penalty corresponding to the description of the timetabling competition problem.

The parameter α helps to control the infeasibility of the solution:

if the current timetabling solution is feasible, we decrease the value of α considering that we better concentrate on the soft constraints,

otherwise we increase the value of α in order to avoid the solution to become too much infeasible and make it difficult to come back in the feasible domain set.

The details of the computation of α and the evaluation function are described below.

Evaluation of the current solution X

Compute the number of violated constraints, i.e., $N_{HC}(X)$, $N_{SC1}(X)$, $N_{SC2}(X)$ and $N_{SC3}(X)$;
Determine α as follows:

If $N_{HC}(X) = 0$ then

$\alpha \leftarrow \min \{1, \alpha / \delta\}$ with δ randomly chosen in [1.05, 1.19]

else

$\alpha \leftarrow \max\{1000, \alpha \times \delta\}$ with δ again randomly chosen in [1.05, 1.19] ;

Compute $f(X) = \alpha \times N_{HC}(X) + N_{SC1}(X) + N_{SC2}(X) + N_{SC3}(X)$.

We can now describe the basic Tabu Search procedure.

Basic Tabu Search procedure TS-MOUSTIK

Let X^{init} be an initial feasible timetabling solution ;
Compute the values of all possible moves ;
Initialize the incumbent solution: $f^{opt} \leftarrow +\infty$; $X^{opt} \leftarrow X^{init}$;
Initialize the current timetabling solution: $X \leftarrow X^{init}$;
While the stopping criterion is not satisfied ***do***

- select the non tabu move $X \rightarrow X'$ such that $f(X')$ is minimum with an aspiration criterion such that if a tabu timetabling move leads to a better solution than the incumbent one, accept this move even if it is tabu ;
- update the incumbent solution:
  if $f(X') < f^{opt}$ then $f^{opt} \leftarrow f(X')$; $X^{opt} \leftarrow X'$;
- add X' to the tabu list for N_{TABU} iterations ;
- update the values of the moves whenever it is necessary ;
- update the tabu list

EndWhile.

Due to the rules of the timetabling competition, the stopping criterion corresponds to the computing time limit.

In practice, the N_{TABU} parameter is randomly selected in $[N_{moves} - 5, N_{moves} + 5]$, where N_{moves} is equal to the overall number of possible moves for the timetabling instance that is solved. We observe that, on the average, this entails that a given move may not be considered for about 2000 to 3000 iterations depending on the size of the timetabling instance, i.e., depending on the overall huge number of moves.

In order to reduce the computing time of each iteration, we do not recompute from scratch the value of each possible move at each iteration, but update those values from the previous iterations. This means that we first identify the moves such that their value has changed

considering the move of the current iteration, and that we next revise (i.e., update) only the values of this subset of moves.

5.2 Improvements of the Basic Tabu Search Algorithm

5.2.1. Exchange moves

Considering a given timetabling solution, there are many close solutions that satisfy the hard constraints. Indeed, let us consider two triplets $(R_{j_1}, P_{t_1}, E_{i_1})$ and $(R_{j_2}, P_{t_2}, E_{i_2})$ and the following exchange:

$$\underbrace{\left\{\begin{matrix}(R_{j_1}, P_{t_1}, E_{i_1}) \\ (R_{j_2}, P_{t_2}, E_{i_2})\end{matrix}\right\}}_{\text{before the exchange}} \xrightarrow{\text{exchange of room and period between two events}} \underbrace{\left\{\begin{matrix}(R_{j_2}, P_{t_2}, E_{i_1}) \\ (R_{j_1}, P_{t_1}, E_{i_2})\end{matrix}\right\}}_{\text{after the exchange}}.$$

We observe that if the timetabling solution was feasible before the exchange, it remains feasible after the exchange. Similarly, if the solution wasn't feasible before the exchange, the infeasibility changes after the exchange, but hopefully not so much. However, performing such exchanges allows some perturbation in the current timetabling solution, and possibly helps to improve the solution. We therefore include some exchange moves through the following scheme.

We consider the set of all possible pairs of triplet exchanges. Let L be the list of the n_L pairs of triplet that can be considered for an exchange move.

Procedure Exchange_moves

While some improvements are obtained ***do***
    For e from *1* to n_L ***do***
        Let X be the current timetabling solution ;
        Consider the L_e exchange move ;
        Let X' be the new timetabling solution if exchange L_e is performed
        If the infeasibility of X' is smaller than the infeasibility of X
            and $f(X') < f(X)$ ***then***
                $X \leftarrow X'$
    EndFor
EndWhile.

The procedure Exchange_moves is used every 5 or 10 iterations in the Tabu Search TS-MOUSTIK algorithm. Indeed, it is every 5 iterations, except when all the three following conditions are satisfied:

- Condition 1. There are events such that their number of students lies between 15 and 25 inclusively,
- Condition 2. $N_E(N_S^{\max}) + N_E(N_S^{\max-1}) \geq \mathrm{N_E} / 2$, where

  $N_S^{\max}$ is the maximum number of students participating to a given event,

  $N_E(N_S^{\max})$ is the number of events with $N_S^{\max}$ students, and

  $N_E(N_S^{\max-1})$ is the number of events with $N_S^{\max}$ -1 students.
- Condition 3. $N_E(N_S^{\max}) + N_E(N_S^{\max-1}) \geq \mathrm{N_E} / 2$, where

  $N_R^{\min}$ is the minimum number of rooms available for a given event,

  $N_E(N_R^{\min})$ is the number of events with $N_R^{\min}$ rooms, and

  $N_E(N_R^{\min-1})$ is the number of events with $N_R^{\min}$ -1 rooms.
-

5.2.2. Pertubation 1.

If we observe no improvement of the incumbent value during 3 consecutive iterations, we then use a first perturbation.

Consider the set of events for a given period. Apply a random permutation on the events such that, when replacing a given event by another one, the room requirements are still satisfied.

Observe that performing this perturbation does not change the value of the evaluation function *f*.

5.2.3. Pertubation 2

If no improvement after 200 iterations then

Restart with the incumbent value with a complete reset of the tabu list (i.e. empty the tabu list).

If still no improvement after another 200 iterations then

Restart with the incumbent value together with the tabu list associated with when it was found (the algorithm will not cycle as the tabu parameter is randomly defined).

If still no improvement then

Apply the ejection chain procedure.

5.2.4 Ejection chains

Let us define ALEA([10,20]) as a random integer between 10 and 20.

Procedure Ejection_Chain

N_{iter} ← ALEA([10,20]) ;
event_eject ← .false. ;
For N_{iter} iterations ***do***
If *event_eject* = .false. ***then***
Determine the event E_{eject} (i.e., the triplet R_j, P_t, E_i) that entails the best improvement of the evaluation function if we remove it from the current timetabling solution and eject this event from the current timetabling solution;
Reintroduce the ejected event E_{eject} at the best possible room/period with respect to the evaluation function, i.e., identify R'_j and P'_t leading to the best improvement of the evaluation function ;
If there is already an event $E_{i'}$ that takes place at period P'_t in room R'_j ***then***
event_eject ← .true. ;
$E_{eject} \leftarrow E_{i'}$
otherwise
event_eject ← .false.
EndFor.

References

1. R. Jonker and A. Volgenant, A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems, *Computing* 38 (1987) 325-340. C++ source available at http://www.magiclogic.com/assignment.html.