

# TimeTabling: Guided Simulated Annealing + Local Searches

R. Montemanni  
IDSIA  
Galleria 2, CH-6928 Manno, Switzerland  
roberto@idsia.ch

March 26, 2003

## 1 Introduction

The algorithm we present is an adaptation of the method developed for the Frequency Assignment Problem (FAP) and described in Montemanni [1]. Within the algorithm some different methods are run in order to search the solution space following different criterion. The different methods used within the algorithm are described in Section 2, while the full algorithm is described in Section 3. Implementation details are given in Section 4.

## 2 Procedures used within the algorithm

**Guided Simulated Annealing.** See Montemanni [1] for a description of the original Simulated Annealing algorithm. We modify the algorithm in order to control the temperature, which is in our case connected to the cost of the best solution found.

The initial temperature in our algorithm is:

$$\begin{aligned} &0.000009 * (1 + rand()\%16) * \text{Starting solution cost} * \\ & * (\text{number of events in the problem} / 100) * \\ & * \text{average suitable rooms per event} * ((500 + rand()\%1000) / 1000) \end{aligned} \quad (1)$$

At each iteration a random event  $e$  is selected, together with a random suitable room  $r$  for it and a random time slot  $t$ . The solution obtained by assigning  $e$  to  $r$  and  $t$  becomes the new current solution if:

$$rand()\%1000 / 1000 < e^{-(\Delta\text{cost}) / (\text{temperature})} \quad (2)$$

Otherwise the new assignment is rejected.

Once a new best solution is found and  $(rand()\%1000) < 250$  the procedure 1-OPT described below is run.

Every 20000 iterations of the algorithm the temperature is updated through the following formulae:

- If no improvement occurred in the last 1000000 iterations:

$$\text{temperature} = \text{temperature} * 0.97 \quad (3)$$

- Otherwise:

$$\begin{aligned} \text{temperature} = & 0.000009 * (1 + \text{rand}()\%16) * \text{Best solution cost} * \\ & *(\text{number of events in the problem} / 100) * \\ & * \text{average suitable rooms per event} * ((500 + \text{rand}()\%1000) / 1000) \quad (4) \end{aligned}$$

The procedure stops when it has run for maxtime seconds or when 3000000 iterations without improvement have been carried out.

**1-OPT.** Iteratively a random even  $e$  is selected and all the possible solutions obtained by combining feasible rooms for it and time slots are checked. Every time a solution improving the current one is encountered it becomes the new current solution.

The procedure stops when it has run for maxtime seconds or when  $400 + \text{rand}()\%300$  iterations without improvement have been carried out.

**2-OPT.** Iteratively a random event  $e1$  is selected and each possible time slot  $t1$  is selected for it. Every even  $e2$  colliding with  $e1$  in case this is assigned to  $t1$  is considered. Every combination of new time slot  $t2$  for  $e2$  and rooms  $r1$  and  $r2$  for  $e1$  and  $e2$  respectively are considered and for each of them new solution cost is evaluated. If a solution better then the current one is encountered, it becomes the new current solution.

The procedure stops when it has run for maxtime seconds or when there is no improvement for noimprove seconds.

**Even-Swap.** Two random events  $e1$  and  $e2$  are selected and their room and time slot allocations are swapped.

The procedure stops when it has run for maxtime seconds or when there is no improvement for noimprove seconds.

**Tslot-Swap.** Two random time slots  $t1$  and  $t2$  are selected and all the events which take place in  $t1$  are swapped with those taking place in  $t2$ . Rooms assigned to events are maintained.

The procedure stops when it has run for maxtime seconds or when there is no improvement for noimprove seconds.

### 3 Pseudo-code

The algorithm we propose can be summarized with the help of the following pseudo-code, where totaltime is the maximum computation time allowed:

```
maxtime = 0.81 * totaltime;
inizio = time();
create a starting solution by assigning events to random suitable rooms and
time slots;
While(time() - inizio < maxtime)
  If (first iteration or rand() % 1000 < 100)
    run Guided Simulated Annealing;
  sel = rand() % 10000;
```

```

    if (sel < 2500) run 1-OPT;
    if (2500 ≤ sel < 5000) run Even-Swap;
    if (5000 ≤ sel < 7000) run Tslot-Swap;
    if (7000 ≤ sel < 10000) run 2-OPT;
maxtime = totaltime;
While(time() - inizio < maxtime)
    sel = rand() % 10000;
    if (sel < 2000) run Even-Swap;
    if (2000 ≤ sel < 3000) run Tslot-Swap;
    if (3000 ≤ sel < 10000) run 2-OPT;
return the best solution found;

```

## 4 Implementation details

We maintain a table of dimension number of events  $\times$  number of rooms  $\times$  number of time slots, called the cost change table, where position  $(e, r, t)$  contains the partial cost (students with more than 2 consecutive classes in a day and students with only one class in a day are not counted here) of the solution obtained by moving event  $e$  to room  $r$  at time slot  $t$ . Each time a move is carried out, the elements of the table affected by the move are updated accordingly. Cost concerning students with more than 2 consecutive classes in a day and students with only one class in a day is calculated every time a complete solution cost has to be evaluated. This extra cost has to be added to the one contained in the cost change table.

## References

- [1] R. Montemanni, “Upper and lower bounds for the fixed spectrum frequency assignment problem,” PhD thesis, University of Glamorgan, November 2001.