Tabu search techniques for large high-school timetabling problems

A. Schaerf

Computer Science/Department of Interactive Systems

CS-R9611 1996

Report CS-R9611
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Tabu Search Techniques for Large High-School Timetabling Problems

Andrea Schaerf
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
e-mail: `aschaerf@cwi.nl`

Abstract

The high-school timetabling problem regards the weekly scheduling for all the lectures of a high school. The problem consists in assigning lectures to periods in such a way that no teacher (or class) is involved in more than one lecture at a time, and other side constraints are satisfied. The problem is NP-complete and is usually tackled using heuristic methods. This paper describes a solution algorithm (and its implementation) based on *tabu search.* The algorithm interleaves different types of *moves* and makes use of an adaptive relaxation of the hard constraints. The implementation of the algorithm has been successfully experimented in some large high schools with various kinds of side constraints.

AMS Subject Classification: 68M20, 68T20
CR Subject Classification: I.2.8, F.2.2, G.2.1
Keywords & Phrases: timetabling, combinatorial optimization, scheduling, local search techniques, tabu search
Note: This work has been carried out as part of the ERCIM fellowship Programme and financed by the Commission of the European Communities. An abridged version of this work will appear in the proceedings of the 13^{th} National Conference of the American Association for Artificial Intelligence (AAAI-96).

1 Introduction

The high-school timetabling problem regards the weekly scheduling for all the lectures of a high school. The problem consists in assigning lectures to periods in such a way that no teacher (or class) is involved in more than one lecture at a time, and other side constraints are satisfied.

The manual solution of the timetabling problem usually requires several days of work. In addition, the solution obtained may be unsatisfactory to some respect; for example, a teacher may be idle for one hour between two lectures.

For the above reason, a considerable attention has been devoted to automated timetabling. During the last thirty years, starting in the early 60's with Gotlieb (1963) and others, many papers related to automated timetabling have appeared in conferences and journals, and several applications have been developed and employed with a quite good success.

Most of the early techniques (see Schmidt & Strohlein, 1979; Junginger, 1986) were based on a simulation of the human way of solving the problem. All such techniques, that we call *direct heuristics*, were based on a *successive augmentation.* That is, a partial timetable is filled in, lecture by lecture, until either all lectures have been scheduled or no lecture can be scheduled without violating the constraints.

Later on, researchers started to apply general techniques to this problem. Then, we see algorithms based on *integer programming* (Tripathy, 1984, 1992), *network flow* (Ostermann & de Werra, 1983), and others. In addition, the problem has also been tackled by reducing it to a well-studied theoretical problem: *graph coloring* (Neufeld & Tartar, 1974).

More recently, some approaches based on new search techniques appeared in the literature; among others, we have *simulated annealing* (Abramson, 1991), *tabu search* (Costa, 1994), *genetic algorithms* (Colorni, Dorigo, & Maniezzo, 1992), *constraint satisfaction* (Yoshikawa, Kaneko, Nomura, & Watanabe, 1994), and combination of different methods (Cooper & Kingston, 1993).

The problem has a large number of variants, depending on the country, on the type of school, and even on the specific school involved, see (Schaerf, 1995). The problem we tackle comes from the Italian high-school system, and it is a slight modification of the theoretical problem defined by Gotlieb (1963) and others. In some countries, e.g. Germany (see Junginger, 1986), the high school is organized in a similar way, and so the solution proposed in this paper applies as well; in some others, e.g. Holland (see Schreuder & Visscher, 1995), student are more free to choose a certain number of subjects, and so it is organized more like a college, and different algorithms and techniques apply.

The algorithm we present in this paper is based on *local search* (or *neighborhood search*) and, more specifically, it is an adaptation of tabu search, with some modifications.

Our algorithm is suitable for both interactive and batch run. That is, it allows the user for manual modifications of the timetable and the constraints during the search phase. The importance of the interactivity of the system has been stressed by several authors (see e.g. Chahal & de Werra, 1989).

The paper is organized as follows: Section 2 defines the high-school timetabling problem. Section 3 introduces local search techniques in general, and the tabu search in particular. Section 4 describes the way we represent the search space and the neighborhood relation. Section 5 explains in details the algorithm employed. Section 6 shows the experimental results and the performances obtained. Finally, in Section 7 we draw some conclusions.

2 High-School Timetabling Problem

The problem we deal with is an optimization problem, and it is therefore defined through a solution space and an objective function. In Section 2.1 we define the solution space by introducing the underlying search problem, in Section 2.2 we add to it our objective function.

2.1 Underlying Search Problem

There are m classes $c_1, \ldots, c_m$, n teachers $t_1, \ldots, t_n$, and p periods $1, \ldots, p$. It is given a non-negative integer matrix $R_{m \times n}$, called *Requirements matrix*, where r_{ij} is the number of lectures that teacher t_j must give to class c_i. The unavailabilities of teachers and classes are taken into account by introducing two binary matrices $T_{n \times p}$ and $C_{m \times p}$ such that $t_{jk} = 1$ (resp. $c_{ik} = 1$) if teacher t_j (resp. class c_i) is available at period k, and $t_{jk} = 0$ (resp. $c_{ik} = 0$) otherwise. The mathematical formulation of the problem is the following (see de Werra, 1985; Junginger, 1986)

$$\boxed{\text{TTP}} \qquad \textit{find} \quad x_{ijk} \qquad (i = 1, \ldots, m; j = 1, \ldots, n; k = 1, \ldots, p)$$

$$s.t. \quad \sum_{k=1}^{p} x_{ijk} = r_{ij} \qquad (i = 1, \ldots, m; j = 1, \ldots, n) \tag{1}$$

$$\sum_{i=1}^{m} x_{ijk} \leq t_{jk} \qquad (j = 1, \ldots, n; k = 1, \ldots, p) \tag{2}$$

$$\sum_{j=1}^{n} x_{ijk} \leq c_{ik} \qquad (i = 1, \ldots, m; k = 1, \ldots, p) \tag{3}$$

$$x_{ijk} = 0 \text{ or } 1 \qquad (i = 1, \ldots, m; j = 1, \ldots, n; k = 1, \ldots, p) \tag{4}$$

The TTP has been shown NP-complete by Even, Itai, and Shamir (1976) and it appears also in (Garey & Johnson, 1979, SS19, p. 243).

In order to deal with real instances of the problem, we tackle a variant of TTP. In particular, we add two other types of constraint.

First, each class, for a given set of periods, must necessarily be involved in (at least) one lecture. This constraint can be expressed as follows

$$\sum_{j=1}^{n} x_{ijk} \geq d_{ik} \qquad (i = 1, \ldots, m; k = 1, \ldots, p) \tag{5}$$

where $D_{m \times p}$ is a binary matrix such that $d_{ik} = 1$ if class c_i must necessarily be taught at time k, and $d_{ik} = 0$ otherwise. Constraints 5 are very crucial for high-school timetabling, and they represent one of the major differences between high-school timetabling and university timetabling. In fact, they require the timetable to be completely filled in, which is a constraint genuinely hard to satisfy. The set of k's for which $d_{ik} = 1$ usually comprises all the periods but either the last one of each day or the last two, depending on the total requirements of the specific class c_i.

Second, some pairs of lectures require to be scheduled simultaneously. Specifically, there exists a set of quadruples $\langle i_1, j_1, i_2, j_2 \rangle$ such that all lectures of teacher t_{j_1} to class c_{i_1} must be simultaneous to lectures of teacher t_{j_2} to class c_{i_2}. Calling S such set a quadruples, the above constraints can be expressed as follows.

$$x_{i_1 j_1 k} x_{i_2 j_2 k} + \overline{x}_{i_1 j_1 k}\, \overline{x}_{i_2 j_2 k} = 1 \qquad (\langle i_1, j_1, i_2, j_2 \rangle \in S; k = 1, \ldots, p) \tag{6}$$

where $\overline{x}$ denotes the complement of the binary variable x; that is, $\overline{x} = 1$ if $x = 0$, and vice versa.

Constraints 6 also are necessary for tackling practical cases. In fact, they turned out to be a quite general mechanism using which it is possible to model various features that are usually present in actual schools: laboratory assistants, shared gymnastic rooms, bilingual classes (i.e. two foreign languages taught at the same time to different students of a single class), and others. All such features, although they may require either a single class or a single teacher, can always be reduced to constraint of the above form introducing dummy classes.

2.2 Objective Function

The computation of the objective function presuppose the definition of a number of items for teachers, classes, and teacher/class pairs. In particular, for each teacher we define: minimum and maximum number of teaching periods per day, undesired teaching periods, and seniority. For each class we define the site in which its class room is located. For each pair teacher/class we define: maximum number of lectures per day, and length of the class work (i.e. minimum number of lectures that must be given consecutively in one day, at least once a week).

Our objective function is a weighted sum of the following components, which refer to the schedule of a single teacher, and are summed up for all teachers. In brackets we put the default penalty weight, which can be modified for specific schools (some of the features may be not significant for some of the specific schools, in that case their weight is set to 0).

Holes [1]: Idle periods between two lectures in the teaching assignments of a day.

Splits [6]: Two lectures to a class separated by one or more lectures to different classes.

Under_use [4]: Number of teaching periods in a day less than the minimum specified for the teacher.

Over_use [3]: Number of teaching periods in a day more than the maximum specified for the teacher.

Clusters [6]: More lectures to the same class in the same day than the maximum specified for the pair teacher/class.

Undesired [3]: Teaching in an undesired period. The penalty of this feature is normalized based on the number of requests of each teacher and multiplied by the seniority of the teacher.

Class_work [10]: Not having (at least once a week) enough joint teaching periods to run the class work (or the practical class in the laboratory) for the specific teacher/class pair.

Commutations [5]: Moving from one site to another between two consecutive teaching periods.

In order to compute correctly some of the above components (i.e. holes, over_use, under_use), we need to split the unavailabilities of teachers into two kinds: *out-school* and *in-school*. The out-school ones are the ones in which the teacher cannot teach because he/she is away from the school, the in-school ones represent the situation in which the teacher is at school but he/she is not available to teach (e.g. for administrative work or professional development).

The in-school unavailabilities can also be used (in conjunction with class unavailabilities) to model teaching *preassignments*, which are not explicitly included in our model.

3 Local Search and Tabu Search

Local search techniques are a family of general-purpose techniques for the solution of optimization problems. They are based on the notion of *neighbor*. Consider an optimization problem, and let S be its search space and f its objective function to minimize. A function N, which depends on the structure of the specific problem, assigns to each feasible solution $s \in S$ its *neighborhood* $N(s) \subseteq S$. Each solution $s' \in N(s)$ is called a neighbor of s.

A local search technique, starting from an initial solution s_0 (which can be obtained with some other technique or generated at random), enters in a loop that *navigates* the search space, stepping iteratively from one solution to one of its neighbors. We call *move* the modification that transforms a solution to one of its neighbors.

Among the local search techniques, we have the *steepest descendent method*: It analyses all the possible moves and chooses the one giving the best improvement. It accepts the candidate move only if it improves the value of the objective function, and it stops as soon as it reaches a local minimum.

The above method requires the exploration of the whole neighborhood. The *randomized descendent method* instead analyse a random neighbor and accepts it if it is better of the current one, otherwise the current solution is left unchanged and another neighbor is generated. It stops after a fixed number of iterations without improving the value of the objective function.

The randomized descendent method is also trapped in the very first local minimum it gets. The *randomized non-ascendent method* (RNA) instead accepts the random neighbor if it is better *or equal* to the current one and it also stops after a fixed number of iterations without improving the value of the objective function. Allowing also for *sideways moves* (as they are called in Selman, Levesque, & Mitchell, 1992), this method has the feature of being able to follow descending paths that pass through *plateaus*. That is, if the search lands in a plateau, it is able to move within it, and might get down from it through a solution different from the one from which it reached the plateau.

The *steepest non-ascendent method* combines the search for the steepest move with the use of *sideways moves*. This method, and some variants of it, have been used for the SAT problem with the name GSAT (Selman et al., 1992; Selman, Kautz, & Cohen, 1994). A characteristic of GSAT is that the choice between two equally descending moves is randomized.

Simulated annealing (van Laarhoven & Aarts, 1987) accepts always the randomly generated neighbor if it is better or equal than the current solution, and accepts it if it is an up-going move with a certain probability. Such probability depends on the increase of the objective function and on a parameter (called *temperature*) which decreases with the number of iteration performed. We will not consider simulated annealing in this paper. We experimented with it in the preliminary work (Schaerf & Schaerf, 1995), which shows a quite clear dominance of tabu search over simulated annealing for high-school timetabling.

3.1 Tabu Search

We now briefly describe the basic principles of tabu search (TS). We refer to (Glover, 1989, 1990; Glover & Laguna, 1993) for a comprehensive presentation.

Starting from the initial solution s_0, the TS algorithm iteratively explores a subset V of the neighborhood $N(s)$ of the current solution s; the member of V that gives the minimum value of the objective function becomes the new current solution independently of the fact that its value is better or worse than the value in s.

In order to prevent cycling, there is a so-called *tabu list*, which is the list of moves which is forbidden to make. This is the list of the reverse of the last k accepted moves (where k is a parameter of the method) and it is run as a queue of fixed size; that is, when a new move is added, the oldest one is discarded.

There is also a mechanism that overrides the tabu status of a move: If a move gives a *large* improvement of the objective function, then its tabu status is dropped and the resulting solution is accepted as the new current one. More precisely, we define an *aspiration function* A that, for each value v of the objective function, returns another value v' for it, which represents the value that the algorithm *aspires* to reach from v. Given a current solution s, the objective function f, and a neighbor solution s' obtained through the move m, if $f(s') \leq A(f(s))$ then s' can be accepted, even if m is in the tabu list.

The procedure stops either after a given maximum number of iterations without improvements or when the value of the objective function in the current solution reaches a given lower bound.

The main control parameters of the procedure are the length of the tabu list k, the aspiration function A, the cardinality of the set V of neighbor solutions tested at each iteration, and `TSmax`, the maximum number of iterations without improving the objective function.

3.2 Tabu Search for Interactive Timetabling

Local search techniques, giving the possibility to start the search from any timetable, easily allow for interactive construction and maintenance of timetables. In fact, once a timetable as been generated, it can be used as the starting point for a new search after some constraints (or the timetable itself) have been manually modified.

As an example, suppose that after finding a (good) timetable the requirement matrix is slightly altered by increasing the requirement of a given teacher to a given class. Using our method we can add the missing lectures manually to the timetable (at any period) and restart the search. Such process generally leads to a good timetable for the new problem in a short amount of time. As another example, if a teacher leaves a school and another one takes his/her place, and the incoming one has different unavailabilities and desiderata, it is still possible to modify the configuration and start from the current timetable and improve it using our algorithm.

The ability to work interactively is widely recognised as crucial for timetabling systems in the research community. To this respect, local search has a great advantage upon other methods, such as constructive ones and genetic algorithms.

4 Representation of the Problem

A timetable is represented as an integer-valued matrix $M_{m \times p}$ such that each row j of M represents the weekly assignment for teacher t_j. In particular, each entry m_{jk} contains the name of the class that teacher t_j is meeting at period k. The value $m_{jk} = 0$ represents the fact that t_j is not teaching at period k. Values larger than the number of classes m represent special activities, such as being at disposal for temporary teaching posts, teacher-parents meetings, and assisting assignments.

We choose this representation because it allows for the definition of simple and natural types of moves (see Section 4.2), which permit to navigate effectively the search space. In addition, this is generally the representation upon which the persons that do manual timetabling reason, and therefore, it is also suitable for interactive timetabling with manual corrections.

This representation has been used also by Colorni et al. (1992). Conversely, Carmusciano and De Luca Cardillo (1995) use a dual representation with a matrix $N_{n \times p}$ such that each entry n_{ik} stores the name of the teacher that teaches to class c_i at time k. Such dual representation allows for a more compact storing of the timetable, since n is generally lower than m; however, in our experience, it makes more difficult to compute the various features of the objective function, which are more based on the scheduling of the single teachers that on the scheduling of the single classes.

Figure 1 shows a fragment of a real timetable. For the sake of readability, in the external representation the class numbers are converted into their names ("1A", "2A", ...); the symbol "<>" represents periods in which the teacher is assigned to be at disposal for possible supply teaching. The figure also shows the unavailabilities of teachers, where "--" represents out-school ones and "**" represents in-school ones.

The lowercase class names represent dummy classes which are used for simultaneous assignments (Constraints 6). For example, in Figure 1 teacher 13 is an English teacher that teaches "First Foreign Language" to the *bilingual* class 2E being assisted by the French teacher 14 who takes care of the students than take French as first foreign language; therefore, when teacher 13 teaches to class "2E" (Wednesday second period, in Figure 1), teacher 14 must teach to the dummy class "2e". Conversely, "Second Foreign Language" is taught to class 2E by the French

```
---------------------------------------------------------------------------------
nr  |teacher     |       Monday       |      Tuesday       |     Wednesday      |
---------------------------------------------------------------------------------
3   |parenti     |       3A 3A <> 2A |                <> 2A | -- -- -- -- -- -- | ..
4   |scocchera   | 1A 1A 4A 5A 5A -- | 1A 4A 4A <>    -- | <> 1A          -- | ..
5   |mariani     | 2B 2B 2B <> 1B    | 2B 3B             |    2B <> <> 3B 1B | ..
6   |marcattili  | 1B 1B 4B 4B       | <> 1B ** 4B 5B    | -- -- -- -- -- -- | ..
7   |giacchetti  | 1C <> 3C 2C       |       <> 3C 3C 2C | 3C <> 2C 2C 2C    | ..
8   |mori        | 5C 4C 4C <> 1C    |          5C 1C 1C | 4C 4C <> 5C       | ..
9   |saconis     | 1D <> <> 2D 2D    |             2D 2D | 1D 1D             | ..
10  |gabriele    | -- -- -- -- -- -- | 2E 2E 3D 3D 5D    | 5D 5D 2E 3D       | ..
11  |giagnoro    | 3A 2A             |    5A 3A 2A 1A 4A | 1A <> 2A          | ..
12  |greco       | -- -- -- -- -- -- |    2C 3C 2B 1B 3B | 1C 3C    3B 1B 2B | ..
13  |ranieri     |    2d 2D 1D 3D 2e |          1D 3D    | 3D 2E 2d 2e       | ..
14  |spadoni     | -- -- -- -- -- -- |             3A 4a |    2e <> 2A 1A 3A | ..
15  |di_pofi     |          2B 4B    |          1B 3B 5B | 2B 5B 4B          | ..
16  |meloni      | <> <> 5C 1C 4C    | -- -- -- -- -- -- | 2C 1C 3C          | ..
17  |valle       | -- 2D 2d 3D 1D 2E | -- -- -- -- -- -- | --    2D 2E 5D    | ..
---------------------------------------------------------------------------------
```

Figure 1: A fragment of a timetable

teacher 17, which is assisted by teacher 13 for the students that take French as first foreign language (and English as second one).

4.1 Embedding Infeasibilities in the Objective Function

Infeasible timetables are also included in the search space of the algorithms. The objective function described in Section 2 is augmented so as to embed also the number of infeasibilities. In particular, we count: (1) the number of times that either two teachers teach to the same class in one period or a class is uncovered (2) the number of times a simultaneous assignment is missed, and (3) the number of times a teacher (a class) teaches (is taught) when he/she (it) is not available.

The possibility that a teacher teaches simultaneously to two (or more) classes is ruled out automatically in the representation chosen.

The weight W given to the infeasibilities is set to a value ($W = 20$) higher than the weight of all other quantities. However, as explained in Section 5, in order to ensure a better navigation of the search space, such weight is allowed to vary during the search phase.

4.2 Types of Move and Neighborhood Structure

The first type of move that we consider is the one that naturally fits in our representation. It is obtained by simply swapping two distinct values on a given row. That is, the lectures of a teacher t in two different period p_1 and p_2 are exchanged between them, or, in case that one value is 0, one lecture is moved to a different period. We call a move of such type *atomic move*, and we identify it through the triple $\langle t, p_1, p_2 \rangle$. For simplicity, we assume always $p_1 > p_2$, which makes a move to be the (unique) reverse of itself.

Atomic moves applied to feasible timetables generally create infeasibilities assigning two teachers to the same class. For this reason, we also consider more complex move types. In particular, we consider *double moves*, which are moves made by a pair of atomic moves, so that the second one "repairs" the infeasibility (or one of the infeasibilities) created by the first one by exchanging the lecture that conflicts with the one just inserted. If the first atomic move creates no infeasibilities, then there is no second move and the double moves reduces to an atomic one.

It can be easily shown that the search space is connected under the neighborhood relation in both cases of atomic and double moves. In particular, a timetable can be reached by any other one in a number of moves (either atomic or double) which is at most equal to the total number of lectures (i.e. the sum of all elements of the matrix R).

Costa (1994) employs a different type of move. That is, he allows only for the reassignment of a single lecture to a different period. In his representation, however, a single teacher can teach more than one lecture at the same time, therefore a swap of assignments for a single teacher can be done in two consecutive moves letting both assignment in the same period at the intermediate step. We don't follow such choice because, in our case, it would be difficult to define an effective objective function (which is very much based on single teacher assignments) in presence of multiple assignments for a single period.

5 Application of the Tabu Search

Our algorithm is basically a TS with the neighborhood constituted by atomic moves. However, the TS is interleaved with a phase of RNA using double moves. The idea of alternating simple and complex moves has been suggested, as a *tactical improvement*, by Glover, Taillard, and de Werra (1993).

In details, the initial solution is obtained by scheduling the lectures for each teacher randomly, respecting the requirement matrix (or it can be obtained from previous runs, in case of interactive timetabling). Then, the RNA starts to work on the random timetable as far as it makes no improvements for a given number of iterations (`RNAmax`). At this point, the TS starts and goes on until it makes a given number of iterations without improving (`TSmax`). The whole process (RNA + TS) is repeated on the best solution found, and it stops when it gives no improvements for a given number of times (`Cycles`).

The RNA is used for two different purposes: First, it generated the initial solution for the TS. In fact, the use of the TS starting from the random solution is too time consuming, and RNA instead represents a fast method to generate a reasonably good initial solution (which generally still contains a few infeasibilities). Second, after the TS has given no improvements for a given number of iterations, it is useful to run the RNA on the best solution found. The reason for it is twofold: On the one hand, the RNA (using double move) might find improvements that the TS is not able to find at that stage. On the other hand, the RNA with double moves, even if it does not improve the solution (which is often the case), it makes substantial sideways modifications. Therefore, it "shuffles" the solution before the TS starts again to try to improve it. The idea underlying such procedure is that after the TS has worked unsuccessfully for a given number of iterations, it is useful to modify the solution so that the TS can start in a different direction.

The algorithm is shown in Figure 2. In the following subsections, we discuss a number of features of it.

5.1 Adaptive Relaxation

During the RNA phase the weight of the infeasibilities is set to the value $W = 20$, which is higher than all the other weights involved in the objective function. Conversely, during the TS stage, such weight is dynamically adjusted in the following way (as proposed in Gendreau, Hertz, & Laporte, 1994): For each of the three sources of infeasibilities (see Section 4.1), namely (1) clash

```
Algorithm High School TimeTabling
begin
    GetInitialSolution;    (* generated at random or from input file *)
    InitializeSearchVariables;
    for c := 1 to Cycles
    do begin
        NoChanges := 0;
        while (NoChanges < RNAmax)    (* make the RNA phase *)
        do begin
            NoChanges := NoChanges + 1;
            DoubleMove := RandomDoubleMove;
            if delta(DoubleMove) ≤ 0
            then begin
                UpdateCurrentTimeTable(DoubleMove);    (* make the move *)
                if delta(DoubleMove) < 0 then NoChanges := 0
            end
        end;
        NoChanges := 0;
        while (NoChanges < TSmax)    (* make the TS phase *)
        do begin
            NoChanges := NoChanges + 1;
            BestMove := AtomicRandomMove;
            BestDelta := delta(BestMove,DynamicWeights);
            forall Move in LegalAtomicNeighborhood(CurrentTimeTable)
            begin if delta(Move,DynamicWeights) < BestDelta
                and (not Tabu(Move) or Aspiration(Move))
                and (delta(Move,DynamicWeights) < 0 or not SemiIllegal(Move))
                then begin
                    BestMove := Move;
                    BestDelta := delta(Move,DynamicWeights)
                end
            end;
            UpdateCurrentTimeTable(BestMove);    (* make the move *)
            UpdateTabuList(BestMove);
            if IsBest(CurrentTimeTable) then BestTimeTable := CurrentTimeTable;
            if UpdateIteration then UpdateDynamicWeights
        end;
        CurrentTimeTable := BestTimeTable
    end
end
```

Figure 2: Timetabling Algorithm

of teachers or uncovered classed, (2) simultaneity of lectures, and (3) unavailability, we multiply W by a real-valued factor α_i (for $i = 1, 2, 3$) which varies according to the following scheme:

1. At the beginning of the search we set $\alpha_i := 1$.
2. Every k moves (with $k = 10$ in our experiments):
    - if all the k solutions visited are feasible w.r.t. the infeasibility (i) then $\alpha_i := \alpha_i/\gamma$;
    - if all the k solutions visited are infeasible w.r.t. the infeasibility (i) then $\alpha_i := \alpha_i \cdot \gamma$;
    - if some solutions are feasible and some others are infeasible then α_i is left unchanged.

The real-valued parameter γ is randomly selected (at each time) in the interval $[1.8, 2.2]$. In (Gendreau et al., 1994), γ is deterministically set to 2; we prefer to randomize such value to avoid that deterministic ratios between different components of the objective function could bias the search.

We bound the value of each α_i by two constants $\alpha_{i,min}$ and $\alpha_{i,max}$. That is, if α_i gets a value higher than $\alpha_{i,max}$, then it is set to $\alpha_{i,max}$. Similarly, when it goes below $\alpha_{i,min}$, it is set to $\alpha_{i,min}$. The value of α_i is limited to prevent it to get too high (resp. too low) after a long sequence of infeasible (resp. feasible) solutions. In fact, in that case, it would take too long to get back to values comparable to the other values in the objective function when such sequence is interrupted. In addition, the three constants $\alpha_{i,max}$ are set to different values (in our experiments $\alpha_{1,max} = 1$, $\alpha_{2,max} = 10$, $\alpha_{3,max} = 3$, and $\alpha_{i,min} = 0.01$, for $i = 1, 2, 3$), so that there is no stable situation in which different sources of infeasibilities coexist.

5.2 Tabu List Management

We employ a tabu list of variable size. In particular, each performed move is inserted in the tabu list together with the number of iterations I it is going to be in the list. The number I is randomly selected between two given parameters I_{min} and I_{max} (with $I_{min} \leq I_{max}$). Each time a new move is inserted in the list, the value I of all the moves in the list is updated (i.e. decremented), and when it gets to 0, the move is removed.

We enforce two different tabu mechanisms, although we make use of a single tabu list. The first mechanism states that a move $m = \langle t, p_1, p_2 \rangle$ is tabu if the exact triple $\langle t, p_1, p_2 \rangle$ appears in the tabu list. The second mechanism, which has a short term effect, states that m is tabu if either (t, p_1) or (t, p_2) appear in the last-inserted I_{st} elements (with $I_{st} < I_{min}$) of the tabu list. In other words, the assignments of teacher t at times p_1 and p_2 cannot be swapped for I iterations and they cannot be singularly exchanged with any other assignment for the shorter period of I_{st} iterations.

5.3 Aspiration Function

Due to the use of the adaptive relaxation, the value reached from a given solution depends on the current values of the α's. In order to have a meaningful aspiration criterion, we base the definition of the aspiration function on the value of the non-relaxed objective function, i.e. the one obtained setting $\alpha_i = 1$ for $i = 1, 2, 3$.

We use the simplest standard aspiration function, which accepts a tabu move only if it improves the best current solution. That is, we set $A(f(s))$ equal to $f(s^*) - 1$ for all solutions s, where s^* is the current optimum.

We experimented also with more complex aspiration functions; however, mostly due to the fact that they are based on the non-relaxed objective function and not on the current one, they did not give any improvement to the method.

5.4 Neighborhood Exploring

The size of the neighborhood of any solution s, is given by

$$\mid N(s) \mid = \frac{mp(p-1)}{2}$$

which is the number of teachers times the number of unordered pairs of distinct periods. However, moves that swap two identical lectures do not actually change the timetable; therefore they are considered *illegal* and they are not included in the neighborhood. This fact decreases the number of *legal* moves by a factor that depends on the number of classes n and on the requirement matrix R (in practical cases it has the value of approximately 30%).

Due to the adaptive relaxation it is very difficult to find a way to predict the most promising moves. For this reason we choose to analyze the entire neighborhood at each iteration. However, there are certain moves, e.g. exchanging a non-teaching period with the assignment of being at disposal for possible supply teaching ("<>"), that in many situations do not really affect the structure of the solution. The existence of a large number of such moves gives the possibility to find a move with minimal effect that gives a zero-cost variation. Therefore, the TS does not choose up-going moves (unless we use a very long tabu list), and this fact prevents the TS from exploring effectively the search space. On the other hand, such moves might produce an improvement in the objective function (e.g. by filling holes). For the above reason, we define the concept of *semi-illegal* moves which intuitively are those moves that have a small effect on the structure of the timetable. The constraint we impose on semi-illegal moves is that they can be made only if they *strictly* improve the (current) objective function. We consider semi-illegal those moves that exchange two values such that both of them represent not real classes (e.g. they are non-teaching, assisting, or at disposal) and those that move classes between two periods in both of which they are not requested to be at school. In practical cases, the number of semi-illegal moves is about 10-15% of the total legal ones.

6 Experimental Results

In this section we present our experimental results. All the code has been implemented (by the author) in standard C++ and it runs on a Silicon Graphics workstation INDY. It is composed of about 2500 lines of code, 1000 of which are dedicated to the computation of the objective function (and its variation), 500 to the search procedures, 500 to the timetable management and input/output, and 500 to the general program management. All data structures are implemented using the LEDA public C++ library (Näher & Uhrig, 1995).

We experimented with three schools, which differ to each other for number of lectures and type (and quantity) of constraints.

The first school is a small dummy one, that we specifically constructed to use it as a test example on which it is possible to run a massive number of trials (each run costs about 5 minutes of CPU time). On such school, we experimented with a large number of combinations of values of different parameters so as to understand how they interplay. This school already creates difficulties in its manual timetabling despite its reduced size.

I	$I_{st} = 0$			$I_{st} = 1$			$I_{st} = 2$		
	Best	Avg	Dev	Best	Avg	Dev	Best	Avg	Dev
5	1	9.1	5.3	0	6.5	3.9	2	9.2	3.3
10	0	6.6	3.5	0	4.9	3.2	1	8.6	3.3
15	0	4.8	3.7	0	5.1	3.6	2	8.1	2.9
20	0	3.8	2.7	2	4.9	3.0	3	9.6	3.7
25	0	3.4	2.3	2	5.4	2.2	3	9.5	3.9
30	0	4.5	3.4	1	6.1	3.2	2	10.6	5.2
40	1	4.7	2.5	2	7.4	3.8	2	12.3	5.6
50	0	5.1	2.6	3	7.8	4.3	4	12.4	4.9

Table 1: Results varying I ($I = I_{min} = I_{max}$) and I_{st} [20 trials with `TSmax` = 2000, `RNAmax` = 0, `Cycles` = 1]

The following two are big real schools on which the timetabling is a tough problem that takes several days of manual work. On these schools, we obviously had to re-tuned some of the parameters based on the size and the peculiarities of the specific schools.

6.1 School 1

This school has 3 classes and 13 teachers (11 of which are part time ones). Each class takes 32 lectures weekly within 36 teaching periods in 6 days of 6 periods each. All classes are required to be at school for all periods but the last of each day (i.e. $d_{ik} = 0$ for $k = 6, 12, 18, 24, 30, 36; i = 1, 2, 3$, and $d_{ik} = 1$ otherwise) and all of them are not available for the last period of Saturday (i.e. $c_{ik} = 0$ for $k = 36; i = 1, 2, 3$ and $c_{ik} = 1$ otherwise). Each teacher has a day-off, and there are several other teacher's unavailabilities.

The neighborhood of each solution comprises 3803 legal moves. The number of double moves depends on the specific solution; for feasible timetables it is about 46000. The number of semi-illegal moves also depends on the specific solution and it is about 500.

Tables 1 and 2 highlight the importance of the parameters related with the tabu list. The results on those tables are obtained using the TS in isolation without interleaving it with the RNA (i.e. `RNAmax` = 0). This way the effects of the TS parameters on the overall performance are emphasized. The results on the tables are based on 20 trials, each one starting from a different random solution. For each set of 20 trials, the tables show the best result, the average and the standard deviation.

Table 1 shows the results obtained for a fixed tabu length (i.e. for $I_{min} = I_{max} = I$), for various values of I and I_{st}. The best results are obtained with the values $I = 25$, $I_{st} = 0$. This shows that the short term tabu status is useless for this school, however (as shown later) it has a role for larger schools. It also highlights the way I interplays with I_{st}, in particular for higher values of I_{st} the best results are obtained for lower values of I. This means that if one tabu mechanism is tighter the other one should be more permissive.

Table 2 shows the results for various values of I_{min} and I_{max} keeping fixed $I_{st} = 0$. In particular, we experimented with three different values of the range $I_{max} - I_{min}$ and, for each of them, with four values of I_{min} and I_{max}.

The results show that, for a large variety of values, the performances are comparable with the best ones for a fixed-size list. This shows that the variable-size tabu list actually gives only a little improvement, but the advantage of the use variable-size tabu list is that it makes easier to find a pair of values that give good performances, whereas it might be more difficult to find

$I_{max} - I_{min} = 10$				$I_{max} - I_{min} = 15$				$I_{max} - I_{min} = 20$			
$I_{min} I_{max}$	Best	Avg	Dev	$I_{min} I_{max}$	Best	Avg	Dev	$I_{min} I_{max}$	Best	Avg	Dev
$10 \div 20$	0	4.0	3.4	$10 \div 25$	0	3.7	2.4	$10 \div 30$	1	3.5	1.9
$15 \div 25$	0	3.8	3.4	$15 \div 30$	0	3.6	2.4	$15 \div 35$	0	3.6	2.6
$20 \div 30$	0	4.4	2.6	$20 \div 35$	0	4.6	2.3	$20 \div 40$	0	4.0	2.7
$25 \div 35$	1	3.4	3.0	$25 \div 40$	1	4.3	2.3	$25 \div 45$	0	3.4	2.3

Table 2: Results varying I_{min} and I_{max} [20 trials with TSmax = 2000, RNAmax = 0, Cycles = 0]

MAX = 1000, Trials = 39				MAX = 2000, Trials = 20				MAX = 4000, Trials = 15			
TSmax	Cycles	Avg	Dev	TSmax	Cycles	Avg	Dev	TSmax	Cycles	Avg	Dev
50	20	4.3	3.4	100	20	2.6	2.1	200	20	2.0	2.8
100	10	2.5	2.3	200	10	1.9	1.4	400	10	1.1	1.0
250	4	2.9	2.5	500	4	1.3	1.2	1000	4	1.2	1.2
500	2	2.5	1.7	1000	2	2.0	1.4	2000	2	1.0	1.2
1000	1	3.1	2.4	2000	1	2.6	1.7	4000	1	1.8	1.0

Table 3: Results varying TSmax and Cycles [20 trials with RNAmax = 100000, $I_{min} = 15$, $I_{max} = 30$, and $I_{st} = 0$]

the value for the fixed-length list.

Table 3 shows the results obtained with the complete algorithm for some combinations of the values of TSmax and Cycles (with fixed RNAmax = 100000). We fix the product of TSmax by Cycles, which we call MAX, to be equal to the three values 1000, 2000, and 4000. In this way, for each value of MAX the number of total TS moves performed before stopping is the same. The difference stands on how often the search is interrupted to restart from the best solution and to make the RNA phase.

The number of trials made for each configuration of the parameters is related to the corresponding running time. That is, we fixed approximately the total running time (2 hours) and we run all the experiments that fit in that time.

All experiments shown in the table were able to find a best solution of penalty 0, therefore we focus only on the average value (and the deviation). The results show that there exist a tradeoff between going in depth in one direction (large TSmax, small Cycles) and going in width in different directions (small TSmax, large Cycles). In particular, the best results are found for Cycles around the value 4.

The table shows that, while there is a quite large improvement going from MAX = 1000 to MAX = 2000, there is only a small one extending the search to MAX = 4000 iterations. Further experiments show that for bigger values of MAX the improvement decreases even more effectively.

The table also shows, compared with the results in Tables 1 and 2, that the use of RNA (with double moves) improves quite significantly the performances of the algorithm. Such improvement is considerably more significant for larger schools.

So far, we have discussed the importance of various parameters of the algorithm. However, one of the most important features of the algorithm is the use of the adaptive relaxation. Continuously changing the shape of the objective function, such feature causes TS to visit solution that have a different structure than the previously visited ones. Without relaxation, the best results show an average of 6.6 for the TS alone and an average of 3.8 for the complete algorithm

Method	Move type	# of Trials	Best	Average	Deviation
SD	atomic	3130	76	265.1	62.8
SD	double	25	24	54.0	18.9
RNA	atomic	2727	19	169.5	51.5
RNA	double	195	4	29.2	16.6

Table 4: Performance of other local search methods

(for 20 trials).

The adaptive relaxation also helps in finding a feasible solution. In fact, in some cases (in School 3) the algorithm without adaptive relaxation was not able to find a feasible solution. This is because the relaxation start taking place before a fully feasible solution is found, precisely when at least one of the three sources of infeasibilities is satisfied. When the relaxation of one of the three sources of infeasibilities starts, it helps in improving the objective function as well as in satisfying the hard constraints of the other two types.

Finally, in Table 4 we show the results obtained with the simpler local search methods: steepest descendent (SD) and randomized non-ascendent (RNA). Obviously, each solution trial has a different running time. As before, we compare different methods being equal the total running time (2 hours).

Such results clearly show that RNA with double moves outperforms the others. For example, it gives a better average than SD (with double moves) even though it costs less than an eighth of it in terms of running time.

6.2 School 2

The "S. Pertini" Technical School of Genzano (Italy) has 20 classes and 44 teachers (2 of which are part time ones). Classes take between 30 and 32 lectures weekly within 36 teaching periods in 6 days of 6 periods each. Exactly like School 1, all classes are required to be at school for all periods but the last of each day (i.e. $d_{ik} = 0$ for $k = 6, 12, 18, 24, 30, 36; i = 1, 2, 3$, and $d_{ik} = 1$ otherwise) and all of them are not available for the last period of Saturday (i.e. $c_{ik} = 0$ for $k = 36; i = 1, 2, 3$ and $c_{ik} = 1$ otherwise). Each teacher has a day-off, and there are a few other teacher's unavailabilities.

The neighborhood comprises 18288 legal moves, about 2120 of which are semi-illegal. The number of double moves is about 304600.

The school is split in three sites, and 10 teachers teach in more than one site. Therefore that commutations between two periods can take place and they must be taken into account by the objective function.

Gymnastic lectures, which are the most difficult ones to schedule, are given to two or three classes together and they are supplied in a single pair of joint lectures. In addition, there are some other simultaneous requirements due to four bilingual classes (which require the co-presence of English and French teachers).

A timetable of value 54 has been obtained in 64 minutes with the following setting of the parameters: $I_{min} = 20$, $I_{max} = 35$, $I_{st} = 1$, $\texttt{TSmax} = 1000$, $\texttt{Cycles} = 4$. After some manual adjustments and 36 more minute of running time, a timetable of value 28 has been produced. The manually produced timetable of the school has an objective value of 172.

6.3 School 3

The "Quinto O. Flacco" High School of Potenza (Italy) has 38 classes and 61 teachers (all full time ones). Classes are split in regular (35) and pilot ones (3). The former take between 27 and 29 lectures, whereas the latter take 32 lectures. Lectures are given within 36 teaching periods in 6 days of 6 periods each.

Regular classes are required to be at school for all periods but the last two of each day (i.e. $d_{ik} = 0$ for $k = 5, 6, 11, 12, 17, 18, 23, 24, 29, 30, 35, 36; i = 1, 2, 3$, and $d_{ik} = 1$ otherwise) and they are not available for the last period of every day (i.e. $c_{ik} = 0$ for $k = 6, 12, 18, 24, 36; i = 1, 2, 3$ and $c_{ik} = 1$ otherwise).

Pilot classes, like all classes in School 1 and 2, are required to be at school for all periods but the last of each day (i.e. $d_{ik} = 0$ for $k = 6, 12, 18, 24, 30, 36; i = 1, 2, 3$, and $d_{ik} = 1$ otherwise) and all of them are not available for the last period of Saturday (i.e. $c_{ik} = 0$ for $k = 36; i = 1, 2, 3$ and $c_{ik} = 1$ otherwise).

Almost all teachers have a day-off, and those who gave up the day-off are allowed to be away for 6 periods (corresponding to a day) freely chosen during the week.

The neighborhood comprises 25375 legal moves, about 1100 of which are semi-illegal. The number of double moves is about 388500.

Gymnastic lectures, which also in this case are the most difficult ones to schedule, are given to pairs of classes and they are supplied twice a week. Since there are 38 classes and only 31 available period for gymnastics (due to class unavailabilities), in 7 periods four classes must be together in the gym. In addition, there is one bilingual class.

A timetable of value 51 has been obtained in 272 minutes with the following setting of the parameters: $I_{min} = 20$, $I_{max} = 40$, $I_{st} = 2$, `TSmax` = 1500, `Cycles` = 4.

The timetable produced manually with the interactive support of a commercial package (which schedules one class at a time using some direct heuristic) of the school has an objective value of 279.

7 Conclusions

We have presented a TS-based algorithm for the high-school timetabling problem. The algorithm has given good results for schools of various types, and for different settings of the weights of the objective functions. For all cases, the timetable produced turned out to be better than the hand-made ones.

We found experimenting with different schools (of different types and sizes) absolutely necessary to ensure that the method is correct and general enough. In fact, it prevents the algorithm to be tuned only for the characteristic of the specific school in examination. In addition, it prevents the programmer from hard-coding some of the data of the school, instead of letting them to be part of the configuration supplied to the program. Unfortunately, the absence of a common definition of the problem and of widely-accepted benchmarks prevents us from comparing with other algorithms appeared in the literature.

The disadvantage of local search methods is that they does not allow the user to reason upon timetables only partially filled in. As a consequence, they do not permit to focus only on a group of lectures which are specifically critical to be scheduled. On the other hand, as we have already mentioned in Section 3.2, they offer a great advantage for interactive timetabling, which is generally necessary for finding a solution practically agreeable by the staff of the school.

In addition, in all practical cases our algorithm was able to find a feasible timetable in a reasonable amount of time. Conversely, constructive algorithms may not be able to create a complete timetable (see e.g. Yoshikawa et al., 1994). They generally are able to schedule 90-95% of the lectures, leaving the user with the problem to fit in the remaining 5-10% of the timetable, which can be extremely difficult.

Acknowledgements

I wish to thank Giansalvatore Mecca and Elisabetta Ranieri for their cooperation and for making available to me all the data of Schools 3 and 2, respectively. I also thank Amedeo Cesta, Giansalvatore Mecca, and Marco Schaerf for valuable comments on earlier drafts of this paper.

References

Abramson, D. (1991). Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science*, *37*(1), 98–113.

Carmusciano, F., & De Luca Cardillo, D. (1995). A simulated annealing with tabu list algorithm for the school timetable problem. In *Proc. of the 1st Intl. Conf. on the Practice and Theory of Automated Timetabling*, pp. 231–243.

Chahal, N., & de Werra, D. (1989). An interactive system for constructing timetables on a PC. *European Journal of Operational Research*, *40*, 32–37.

Colorni, A., Dorigo, M., & Maniezzo, V. (1992). A genetic algorithm to solve the timetable problem. Tech. rep. 90-060 revised, Politecnico di Milano, Italy.

Cooper, T. B., & Kingston, J. H. (1993). The solution of real instances of the timetabling problem. *The Computer Journal*, *36*(7), 645–653.

Costa, D. (1994). A tabu search algorithm for computing an operational timetable. *European Journal of Operational Research*, *76*, 98–110.

de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research*, *19*, 151–162.

Even, S., Itai, A., & Shamir, A. (1976). On the complexity of timetabling and multicommodity flow problems. *SIAM Journal of Computation*, *5*(4), 691–703.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability—A guide to NP-completeness*. W.H. Freeman and Company, San Francisco.

Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, *40*(10), 1276–1290.

Glover, F. (1989). Tabu search. Part I. *ORSA Journal of Computing*, *1*, 190–206.

Glover, F. (1990). Tabu search. Part II. *ORSA Journal of Computing*, *2*, 4–32.

Glover, F., & Laguna, M. (1993). Tabu search. In Reeves, C. R. (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*. Scientific Publications, Oxford.

Glover, F., Taillard, E., & de Werra, D. (1993). A user's guide to tabu search. *Annals of Operations Research*, *41*, 3–28.

Gotlieb, C. C. (1963). The construction of class-teacher timetables. In Popplewell, C. M. (Ed.), *IFIP congress 62*, pp. 73–77. North-Holland.

Junginger, W. (1986). Timetabling in Germany – a survey. *Interfaces*, *16*, 66–74.

Näher, S., & Uhrig, C. (1995). *The LEDA User Manual.* Version R3.2.

Neufeld, G. A., & Tartar, J. (1974). Graph coloring conditions for the existence of solutions to the timetable problem. *Communications of the ACM*, *17*(8), 450–453.

Ostermann, R., & de Werra, D. (1983). Some experiments with a timetabling system. *OR Spektrum*, *3*, 199–204.

Schaerf, A. (1995). A survey of automated timetabling. Tech. rep. CS-R9567, CWI, Amsterdam, NL. Available at `http://www.cwi.nl/ftp/CWIreports/AP`.

Schaerf, A., & Schaerf, M. (1995). Local search techniques for high school timetabling. In *Proc. of the 1st Intl. Conf. on the Practice and Theory of Automated Timetabling*, pp. 313–323.

Schmidt, G., & Strohlein, T. (1979). Timetable construction - an annotated bibliography. *The Computer Journal*, *23*(4), 307–316.

Schreuder, J. A. M., & Visscher, A. J. (1995). Timetabling in dutch secondary schools: the problem and the strategy to tackle it. In *Proc. of the 1st Intl. Conf. on the Practice and Theory of Automated Timetabling*, pp. 307–312.

Selman, B., Kautz, H. A., & Cohen, B. (1994). Noise strategies for improving local search. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, pp. 337–343.

Selman, B., Levesque, H., & Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proc. of the 10th Nat. Conf. on Artificial Intelligence (AAAI-92)*, pp. 440–446.

Tripathy, A. (1984). School timetabling – A case in large binary integer linear programming. *Management Science*, *30*(12), 1473–1489.

Tripathy, A. (1992). Computerised decision aid for timetabling - A case analysis. *Discrete Applied Mathematics*, *35*(3), 313–323.

van Laarhoven, P. J. M., & Aarts, E. H. L. (1987). *Simulated Annealing: Theory and Applications.* D. Reidel Publishing Company, Kluwer Academic Publishers Group.

Yoshikawa, M., Kaneko, K., Nomura, Y., & Watanabe, M. (1994). A constraint-based approach to high-school timetabling problems: a case study. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, pp. 1111–1116.