Local Search Techniques for Large High-School Timetabling Problems

Andrea Schaerf

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 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 *local search* techniques. The algorithm alternates different techniques and 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.**

Keywords— **Timetabling, Combinatorial Optimization, Scheduling, Local Search, Tabu Search**

I. 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 constraints are satisfied.

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

For the above reasons, considerable attention has been devoted to automated timetabling. During the last thirty years, starting in the early 60's with Gotlieb [1] and others, many papers related to automated timetabling have appeared in conferences (see [2] and [3]) and journals, and several applications have been developed and employed with reasonable success.

Most of the early techniques (see [4] and [5]) were based on a simulation of the human way of solving the problem. All such techniques, that we call *direct heuristics*, were based on *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 [6] and [7], *network flow* [8], and others. In addition, the problem has also been tackled by reducing it to a well-studied theoretical problem: *graph coloring* [9].

More recently, some approaches based on new search techniques appeared in the literature; among others, we have *simulated annealing* [10], *tabu search* [11], *genetic algorithms* [12], *constraint satisfaction* [13], and combination of different methods [14].

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 [15]). The problem we have to resolve comes from the Italian high-school system, and it is a modification of the theoretical problem defined by Gotlieb [1] and others. In some countries, e.g., Germany (see [5]), the high school is organized in a similar way, and so the solution proposed in this paper applies there as well. In some others, e.g., Holland (see [16]), students have more freedom in the selection of the 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 constituted by the alternating of tabu search and randomized hill-climbing. The two phases make use of different types of moves and are guided by different cost functions. In particular, the tabu search phase makes use of an adaptive relaxation of the hard constraints.

Our algorithm is suitable for both interactive and batch run. That is, it allows the user for manual modifications of both 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., [17]).

The paper is organized as follows: Section II defines the high-school timetabling problem. Section III introduces local search techniques in general, and the tabu search in particular. Section IV describes the way we represent the search space and the neighborhood relation. Section V explains in details the algorithm employed. Section VI shows the experimental results and the performances obtained. Section VII deals with related work. Finally, in Section VIII we draw some conclusions.

II. HIGH-SCHOOL TIMETABLING PROBLEM

The problem we are dealing with is an optimization problem, and it is therefore defined through a solution space and an objective function. In Section II-A we define the solution space by introducing the underlying search problem, in Section II-B we add our objective function to it.

The constraints that define the solution space, which must be satisfied by a solution, are called *hard constraints*,

whereas those that contribute to the objective function, and can be violated at a given penalty, are called *soft constraints*.

A. Underlying Search Problem

There are m classes $c_1, \ldots, c_m$, n teachers $t_1, \ldots, t_n$, and p periods $1, \ldots, p$. A non-negative integer matrix $R_{m \times n}$, called *Requirements matrix*, is given 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 [18] and [5])

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

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

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

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

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

The TTP has been shown NP-complete by Even *et al* [19] and it appears also in Garey and Johnson's book [20, 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 hard constraints.

Firstly each class, for a given set of periods, must be involved in one lecture. This constraint can be expressed as follows

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

where $D_{m \times p}$ is a binary matrix such that $d_{ik} = 1$ if class c_i must 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, such constraints can force 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. For a given class c_i, the cardinality of such set, i.e. $\sum_{k=1}^{p} d_{ik}$, depends on (and is obviously less than or equal) the total requirements of class c_i, i.e. $\sum_{j=1}^{n} r_{ij}$.

Secondly, some pairs of lectures must be scheduled simultaneously. Specifically, there exists a set of quadruples of the form $\langle i_1, j_1, i_2, j_2 \rangle$ (with $i_1 \neq i_2$ and $j_1 \neq j_2$) 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 of quadruples, the above constraints can be expressed as follows

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

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

Constraints 6 are also necessary for tackling practical cases. In fact, they turned out to be quite a general mechanism with 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 by introducing dummy classes.

B. Objective Function

The computation of the objective function presupposes the introduction of a number of items for teachers, classes, and teacher/class pairs. In particular, for each teacher we introduce: minimum and maximum number of teaching periods per day, undesired teaching periods, and seniority. For each class we introduce the site in which its class room is located. For each pair teacher/class we introduce: 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 to minimize is a weighted sum of the following soft constraints, 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 not be significant for some of the specific schools, in that case their weight is set to 0).

Holes (or Windows) [1]: Periods in which the teacher is not assigned to any activity (teaching or otherwise) between two assignments in the same day.

Splits [6]: Two lectures to a class on the same day 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. This feature ensures that lectures of the same teacher are sufficiently spread over the week.

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. This constraint is the soft version of the unavailability constraint for teachers described in Section II-A (Constraints 2).

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.

III. Local Search and Tabu Search

Local search techniques are a family of general-purpose techniques for the solution of optimization problems [21].

Consider an optimization problem, and let S be a possible search space for it. 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* part of the search space, stepping iteratively from one solution to one of its neighbors. The modification that transforms a solution into one of its neighbors is called a *move*.

The selection of the move to be performed at each step of the search is based on the *cost function* f, which assesses the quality of the solution. The cost function generally is composed of the so-called *distance to feasibility*, which accounts for the number of constraints that are violated, and the objective function of the problem.

Among the local search techniques, we have the *steepest descent method* (SD): It analyses all possible moves and chooses the one that gives the best improvement. It accepts the candidate move only if it improves the value of the cost function, and it stops as soon as it reaches a local minimum.

The above method requires the exploration of the whole neighborhood. The *randomized descent method* (RD) instead analyses a random neighbor and accepts it if it is better than 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 cost function.

The randomized descent method is also trapped in the very first local minimum it gets. The *randomized non-ascendent method* (RNA) instead accepts the random neighbor only if it is better *or equal* to the current. Its stop criterion is also based on a fixed number of iterations without improving the value of the cost function.

Allowing also for *sideways moves*, RNA has the feature of being able to follow descending paths that pass through *plateaux* (see [22]). That is, if the search lands in a plateau, it is able to move inside the plateau itself, and might get down from it through a solution different from the one from which it reached the plateau.

The *steepest non-ascendent method* (SNA) combines the search for the steepest move with the use of *sideways moves*. This method, and some variants of it, have been used, for example, for the SAT problem with the name GSAT (see [22] and [23]).

A. Tabu Search

We now introduce the tabu search (TS) technique. Numerous variants of TS have been proposed in the litera-

ture. The one we describe here is what we consider the basic version of TS. We refer to [24] for a comprehensive presentation of its many variants, whereas some modification of its basic features will have already been discussed in Section V.

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 cost 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 that are forbidden. 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 cost 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 cost 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 cost 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 cost function in the current solution reaches a given lower bound.

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

The effectiveness of tabu search for scheduling problems has been demonstrated in several papers and applications. For example, Vaessens *et al* [25] show that tabu search is clear superior than many other local search procedures for the *job-shop scheduling* problem.

We refer to [26] and [27] for an example of the application of tabu search to a problem similar to the high-school timetabling problem, namely the *university timetabling* problem.

B. Local 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 has been generated, it can be used as the starting point for a new search after some constraints have been manually modified.

As an example, suppose that after finding a good timetable the requirement matrix is slightly altered by increasing the number of required lectures of a given teacher to a given class. Using our method we can add the miss-

ing lectures manually to the timetable (at any period) and restart the search. Such a 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 preferences, it is still possible to modify the instance, starting from the current timetable and improving it using our algorithm.

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

IV. REPRESENTATION OF THE PROBLEM

A timetable is represented as an integer-valued matrix $M_{n \times p}$ 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 available 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 IV-B), which permit effective navigation of the search space. In addition, this is generally the representation that the persons that do manual timetabling reason upon, and therefore, it is also suitable for interactive timetabling with manual corrections.

This representation has been used also by Colorni *et al* [12]. Conversely, Carmusciano and De Luca [28] use a dual representation with a matrix $N_{m \times p}$ 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, this makes it more difficult to compute the various features of the objective function, which are more based on the scheduling of the single teachers than 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 available for possible supply teaching. The figure also shows, by the symbol "--", the unavailabilities of teachers, which are not part of the solution.

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 that take French as first foreign language; therefore, when teacher 13 teaches class "2E" (Tuesday second period, in Figure 1), teacher 14 must teach the dummy class "2e". Conversely, "Second Foreign Language" is taught to class 2E by the French teacher

```
----------------------------------------------------
tchr|     Monday       |      Tuesday         |
----------------------------------------------------
3   |            <> 2A | -- -- -- -- -- --  | ..
4   | 1A 4A 4A <>    -- | <> 1A            -- | ..
5   | 2B 3B           |    2B <> <> 3B 1B  | ..
6   | <> 1B ** 4B 5B  | -- -- -- -- -- --  | ..
7   |    <> 3C 3C 2C  | 3C <> 2C 2C 2C     | ..
8   |       5C 1C 1C  | 4C 4C <> 5C        | ..
9   |          2D 2D  | 1D 1D              | ..
10  | 2E 2E 3D 3D 5D  | 5D 5D 2E 3D        | ..
11  |    5A 3A 2A 1A 4A | 1A <> 2A          | ..
12  |    2C 3C 2B 1B 3B | 1C 3C 3B 1B 2B    | ..
13  |          1D 3D  | 3D 2E 2d 2e        | ..
14  |          3A 4a  |    2e <> 2A 1A 3A  | ..
15  |          1B 3B 5B | 2B 5B 4B          | ..
16  | -- -- -- -- -- -- | 2C 1C 3C          | ..
17  | -- -- -- -- -- -- | --    2D 2E 5D     | ..
----------------------------------------------------
```

Fig. 1. A fragment of a timetable

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

A. Including Infeasibilities in the Cost Function

Infeasible timetables are also included in the search space of the algorithms. The cost function is therefore composed of the objective function described in Section II-B augmented by also including the number of infeasibilities of the solution.

To this aim, we split the hard constraints introduced in Section II-A into three new types. In particular, we count:
(I) the number of times that either two teachers teach the same class in one period or a class is uncovered (corresponding to Constraints 3 when $t_{jk} = 1$).
(II) The number of times a simultaneous assignment is missed (corresponding to Constraints 6).
(III) the number of times a teacher (a class) teaches (is taught) when he/she (it) is not available (corresponding to Constraints 2 and 3 when $c_{ik} = 0$ and $t_{jk} = 0$).

The possibility that a teacher teaches simultaneously 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 V, in order to ensure a better navigation of the search space, such weight is allowed to vary during the search phase.

B. 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 in a given row. That is, the lectures of a teacher t in two different periods 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 such a move an *atomic move*, and we identify it by the triple $\langle t, p_1, p_2 \rangle$. In order to have a unique representation of a move, we assume that $p_1 > p_2$. With this assumption, we also understand that a move is the unique reversal of itself.

We also consider a more complex move type. 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 (obviously new infeasibilities can be created). If the first atomic move creates no infeasibilities, then there is no second move and the double move 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, any 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 $R_{m \times n}$.

Costa [11] 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, with both assignments in the same period at the intermediate step.

We don't follow such an approach because we consider a quite different set of hard and soft constraints with respect to Costa [11] and in our case it would be difficult to define an effective cost function (which is based on single teacher assignments) in the presence of multiple assignments for a single period.

V. APPLICATION OF LOCAL SEARCH

Our algorithm is primarily a TS with the neighborhood constituted by atomic moves. However, the TS is alternated with a phase of RNA using double moves.

In detail, 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 until 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 generates 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

```
Algorithm High School TimeTabling
begin
    GetInitialSolution;      (* random or from input file *)
    InitializeSearchVariables;
    IdleCycles := 0;
    while IdleCycles < Cycles
    do begin
        NoImprovements := 0;
        while (NoImprovements < RNAmax)
        do begin     (* RNA phase *)
            NoImprovements := NoImprovements + 1;
            DoubleMove := RandomDoubleMove;
            if Delta(DoubleMove) ≤ 0
                (* Delta is the variation of the objective function *)
            then begin
                UpdateCurrentTimeTable(DoubleMove);
                if Delta(DoubleMove) < 0
                then begin
                    NoImprovements := 0
                    IdleCycles := 0
                end
            end
        end;
        NoImprovements := 0;
        BestTimeTable := CurrentTimeTable
        while (NoImprovements < TSmax)
        do begin     (* TS phase *)
            NoImprovements := NoImprovements + 1;
            BestMove := AtomicRandomMove;
            BestDelta := Delta(BestMove,DynamicWeights);
            forall Move in AtomicNeighborhood(CurrentTimeTable)
            begin if Delta(Move,DynamicWeights) < BestDelta
                and (not Tabu(Move) or Aspiration(Move))
                and (Delta(Move,DynamicWeights) < 0
                or not LowInfluence(Move))
                (* low-influence moves are explained in Section V-D *)
            then begin
                BestMove := Move;
                BestDelta := Delta(Move,DynamicWeights)
            end
        end;
        UpdateCurrentTimeTable(BestMove);
        UpdateTabuList(BestMove);
        if IsBest(CurrentTimeTable)
        then begin
            BestTimeTable := CurrentTimeTable;
            NoImprovements := 0
            IdleCycles := 0
        end
        if UpdateIteration then UpdateDynamicWeights
    end;
    CurrentTimeTable := BestTimeTable
    end
end
```

Fig. 2. Timetabling Algorithm

reason for it is twofold: On the one hand, the RNA, using double moves, might find improvements that the TS was 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), makes substantial sideways modifications. Therefore, it "shakes up" the solution before the TS starts again to try to improve it. The idea underlying such a procedure is that after the TS has worked unsuccessfully for a given number of iterations, it is useful to try to modify the solution so that the TS can start in a different direction.

The complete algorithm is shown in Figure 2. In the

following subsections, we discuss a number of features of it.

A. 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 cost function. Conversely, during the TS stage, such weight is dynamically adjusted in the following way (as proposed in [29]): For each of the three sources of infeasibilities (see Section IV-A), namely (1) clash of teachers or uncovered classes, (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 (each time) in the interval $[1.8, 2.2]$, whereas in [29] γ is deterministically set to 2. This randomization avoids deterministic ratios between different components of the cost function that could bias the search.

Differently from [29], we bound the value of each α_i by two constants $\alpha_{i,min}$ and $\alpha_{i,max}$. Thus, 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 from getting too high (resp. too low) after a long sequence of infeasible (resp. feasible) solutions. In that case, when such a sequence is interrupted, too many iterations would be wasted before the value of α_i gets back to a value that is comparable to the values of the other components of the cost function.

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.

From this point on, to refer to the original cost function, i.e. the one obtained setting $\alpha_i = 1$ for $i = 1, 2, 3$, we use the name *non-relaxed* cost function.

B. Tabu List Management

We employ a tabu list of variable size (as initially proposed by [30]). Following the approach taken by Gendreau *et al* [29], 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 (similarly to [11]). 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 of the tabu list, where I_{st} is another user-specified parameter such that $I_{st} < I_{min}$. 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.

C. 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 cost function.

We use the simplest standard aspiration function, which accepts a tabu move only if it improves the current best 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 cost function and not on the current one, they did not give any improvement to the method.

D. Neighborhood Exploring

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

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

which is the number of teachers multiplied by 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 our three 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.

An empirical analysis of the neighborhood composition reveals that there are moves that have "small influence" on the structure of the solution. That is, certain moves can be applied in many different solutions, but their execution does not change the quality of most of the other possible moves. For example, moves of this kind are the exchanges of a idle period with the assignment of being available for possible supply teaching ("<>").

We call such moves *low-influence moves*, and we define them formally as the moves that exchange two values in which neither represents an actual class (e.g., they are idle, assisting, or at disposal) and those that move classes between two periods in which both classes are not requested to be at school.

This analysis also shows that in many situations a large number of low-influence moves giving a zero-cost variation (i.e., sideways low-influence moves) are available. Due to the TS selection strategy, such sideways low-influence moves are preferred to worsening moves that might instead alter the structure of the solution, and lead eventually to a more promising part of the search space. This fact prevents the TS from exploring effectively the search space, unless we use a very long tabu list.

On the other hand, low-influence moves might sometime produce an improvement in the cost function (e.g., by filling holes), therefore we do not want to forbid them.

In order to balance these two conflicting needs we adopt the following strategy: Low-influence moves are accepted only if they *strictly* improve the cost function (see the algorithm of Figure 2).

This strategy gives better results than both considering low-influence moves as normal ones and forbidding them completely.

In our three examples, the number of low-influence moves is about 10-15% of the total legal ones.

VI. EXPERIMENTAL RESULTS

In this section we present our experimental results. All the code (4000 lines) has been implemented by the author in standard C++ and it runs on a Silicon Graphics workstation INDY. All data structures are implemented using the public C++ library LEDA [31].

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

The first school is a small dummy one, specifically constructed for use 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 this 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.

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

A. 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 none of them are 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 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 low-influence moves also depends on the specific solution and it is about 500.

Tables I and II highlight the importance of the parameters related to 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 (which is not reused for different settings). For all settings, a feasible solution has always been found by all trials. The tables show, for each set of 20 trials, the best value of the objective function, the average and the standard deviation.

Table I 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 looser.

Table II 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 using it is that it makes easier to find a pair of values that give good performances, whereas it might be more difficult to find the best value for the fixed-length list.

Table III 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 total number of TS moves performed before stopping is the same. The difference is in how often the search is interrupted to restart from the best solution and the RNA phase is invoked.

In this set of experiments the running time for different setting can vary substantially. For this reason, we fix the number of trials made for each configuration of the parameters is related to the corresponding running time. That is, we fixed the total running time (2 hours) and we ran as many experiments as possible within that timeframe.

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 exists a tradeoff between doing few long tabu search explorations (large `TSmax`, small `Cycles`) and performing many shake-ups (small `TSmax`, large `Cycles`). In particular, the best results are found for `Cycles` around the value 4.

The table shows that, while there is quite a large improvement going from `MAX = 1000` to `MAX = 2000`, there is only a small one extending the search to `MAX = 4000` iter-

TABLE I

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

	$I_{st} = 0$			$I_{st} = 1$			$I_{st} = 2$		
I	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 II

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

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

ations. Further experiments show that for bigger values of MAX the marginal improvement decreases even more.

The table also shows, compared with the results in Tables I and II, that the use of RNA (with double moves) improves quite significantly the performance of the algorithm.

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 cost function causes TS to visit solutions 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 (for 20 trials).

Finally, in Table IV we show the results obtained with the simpler local search methods, namely steepest descent (SD) and randomized non-ascendent (RNA). These methods are tested in isolation, with no interleaving with other methods. Obviously, each solution trial has a different running time. As before, we compare different methods by using the same total running time (2 hours) for each. These results clearly show that RNA with double moves outperforms SD. For example, RNA with double moves gives a better average than SD (with double moves) even though it requires less than an eighth of the running time. They also show that the use of TS, which is similar in spirit to SD, working on the neighborhood composed of double moves would be hopelessly inefficient, and thus ineffective too.

B. School 2

This technical school 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 $k = 6, 12, 18, 24, 30, 36; i = 1, \ldots, 20$, and $d_{ik} = 1$ otherwise) and all of them are not available for the last period of Saturday (i.e. $c_{ik} = 0$ for $k = 36; i = 1, \ldots, 20$ and $c_{ik} = 1$ otherwise). Each teacher has a day-off, and there are a few other teacher unavailabilities.

The neighborhood of each solution comprises 18288 legal moves, about 2120 of which are low-influence. The number of double moves is about 304600.

The school is split into three sites, and 10 teachers teach in more than one site. Therefore movements of teachers in the time between two consecutive periods can take place and 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 feasible timetable of value 54 has been obtained in 64 minutes (not including parameter setting) with the following values of the parameters: $I_{min} = 20$, $I_{max} = 35$, $I_{st} = 1$, TSmax = 1000, Cycles = 4. After some manual adjustments and 36 more minutes of running time, a timetable of value 28 has been produced. The manually produced timetable of the school has an objective function

TABLE III

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

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 IV

Performance of other local search methods

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

value of 172.

C. 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 $k = 5, 6, 11, 12, 17, 18, 23, 24, 29, 30, 35, 36; i = 1, \ldots, 38$, and $d_{ik} = 1$ otherwise) and they are not available for the last period of every day (i.e. $c_{ik} = 0$ for $k = 6, 12, 18, 24, 36; i = 1, \ldots, 38$ 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 of each solution comprises 25375 legal moves, about 1100 of which are low-influence. 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. In addition, there is one bilingual class.

A feasible 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 by the school with the interactive support of a commercial package (which schedules one class at a time using some direct heuristics) has an objective value of 279.

The experiments with School 3 show that the adaptive relaxation helps also in finding a feasible solution. That is, for this school the algorithm without adaptive relaxation was not able to find a feasible solution. Relaxation can help because it starts taking place before a fully feasible solution is found, precisely when at least one of the three sources of infeasibility is satisfied. When the relaxation of one of the three sources of infeasibility starts, it helps in improving the objective function as well as in satisfying the hard constraints of the other two types.

The costs shown in this section do not consider the parameter tuning phase, which can indeed be quite expensive. However, an approximate scheme based on school size can be generated once and for all, for all the parameters. All runs performed using such a scheme are expected to provide reasonably good solutions. For better ones, the user is allowed to intervene directly in the parameter setting.

VII. RELATED WORK

The literature on timetabling problems comprises hundreds of conference and journal papers.[1] The majority of the recent ones, however, deal with the problem of timetabling university courses and exams, rather than high-school classes. The university problem has quite different features and peculiarities, although some techniques have been applied to both the high-school and university problems. A discussion on university timetabling problems is out of the scope of this paper. We refer to [15] for a

---

[1]There is currently an active community working on various kinds of timetabling problems, with a series of conferences and a working group on the field. It is reachable at http://www.asap.cs.nott.ac.uk/ASAP/watt.

survey on such problem.

We therefore focus on the papers that address specifically the high-school timetabling problem. In particular, we now discuss those that are more tightly related to our work.

- Colorni *et al* [12] deal with a problem which is also taken from an Italian school, and therefore is very similar to ours. However, there are some differences, for example, in their paper there is no mention of simultaneous lectures, which are crucial for the performances of our solution algorithms.

  That paper is mostly devoted to the application of genetic algorithms, but it also describes the implementation of a simple tabu search procedure. The tabu search actually turned out to give better performances than the genetic algorithm proposed in the paper.

  In the tabu search algorithm, they make also use of a non-adaptive form of relaxation of the hard constraints; that is, the (unique) weight of hard constraints is high in the beginning of the search, then it is set to a low value for a fixed number of iterations, and it is raised again for the final part of the algorithm.

- Costa [11] applies tabu search to a Canadian school timetabling problem. His problem also includes two-period lectures and special rooms (e.g., laboratories), whereas he does not consider simultaneous lectures. As already mentioned in section IV-B, Costa makes use of a different representation of the problem, which allows also for more than one lecture for a teacher in the same period. He also uses a different type of move, which fits his representation, and which is generally employed for the university timetabling problem: A move changes the period of a single lecture for a teacher. He makes no use of double moves and interleaving of local search procedures, and he does not relax constraints adaptively.

- Yoshikawa *et al* [13] introduce a quite expressive Lisp-like constraint language, which allows the user to express different kind of hard and soft constraints. Their solution method combines a sophisticated greedy algorithm, called *Really-Fully-Lookahead*, for finding the initial solution and a local search procedure for the optimization phase. The local search algorithm employed is a variation of the *steepest descent* method, called *Min-Conflict Hill-Climbing*, defined by Minton *et al* [32].

- Alvarez-Valdes *et al* [33] deal with a Spanish school timetabling problem, which is quite similar to the Italian case. They use a three-phase solution algorithm based on tabu search. The first phase is a construction based on the solution of an *independent-set* problem. The second phase is a tabu search, which looks for a feasible timetable. The move definition is rather complex, and consists in changing the period of one lecture (similarly to [11]), plus rescheduling all lectures that conflict with the new position of the lecture. They do not use adaptive relaxation. The last phase improves the solution using various heuristics.

Although a large body of research has been done in the field and a few successful applications have been implemented, the problem is still far from being solved in a general and satisfactory way. Therefore, research work is still needed in order to provide a definitive answer to the high-school timetabling problem.

VIII. Conclusions

We have presented a algorithm for the high-school timetabling problem based on local search. 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 sufficiently general. In fact, it prevents the algorithm from being tuned only to the characteristics of the specific school under examination. In addition, it prevents the programmer from hard-coding some of the data of the school, instead letting them 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 the papers discussed in Section VII and with other algorithms appearing in the literature.

The disadvantage of local search methods is that they do not allow the user to reason upon timetables only partially filled in. As a consequence, they do not permit one 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 III-B, they offer a great advantage for interactive timetabling, which is generally necessary for finding a solution agreeable to the staff of the school. An attempt to combine constructive and local search in order to achieve the advandages of both paradigms is provided in [34].

In addition, in all practical cases our algorithm was able to find a feasible timetable in a reasonable amount of time. Conversely, preliminary experiments with constructive algorithms show that they may not be able to create a complete timetable. 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.

References

[1] C. C. Gotlieb, "The construction of class-teacher timetables", in *IFIP congress 62*, C. M. Popplewell, Ed. 1963, pp. 73–77, North-Holland.

[2] E. Burke and B. Paechter, Eds., *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[3] E. Burke and M. Carter, Eds., *Proc. of the 2nd Int. Conf. on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[4] G. Schmidt and T. Strohlein, "Timetable construction - an annotated bibliography", *The Computer Journal*, vol. 23, no. 4, pp. 307–316, 1979.

[5] W. Junginger, "Timetabling in Germany – a survey", *Interfaces*, vol. 16, pp. 66–74, 1986.

[6] A. Tripathy, "School timetabling – A case in large binary integer linear programming", *Management Science*, vol. 30, no. 12, pp. 1473–1489, 1984.

[7] A. Tripathy, "Computerised decision aid for timetabling - A case analysis", *Discrete Applied Mathematics*, vol. 35, no. 3, pp. 313–323, 1992.

[8] R. Ostermann and D. de Werra, "Some experiments with a timetabling system", *OR Spektrum*, vol. 3, pp. 199–204, 1983.

[9] G. A. Neufeld and J. Tartar, "Graph coloring conditions for the existence of solutions to the timetable problem", *Communications of the ACM*, vol. 17, no. 8, pp. 450–453, 1974.

[10] D. Abramson, "Constructing school timetables using simulated annealing: sequential and parallel algorithms", *Management Science*, vol. 37, no. 1, pp. 98–113, 1991.

[11] D. Costa, "A tabu search algorithm for computing an operational timetable", *European Journal of Operational Research*, vol. 76, pp. 98–110, 1994.

[12] A. Colorni, M. Dorigo, and V. Maniezzo, "Metaheuristics for high school timetabling", *Computational Optimization and Applications*, vol. 9, no. 3, pp. 275–298, 1998.

[13] Masazumi Yoshikawa, Kazuya Kaneko, Toru Yamanouchi, and Masanobu Watanabe, "A constraint-based high school scheduling system", *IEEE Expert*, vol. 11, no. 1, pp. 63–72, 1996.

[14] T. B. Cooper and J. H. Kingston, "The solution of real instances of the timetabling problem", *The Computer Journal*, vol. 36, no. 7, pp. 645–653, 1993.

[15] Andrea Schaerf, "A survey of automated timetabling", Tech. Rep. CS-R9567, CWI, Amsterdam, The Netherlands, 1995, To appear in *Artificial Intelligence Review*.

[16] Jan A. M. Schreuder and Adrie J. Visscher, "Timetabling in dutch secondary schools: the problem and the strategy to tackle it", in *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling*, 1995, pp. 307–312.

[17] N. Chahal and D. de Werra, "An interactive system for constructing timetables on a PC", *European Journal of Operational Research*, vol. 40, pp. 32–37, 1989.

[18] D. de Werra, "An introduction to timetabling", *European Journal of Operational Research*, vol. 19, pp. 151–162, 1985.

[19] S. Even, A. Itai, and A. Shamir, "On the complexity of timetabling and multicommodity flow problems", *SIAM Journal of Computation*, vol. 5, no. 4, pp. 691–703, 1976.

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

[21] Emile Aarts and Jan Karel Lenstra, *Local Search in Combinatorial Optimization*, John Wiley & Son, Chichester, 1997.

[22] Bart Selman, Hector Levesque, and David Mitchell, "A new method for solving hard satisfiability problems", in *Proc. of the 10th Nat. Conf. on Artificial Intelligence (AAAI-92)*, 1992, pp. 440–446.

[23] Bart Selman, Henry A. Kautz, and Bram Cohen, "Noise strategies for improving local search", in *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, 1994, pp. 337–343.

[24] Fred Glover and Manuel Laguna, *Tabu search*, Kluwer Academic Publishers, 1997.

[25] Rob Vaessens, Emile Aarts, and Jan Karel Lenstra, "Job shop scheduling by local search", *INFORMS Journal of Computing*, vol. 8, no. 3, pp. 302–317, 1996.

[26] A. Hertz, "Tabu search for large scale timetabling problems", *European Journal of Operational Research*, vol. 54, pp. 39–47, 1991.

[27] A. Hertz, "Finding a feasible course schedule using tabu search", *Discrete Applied Mathematics*, vol. 35, no. 3, pp. 255–270, 1992.

[28] F. Carmusciano and D. De Luca Cardillo, "A simulated annealing with tabu list algorithm for the school timetable problem", in *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling*, 1995, pp. 231–243.

[29] M. Gendreau, A. Hertz, and G. Laporte, "A tabu search heuristic for the vehicle routing problem", *Management Science*, vol. 40, no. 10, pp. 1276–1290, 1994.

[30] E. Taillard, "Robust taboo search for the quadratic assignment problem", *Parallel Computing*, vol. 17, pp. 433–445, 1991.

[31] Kurt Mehlhorn, Stefan Näher, Michael Seel, and Christian Uhrig, *The LEDA User Manual*, 1998, Version 3.6.

[32] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems", *Artificial Intelligence*, vol. 58, pp. 161–205, 1992.

[33] R. Alvarez-Valdes, G. Martin, and J.M. Tamarit, "Constructing good solutions for the Spanish school timetabling problem", *Journal of the Operational Research Society*, vol. 47, 1996.

[34] Andrea Schaerf, "Combining local search and look-ahead for scheduling and constraint satisfaction problems", in *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, 1997, pp. 1254–1259, Morgan-Kaufmann.

Andrea Schaerf received his *Laurea* degree in Electrical Engineering and his Ph.D. in Computer Science from University of Rome "La Sapienza" in 1990 and in 1994, respectively. In 1995-96 he has been ERCIM fellow at CWI in Amsterdam. From 1996 to 1998, he has been Assistant Professor at University of Rome "La Sapienza". In 1998 he joined University of Udine as Associate Professor of Computer Science. His main research interests are in the solution of scheduling problems using local search and constraint programming, and in the design of constraint programming languages and spefication tools for combinatorial problems.