# Tabu Search Techniques for Large High-School Timetabling Problems

**Andrea Schaerf**[*]

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198, Rome, Italy
e-mail: aschaerf@dis.uniroma1.it

## Abstract

The high-school timetabling problem consists in assigning all the lectures of a high school to the time 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.

## 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 these reasons, a considerable attention has been devoted to automated timetabling. During the last thirty years, many papers related to automated timetabling have appeared in conferences and journals, and several applications have been developed and employed.

Most of the early techniques (Schmidt & Strohlein 1979; Junginger 1986) were based on a simulation of the human way of solving the problem. All such heuristic techniques 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.

*This work has been carried out while the author was visiting CWI in Amsterdam. It is part of the ERCIM fellowship Programme and financed by the Commission of the European Communities.

Later on, researchers started to apply general techniques to this problem. Then, we see algorithms based on *integer programming* (Tripathy 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 *et al.* 1996), 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 (Schaerf 1995). The problem we tackle comes from the Italian high-school system. In some countries, e.g. Germany (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, students 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 propose is based on *local search* (or *neighborhood search*) and it is an adaptation of tabu search, with some modifications. Our algorithm is suitable for both interactive and batch run. That is, it can generate a timetable directly from scratch, but it also allows the user for manual modifications of the timetable and the constraints during the search phase.

For the sake of brevity, some aspects of the algorithm are omitted. They can be found in (Schaerf 1996).

## 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.

### Solution Space

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 (de Werra 1985; Junginger 1986).

$$find \quad x_{ijk} \quad (i = 1..m; j = 1..n; k = 1..p)$$

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

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

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

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

This problem has been shown NP-complete in (Even, Itai, & Shamir 1976). In order to deal with real instances of the problem, we tackle a variant of it. 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

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

where $D_{m \times p}$ is a binary matrix such that $d_{ik} = 1$ if class $c_i$ must necessarily be taught at period $k$, 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 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 $\langle i, j, i', j' \rangle$ such that all lectures of teacher $t_j$ to class $c_i$ must be simultaneous to lectures of teacher $t_{j'}$ to class $c_{i'}$. For each quadruple $\langle i, j, i', j' \rangle$, the above constraint can be expressed as follows

$$x_{ijk} x_{i'j'k} + \overline{x}_{ijk} \, \overline{x}_{i'j'k} = 1 \quad (k = 1..p) \qquad (6)$$

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

Constraints 6 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 a single class), and others. All such features, although they may be formulated differently, can always be reduced to constraint of the above form introducing dummy classes.

## 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.

**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 for teaching (e.g. for administrative work or professional development).

## 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 *randomized non-ascendent method* (RNA): It analyzes a random neighbor and accepts it if it is better *or equal* to the current one. It stops after a fixed number of iterations without improving the value of the objective function. Allowing for *sideways moves*, this method has the feature of being able to follow descending paths that pass through *plateaux*.

We now briefly describe the basic principles of tabu search (TS). See for example (Glover, Taillard, & de Werra 1993) for a comprehensive presentation.

Starting from the initial solution $s_0$, the TS algorithm iteratively explores a subset of the neighborhood $N(s)$ of the current solution $s$; the neighbor 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 usually 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 $v$.the given value. 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 of neighbor solutions tested at each iteration, and `TSmax`, the maximum number of iterations without improving the objective function.

Some other local search methods (including *Simulated Annealing*) have been considered in a preliminary work (Schaerf & Schaerf 1995), which shows a quite clear dominance of tabu search over simulated annealing for high-school timetabling.

## 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, which permit to navigate effectively the search space. Moreover, 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, Dorigo, & Maniezzo 1992).

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

### Embedding Infeasibilities

Infeasible timetables are also included in the search space of the algorithms. The objective function 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

```
----------------------------------------------
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    | ..
----------------------------------------------
```

Figure 1: A fragment of a timetable

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

## Move Types 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 moves 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 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, 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.

## Application of the Tabu Search

Our algorithm is basically a TS with the neighborhood defined 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, & de Werra 1993).

In details, the initial solution is obtained by scheduling the lectures for each teacher randomly, respecting the requirement matrix (it can be obtained from previous runs, in case of interactive timetabling). Then, the RNA starts to work on the initial 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 integer-valued quantities `RNAmax`, `TSmax`, and `Cycles` are parameters of the algorithm.

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.

## 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 we multiply $W$ by a real-valued factor $\alpha_i$ (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 $\alpha_i := \alpha_i \cdot \gamma$;
   - otherwise $\alpha_i$ is left unchanged.

The real-valued quantity $\gamma$ is randomly selected (at each time) in the interval $[1.8, 2.2]$. In (Gendreau, Hertz, & Laporte 1994), $\gamma$ 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.

## 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. The first one 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 practice, 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.

## Aspiration Function

Due to the use of the adaptive relaxation, the value reached from a given solution depends on the current values of the $\alpha$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 $f(s^*) - 1$, where $s^*$ is the current optimum, for all solutions $s$.

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

## Neighborhood Exploring

The size of the neighborhood of any solution $s$, is given by $\mid N(s) \mid = mp(p-1)/2$, 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. Actually, not all moves are considered in the same way. Some of them, called ***semi-illegal***, moves are accepted only if they ***strictly*** improve the (current) objective function. The details about semi-illegal moves can be found in (Schaerf 1996).

## Experimental Results

All the code has been implemented in standard C++ and it runs on a Silicon Graphics workstation INDY. We experimented with three schools, which differ from each other for number of lectures and 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. On such school, we experimented with a large number of combinations of values of different parameters so as to understand how they interplay.

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.

For the sake of brevity, we do not present here the configuration of the schools and all the results of our experiments. They can be found in (Schaerf 1996).

We just mention that for the second school (20 classes, 44 teachers, 632 lectures to schedule) a timetable of value 54 has been obtained in 64 minutes with the following setting of the parameters: $I_{min} = 20$, $I_{max} = 25$, $I_{st} = 1$, `TSmax` = 1000, `Cycles` = 4. After some manual adjustments and 36 more minute of running time, a timetable of value 28 has been produced. The manually-produced running timetable of the school has an objective value of 172.

For the third school (38 classes, 61 teachers, and 1098 lectures to schedule) a timetable of value 51 has been obtained in 272 minutes with the following setting of the parameters: $I_{min} = 25$, $I_{max} = 35$, $I_{st} = 2$, TSmax $= 1500$, Cycles $= 4$. The running timetable of the school, produced manually with the interactive support of a commercial package (which schedules one class at a time using some direct heuristic), has an objective value of 279.

## 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 do 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, they offer a great advantage (over constructive methods and genetic algorithms) for interactive timetabling, which is generally necessary for finding a solution practically agreeable by the staff of the school. 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.

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. They generally are able to schedule 90-95% of the lectures, leaving the user with the problem to fit in to remaining 5-10% of the timetable, which can be extremely difficult.

**Acknowledgments.** I wish to thank Giansalvatore Mecca and Elisabetta Ranieri for their cooperation and for making available to me all the data of the schools. 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.

Colorni, A.; Dorigo, M.; and Maniezzo, V. 1992. A genetic algorithm to solve the timetable problem. Technical Report 90-060 revised, Politecnico di Milano, Italy.

Cooper, T. B., and 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.; and Shamir, A. 1976. On the complexity of timetabling and multicommodity flow problems. *SIAM Journal of Computation* 5(4):691–703.

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

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

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

Neufeld, G. A., and 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., and de Werra, D. 1983. Some experiments with a timetabling system. *OR Spektrum* 3:199–204.

Schaerf, A., and 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*, 313–323.

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

Schaerf, A. 1996. Tabu search techniques for large high-school timetabling problems. Technical Report CS-R9611, CWI, Amsterdam, NL. Available at `http://www.cwi.nl/ftp/CWIreports/AP`.

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

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

Yoshikawa, M.; Kaneko, K.; Nomura, Y.; and Watanabe, M. 1996. A constraint-based high school scheduling system. *IEEE Expert*, February 1996:63–72.