Artificial Intelligence Review **13:** 87–127, 1999.

A Survey of Automated Timetabling

A. SCHAERF
Dipartimento di Ingegneria Elettria Gestionale e Meccanica, Università di Udine, Via delle Scienze 208, 33100 Udine, Italy
E-mail: schaerf@uniud.it

Abstract. The timetabling problem consists in scheduling a sequence of lectures between teachers and students in a prefixed period of time (typically a week), satisfying a set of constraints of various types. A large number of variants of the timetabling problem have been proposed in the literature, which differ from each other based on the type of institution involved (university or school) and the type of constraints. This problem, that has been traditionally considered in the operational research field, has recently been tackled with techniques belonging also to Artificial Intelligence (e.g., *genetic algorithms*, *tabu search*, and *constraint satisfaction*). In this paper, we survey the various formulations of the problem, and the techniques and algorithms used for its solution.

Key words: timetabling, combinatorial optimization, scheduling, local search, heuristics

1. Introduction

The timetabling problem consists in scheduling a sequence of lectures between teachers and students in a prefixed period of time (typically a week), satisfying a set of constraints of various types.

The manual solution of the timetabling problem usually requires many person-days of work. In addition, the solution obtained may be unsatisfactory in some respect; for example a student may not be able to take the courses he/she wants because they are scheduled at the same time.

For the above reason, a considerable attention has been devoted to automated timetabling. During the last thirty years, starting with (Gotlieb 1963), many papers related to automated timetabling have been published in conferences proceedings and journals. In addition, several applications have been developed and employed with a good success.

In this section, we informally describe the timetabling problem and the issues associated with it. In the subsequent sections, we state the problem precisely and discuss its solution techniques.

1.1. *Different problems and formulations*

A large number of variants of the timetabling problem have been proposed in the literature, which differ from each other based on the type of institution involved (university or school) and the type of constraints. We classify the timetabling problems into three main classes:

School timetabling: The weekly scheduling for all the classes of a school, avoiding teachers meeting two classes at the same time, and vice versa;

Course timetabling: The weekly scheduling for all the lectures of a set of university courses, minimizing the overlaps of lectures of courses having common students;

Examination timetabling: The scheduling for the exams of a set of university courses, avoiding overlap of exams of courses having common students, and spreading the exams for the students as much as possible.

Based on this classification, we develop a separate discussion for each of the three problems and we devote a section to each of them. However, such classification is not strict, in the sense that there are some specific problems that can fall between two classes, and cannot be easily placed within the above classification. For example, the timetabling of a specific school which gives large freedom to the student regarding the set of courses can be similar to a course timetabling problem.

1.2. *Feasibility, optimality, and complexity*

In some cases, the timetabling problem consists of finding any timetable that satisfies all the constraints. In these cases, the problem is formulated as a *search problem*.

In other cases, the problem is formulated as an *optimization problem*. That is, what is required is a timetable that satisfies all the *hard* constraints and minimizes (or maximizes) a given objective function which embeds the *soft* constraints. As shown later, in some approaches, the optimization formulation is just a means to apply optimization techniques to a search problem. In this case, what is minimized is the so-called *distance to feasibility*. Even when the problem is a true optimization problem, the distance to feasibility may be included in the objective function. This is generally done to facilitate the search for the best solution.

In both cases (search and optimization), we define the *underlying problem*, which is the problem of deciding if there exists a solution, in the case of a search problem, and the problem of deciding if there exists a solution with a

given value of the objective function, in the case of an optimization problem. When we mention the complexity of the problem, we refer to the complexity of the underlying decision problem.

As we will see later in the paper, the underlying problem is NP-complete in almost all variants. Therefore, an exact solution is achievable only for small cases (e.g., less than 10 courses), whereas real instances usually may involve a few hundreds of courses. It follows that only *heuristic* methods (Pearl 1984) are feasible, which have not guaranteed to reach the (optimal) solution.

1.3. *Solution approaches*

Most of the early techniques (see Schmidt and Strohlein 1979) were based on a simulation of the human way of solving the problem. All such techniques, that we call *direct heuristics*, were based on a *successive augmentation*. That is, a partial timetable is extended, lecture by lecture, until all lectures have been scheduled. The underlying idea of all approaches is "schedule the most constrained lecture first", and they differ only on the meaning they give to the expression 'most constrained'.

Later on, researchers started to apply general techniques to this problem. We therefore see algorithms based on *integer programming*, *network flow*, and others. In addition, the problem has also been tackled by reducing it to a well-studied problem: *graph coloring*.

More recently, some approaches based on search techniques used also in Artificial Intelligence appeared in the literature; among others, we have *simulated annealing*, *tabu search*, *genetic algorithms*, and *constraint satisfaction*.

In this paper, we survey the solution techniques, putting the emphasis on the most recent approaches in general, and on Artificial Intelligence techniques in particular.

Notice that we include in our list of techniques also some items, e.g., *logic programming*, that are general tools for the development of the solution, rather than real solution techniques. In those cases, we specify also the technique implemented using the given tool.

1.4. *Previous surveys*

The literature on timetabling includes several surveys. We now briefly discuss their scope and contribution.

Schmidt and Strohlein (1979) provide an annotated bibliography including more than 200 entries, listing virtually all papers on the field that appeared up to 1979.

Junginger (1986) describes the research in Germany on the school timetabling problem. In particular, he describes the various software products

implemented, and their actual utilization by the staffs of the institutions. The paper also describes the underlying approaches, most of which are based on direct heuristics.

De Werra (1985) states the various problems in a formal way, and provides different formulations for them. He also describes the most important approaches to the problem (up to date), stressing the graph-theoretic ones. We follow mostly his paper for the terminology and the problem formulations.

Carter (1986) surveys the approaches to the examination timetable problem. He mainly focuses on the approaches based on the reduction to the graph coloring problems.

Corne et al. (1994a) provide a survey of the application of genetic algorithms to timetabling. The paper discusses also future perspectives of such approach, and compares its results obtained so far with respect to some other approaches.

Other surveys are given in (Dempster et al. 1973; Hilton 1981; Klein 1983; Vincke 1984; Ferland et al. 1986).

There is also a Web page and a mailing list for the timetabling community, which includes bibliographies and papers.[1]

1.5. *Interactive vs. batch timetabling*

Many authors believe that the timetabling problem cannot be completely automated. The reason is twofold: On the one hand, there are reasons that make one timetable better than another one that cannot easily be expressed in an automatic system. On the other hand, since the search space is usually huge, a human intervention may bias the search toward promising directions that the system by itself might be not able to find. For the above reasons, most of the systems allow the user at least to adjust manually the final output. Some systems however require a much larger human intervention, so that we call them *interactive* (or *semi-automatic*) timetabling systems.

The systems described in (Klingen 1981; Chahal and De Werra 1989; Dinkel et al. 1989; Wong and Ng 1990; Mathaisel and Comm 1991; Boufflet et al. 1995) are examples of interactive systems.

In this paper, we do not deal with the issues specifically related to interactive systems, and we concentrate on *batch* systems.

1.6. *Outline of the paper*

In Sections 2, 3, and 4 we discuss in detail school timetabling, course timetabling, and examination timetabling, respectively. In Section 5, we discuss some general issues that are common to all three problems. In

Section 6, we briefly describe some related problems. In Section 7, we discuss some possible research directions.

Some of the techniques described in Sections 2, 3, and 4 are sketched in the appendices.[2] In particular, Appendix A describes the graph coloring problem and the main approaches to its solution. Appendices B, C, and D illustrate three general techniques for the solution of optimization problems, namely *tabu search*, *simulated annealing*, and *genetic algorithms*.

2. School timetabling

In this section, we describe in detail the school timetabling problem, also known as *class/teacher model*. We start by describing a simplified version, which can be solved in polynomial time; thereafter, we move to the basic formulation. It is still not a "real" problem, but it has the minimal set of constraints that makes it a hard problem, and its solution requires the heuristic techniques employed also for the more complex cases. Subsequently, we introduce the corresponding optimization problem, and describe some of its variants considered in the literature. Finally, we discuss solution techniques and approaches.

2.1. *Simplified polynomial problem*

Let $c_1, \ldots, c_m$ be m classes, $t_1, \ldots, t_n$ be n teachers and $1, \ldots, p$ be p periods. We are also given a non-negative integer matrix $R_{m \times n}$, called *Requirements matrix*, where r_{ij} is the number of lectures given by teacher t_j to class c_i.

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. The mathematical formulation is as follows (de Werra 1985):

$$\boxed{\text{TTP1}} \quad find \quad x_{ijk} \quad (i = 1..m; j = 1..n; k = 1..p)$$

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

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

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

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

where $x_{ijk} = 1$ if class c_i and teacher t_j meet at period k, and $x_{ijk} = 0$ otherwise.

Constraints (1) ensure that each teacher gives the right number of lectures to each class. Constraints (2) (resp. Constraints (3)) ensure that each teacher (resp. class) is involved in at most one lecture for each period.

Even et al. (1976) prove that there exists always a solution of this problem, unless a teacher or a class is required to be involved in more than p lectures. More precisely, there exists a solution if and only if

$$\sum_{i=1}^{m} r_{ij} \leq p \quad (j = 1..n) \tag{5}$$

$$\sum_{j=1}^{n} r_{ij} \leq p \quad (i = 1..m) \tag{6}$$

In order to solve TTP1, we may associate to an instance of the problem a bipartite *multigraph*: Classes and teachers are associated to vertices, and each class c_i is linked to each teacher t_j by r_{ij} parallel edges. The solution technique employed in (Even et al. 1976) is based on finding a sequence of maximal *matchings* in the resulting bipartite multigraph, where a matching is a set of edges with no common nodes.

Regarding the complexity of the method, Hopcroft and Karp (1973) prove that the required matching can be found in polynomial time with respect to the size of the multigraph. Since the method in (Even et al. 1976) requires p matchings, and the size of the multigraph involved is polynomial w.r.t. n, m, p, the whole method runs in polynomial time.

Alternatively, the problem can be reduced to a problem of *edge coloring* on graphs (de Werra 1985): Given p colors (each period corresponding to a color), the problem consists of finding an assignment of a color to each edge such that no two adjacent edges have the same color. Thereafter, the variable x_{ijk} gets value 1 if one of the edges between c_i and t_j gets color k.

De Werra considers also some variants of TTP1 which are still solvable in polynomial time. He considers the possibility that a teacher (and a class) can be involved in more than one lecture for each period. In such variant a period represents not an atomic time slot but a set of them (for example, a day). He also considers the case in which the lectures are constrained so that they must be spread as much as possible throughout all the periods.

2.2. *Basic search problem*

The problem TTP1 does not include any constraints on the possible scheduling of the lectures. In real instances, instead, we must take into account the possibility that a teacher (or a class) is unavailable at a given time.

We now introduce the school timetabling problem with *unavailabilities* of teachers and classes. The following formulation is due to (Junginger 1986);

alternative ones can be found for example in (Even et al. 1976; de Werra 1985; Garey and Johnson 1979, SS19, p. 243). Junginger introduces two binary matrices $T_{m\times p}$ and $C_{n\times p}$ such that $t_{ik} = 1$ (resp. $c_{jk} = 1$) if teacher t_i (resp. class c_j) is available at period k, and $t_{ik} = 0$ (resp. $c_{jk} = 0$) otherwise. Thereafter, he replaces Constraints (2) and (3) in TTP1 by Constraints (7) and (8) as below:

$$
\begin{aligned}
\boxed{\text{TTP2}} \qquad & find \quad x_{ijk} \quad (i = 1..m; j = 1..n; k = 1..p) \\
s.t. \quad & \sum_{k=1}^{p} x_{ijk} = r_{ij} \quad (i = 1..m; j = 1..n) \\
& \sum_{j=1}^{n} x_{ijk} \leq t_{ik} \quad (i = 1..m; k = 1..p) \qquad (7) \\
& \sum_{i=1}^{m} x_{ijk} \leq c_{jk} \quad (j = 1..n; k = 1..p) \qquad (8) \\
& x_{ijk} = 0 \text{ or } 1 \quad (i = 1..m; j = 1..n; k = 1..p)
\end{aligned}
$$

De Werra (1985) considers also constraints due to *preassignments*: A particular lecture can be imposed to be scheduled at a given time. Preassignments can be expressed adding a set of constraints of the following form

$$x_{ijk} \geq p_{ijk} \quad (i = 1..m; j = 1..n; k = 1..p) \qquad (9)$$

where $p_{ijk} = 0$ if there is no preassignment, and $p_{ijk} = 1$ when a lecture of teacher t_j to class c_i is preassigned to period k. de Werra also shows that unavailability can be expressed as preassignments with *dummy* classes or teachers.

The problem TTP2 has been shown NP-complete by Even et al. (1976) through a reduction from 3-SAT (Garey and Johnson 1979, LO2, p. 259). Even et al. also prove that the problem is polynomial for the special case in which classes are always available and each teacher is available for exactly two periods.

2.3. *Optimization problem*

The problem TTP2 is a search problem, whose solution is any feasible timetable. However, in real applications a feasible timetable can be better than another one, and the goal is to find the optimal one. This consideration forces us to formulate the timetabling problem as an optimization problem with an objective function to minimize (or maximize).

Junginger (1986) proposes to add to the basic problem TTP2 the following objective function

$$min \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{p} d_{ijk} x_{ijk} \tag{10}$$

where a large d_{ijk} is assigned to periods k in which a lecture of teacher t_j to class c_i is less desirable.

Colorni et al. (1992) introduce a more complex objective function, which includes several aspects of the timetable. We don't show such function in detail here, we just mention that it is based on the following quantities (with decreasing weight):

- the *didactic cost*: e.g., spreading the lectures over the whole week;
- the *organizational cost*: e.g., having a teacher available for possible temporary teaching posts;
- the *personal cost*: e.g., a specific day-off for each teacher.

A different approach is taken in (Yoshikawa et al. 1996). They introduce a constraint language, and associate a penalty for each constraint violated. Their objective is to minimize the overall penalty. As an example of constraint violation, they consider the possibility that a teacher is forced to teach in a period in which he/she is not available.

2.4. *Variants of the problem*

Several variants of the above basic problem have been proposed in the literature so as to deal with real-life problems. We now list some of the most popular ones.

2.4.1. *Simultaneous lectures*

Real timetables usually include some lectures that are simultaneously given to more than one class. For example, in some high schools a gymnastic lesson generally involves two classes together. Obviously, if a simultaneous lesson is scheduled at a given time, all the classes involved cannot be scheduled for any other lectures at that time. Simultaneous lessons are taken into account in (Yoshikawa et al. 1996; Schaerf 1996b).

2.4.2. *Teachers for more than one subject*

So far we have assumed that a class must take a number of lectures with a set of specific teachers. Cooper and Kingston (1993), instead, consider the case that a class must take some specific subjects, and that different teachers may teach the same subject. In addition, a teacher may teach different subjects. However, they assume that all the lectures to a specific class of a given subject are given by the same teacher.

2.4.3. *Uncovered classes*
Especially in primary schools, it is required that classes are not left "uncovered" during certain periods.

In some schools, the sum of the requirements for the classes exactly equals p the number of periods. In these cases, classes are necessarily always covered. Conversely, if the requirements of a class are less than p, then it is usually required that classes can be uncovered only in the first or the last period of the day.

In (Schaerf 1996b) such constraint is taken into account in the following way: For each class it is given a set of period in which it must be necessarily involves in one lecture, whereas it can be uncovered foe the others.

2.4.4. *Special rooms*
The availability of rooms is not taken into account in the basic problem because of the implicit assumption that each class has a dedicated room. However, some particular lectures may require special equipments, such as science labs or music rooms. The number of special rooms is obviously limited, and therefore, there is the additional constraint that no more than a given number of lectures requiring a special room can be scheduled at the same period.

2.5. *Solution techniques and approaches*

We now list the solution techniques and approaches proposed in the literature. For this list, we roughly follow the chronological order.

As mentioned in Section 1.3, we distinguish between techniques and approaches; with the term technique we mean an algorithm or a set of algorithms for the solution of the problem (e.g., genetic algorithms). An approach instead is a general framework for the development of a solution algorithm (e.g., constraint logic programming).

2.5.1. *Direct heuristics*
Direct heuristics usually fill up the complete timetable with one lecture (or one group of lectures) at a time as far as no-conflicts arise. At that point they start making some swapping so as to accommodate other lectures.

A typical example of this method is the system SCHOLA described in (Junginger 1986). The system is based on the following three strategies:

A: Assign the *most urgent* lecture to the *most favorable* period for that lecture.

B: When a period can be used only for one lecture, assign the period to that lecture.

C: Move an already-scheduled lecture to a free period so as to leave the period for the lecture that we are currently trying to schedule.

A lecture is "urgent" when it is tightly constrained; that is, when the teacher (and the class) has little availability and many lectures to give. A period is "favorable" when few other lectures can be scheduled at that period based on the availability of the other teachers and classes.

The system SCHOLA schedules the lectures alternating Strategies A and B as much as possible. When no more lecture can be scheduled in this way, it starts using Strategy C.

Strategy A is the core of the system, and it is employed almost in all systems, with different way of defining urgency and favorableness. The use of Strategy B might prevent Strategy A to enter in dead-ends. Strategy C provides a limited form of backtracking to recover from the "mistakes" of Strategy A.

Many of the early papers propose some direct heuristics for the solution of the timetable problem. We refer to (Schmidt and Strohlein 1979) for a comprehensive list of such algorithms up to that date.

The algorithm in (Papoulias 1980) can also be considered a direct heuristics. Such paper stresses on the requirement that lectures must be spread across days. The favorableness of a period for a lecture is therefore based also on the fact that another lecture of the same teacher to the same class has not been already assigned to a consecutive day.

2.5.2. *Reduction to graph coloring*

Neufeld and Tartar (1974) propose a reduction to the graph coloring problem (Appendix A). In their reduction, each lecture is associated with a vertex in a graph, and there is an edge between each pair of lectures that cannot be scheduled at the same time. In particular, lectures that share a common teacher or a common class (or both) are joined. Unavailabilities and preassignments are managed imposing some *external constraints* on the colorability of specific vertices of the graph.

A coloration of the resulting graph, respecting the external constraints, can be easily turned into a timetable, by assigning a period to each color, and consequently scheduling the lectures corresponding to a vertex to the period corresponding to its color.

In addition, Neufeld and Tartar show that the problem of the coloration of a graph with the given type of external constraints can be transformed into the colorability of a graph without any constraints.

2.5.3. *Network flow techniques*

Ostermann and de Werra (1983) reduce the timetabling problem to a sequence of network flow problems. The general network model can be formulated as follows:

$$\begin{aligned} min \quad & \sum_{j=1}^{m} c_j x_j, \\ s.t. \quad & Ax = b, \\ & l \leq x \leq u \end{aligned}$$

where $A_{n\times m}$ is the vertex edge incidence matrix, $b_{n\times 1}$ is the vector of supplies, and $u_{1\times m}, l_{1\times m}$ are vectors of capacities and lower bounds.

Ostermann and de Werra create a network for each period so that the flow in the network identifies the lectures given in that period. De Werra (1985) proposes a similar method creating a network for each class. We now briefly describe the latter one.

For a given class c_i do the following steps: (i) introduce a vertex for each period k and each teacher t_j; (ii) connect k with t_j if teacher t_j is available at period k and he/she has not been assigned to another class for period k in a previous network; (iii) introduce a source vertex s with edges (s, k) for all periods k and a sink vertex t with edges (t_j, t) for all teachers t_j; (iv) set both the capacities $u(t_j, t)$ and the lower bounds $l(t_j, t)$ to r_{ij}; (v) for all other edges $u = 1$ and $l = 0$.

The solution of the network, which is always integer due to the total unimodularity property (Papadimitriou and Steiglitz 1982, pp. 316–318), gives a schedule for all the lectures for the given class.

The construction of the network is repeated for all classes and eventually, if a solution is found for all networks, it leads to a complete timetable. Obviously, since there is no backtracking on the classes already scheduled, we have no guarantee that the solution is found whenever it exists.

A network flow approach has been recently used by Ikeda et al. (1995) for the implementation of the SECTA system.

2.5.4. *Genetic algorithms*

A genetic algorithm (Appendix D) has been applied to the school timetabling problem by Colorni et al. (1992). They consider the optimization problem with the objective function mentioned in Section 2.3.

Infeasible timetables are also included in the search space of the algorithms. The objective function embeds the number of infeasibilities. In order to bias the search toward feasible timetables the infeasibilities are given a very high weight in the objective function w.r.t. other constraints.

A solution is represented as a matrix $M_{m \times p}$ such that the row i^{th} of M represents the timetable for teacher t_i. In particular, each entry m_{ik} contains the name of the class that the teacher is meeting at period k.

The crossover operator is applied to two timetables T_1 and T_2 in the following way: There is a local fitness function that computes the fitness of the schedule of a specific teacher. The rows of T_1 are sorted in order of decreasing local fitness and the best k rows are taken together with the other $m - k$ rows taken from T_2. The second offspring is obtained from the unutilized rows of T_1 and T_2. The value of k is determined on the basis of the local fitness of both T_1 and T_2.

The mutation operator takes h contiguous genes and swaps them with another h contiguous non-overlapping ones belonging to the same row. The algorithm includes also a local search phase that moves a solution to its local optimum.

Genetic algorithm have been more extensively applied for exam timetabling. See Section 4.4.4 for more cases of applications of genetic algorithms to timetabling problems.

2.5.5. *Simulated annealing*

Abramson (1991) applies simulated annealing (Appendix C) to school timetabling. He also considers, as an extension, the possibility that two different classes may have common students. With this extension, his framework may fall also into the course timetabling category.

A solution is described by a list of sets of lectures, one list for each period. Given a solution, the choice of the neighbor solution is performed by selecting at random a period and a lecture in the selected period, and moving the lecture to a different period randomly chosen.

The objective function f (to be minimized) is a weighted sum of the number of conflicts of classes and conflicts of teachers.[3]

Abramson chooses a cooling rate of 0.9, although he experimented also with various other values. The number of iterations performed is on the order of three millions.

2.5.6. *Logic programming approach*

Kang and White (1992) propose the logic programming approach to the school timetable problem. In particular, they use PROLOG as the implementation language for their timetabling program. The main advantage of this approach is the ability to express in a declarative way the constraints involved in the problem.

The full backtracking capability of the PROLOG machine is overridden by a heuristics that allows only for a limited attempt to reschedule assignments that create conflicts. In particular, when a lecture becomes unschedulable,

the procedure finds an "equivalent" (defined in the paper) lecture already scheduled and reassigns it to a different period (similarly to Strategy C of Section 2.5.1). If no equivalent lecture can be moved to a different period, leaving a feasible period for the currently-processed lecture, it is put into a list of lectures which will be manually scheduled later.

2.5.7. *Constraint-based approach*
The work in (Yoshikawa et al. 1996) proposes the use of a general-purpose *Constraint Relaxation Problem* solver, called COASTOOL. In a constraint relaxation problem, a given penalty is assigned to each constraint and the objective is to find an assignment of the problem variables that minimizes the total penalty.

The constraint language allows the user to express several types of constraints. For example, it is possible to express the unavailabilities of a given teacher. The following constraint states that the set lectures of the teacher Smith, identified by the set `SmithLessons`, cannot take place during the set of period representing the unavailabilities of Smith, identified by the set `SmithAbsence`.

```
(define-constraint PartTimerSmith
  :object ((:set lesson SmithLessons))
  :variables ((v lesson))
  :condition (not (is-a v SmithAbsence))
  :penalty 10)
```

The solution method combines a greedy algorithm for finding an initial solution and a hill-climbing procedure for the optimization phase (Minton et al. 1992). The employment of a smart greedy algorithm, called *Really-Fully-Lookahead* algorithm, for the initialization phase, and a strongly biased optimization algorithm, for the optimization phase, allows the method to find a high-quality solution in a reasonable amount of time.

Constraint logic programming languages have been extensively used for course timetabling problems. See Section 3.4.6 for their presentation.

2.5.8. *Tabu search*
Costa (1994), Alvarez-Valdes et al. (1996), and Schaerf (1996b) apply tabu search (Appendix B) to a quite standard high school timetabling problem. Alvarez-Valdes et al. (1996) and Schaerf (1996b) also experiments with complex move types and with combinations of tabu search with other local search techniques.

The representation chosen by Schaerf (1996b) is the same as (Colorni et al. 1992), that is a solution is represented as a matrix $M_{m\times p}$ such that each entry m_{ik} contains the name of the class that the teacher is meeting at period

k. A move consists in exchanging two lectures for a given teacher or moving a lecture to a different period. Costa (1994) instead 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.

Tabu search has been previously applied to course scheduling in (Hertz 1991; Hertz 1992). See Section 3.4.4 for a more detail presentation of the application of tabu search to timetabling problems.

2.5.9. *Combination of methods*

The algorithm in (Cooper and Kingston 1993) combines several heuristics. As a core strategy, it uses a form of bipartite graph matching, that they call *meta-matching*.

The algorithm identifies groups of lectures that must be scheduled all at different times, which are called *meeting-sets*. Thereafter, it iteratively performs a matching between lectures belonging to a meeting set, on one side, and the so-called *prototimes* on the other; the prototimes are variable time slots that are later assigned to actual periods in a successive phase.

The algorithm is improved by choosing among the possible assignments those that assign as many lectures as possible to the prototimes already used for previously scheduled meeting-sets. This enhancement helps in finding a feasible timetable in presence of many large meeting-sets.

As already mentioned, the cited paper solves an extension of the basic problem which requires to deal with different interchangeable resources, such as teachers and rooms. Actual resources are assigned to lectures by a specific procedure. Such procedure, depending on the number of resources involved may use a brute-force algorithm or a covering technique called *beam search*.

3. University course timetabling

The university course timetabling problem consists in scheduling a set of lectures for each course within a given number of rooms and time periods. The main difference from the school problem is that university courses can have common students, whereas school classes are disjoint sets of students. If two courses have common students then they conflict, and they cannot be scheduled at the same period. Moreover, school teachers always teach more than one class, whereas in universities, a professor usually teaches only one course. In addition, in the university problem, availability of rooms (and their size) plays an important role, whereas in the school problem they are often

neglected because, in most cases, we can assume that each class has its own room.

We start describing the basic formulation of the problem. Thereafter, we introduce the optimization problem and we discuss the variants of the problem. Finally, we present the solution techniques and approaches.

3.1. *Basic search problem*

There are various formulations of the course timetabling problem (see e.g., Tripathy 1992). The one given here is taken from (de Werra 1985).

There are q courses $K_1, \ldots, K_q$, and for each i, course K_i consists of k_i lectures. There are r *curricula* $S_1, \ldots, S_r$, which are groups of courses that have common students. This means that courses in S_l must be scheduled all at different times. The number of periods is p, and l_k is the maximum number of lectures that can be scheduled at period k (i.e. the number of rooms available at period k). The formulation is the following:

$$\boxed{\text{TTP3}} \qquad find \quad y_{ik} \qquad (i = 1..q; k = 1..p)$$

$$s.t. \quad \sum_{k=1}^{p} y_{ik} = k_i \qquad (i = 1..q) \tag{11}$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k \qquad (k = 1..p) \tag{12}$$

$$\sum_{i \in S_l} y_{ik} \leq 1 \qquad (l = 1..r; k = 1..p) \tag{13}$$

$$y_{ik} = 0 \text{ or } 1 \qquad (i = 1..q; k = 1..p) \tag{14}$$

where $y_{ik} = 1$ if a lecture of course K_i is scheduled at period k, and $y_{ik} = 0$ otherwise.

Constraints (11) impose that each course is composed of the correct number of lectures. Constraints (12) enforce that at each time there aren't more lectures than rooms. Constraints (13) prevent conflicting lectures to be scheduled at the same period.

Problem TTP3 can be shown to be NP-complete through a simple reduction from the graph coloring problem.

A formulation equivalent to TTP3 is based on the *conflicts matrix* instead of the curricula. The conflict matrix $C_{q \times q}$ is a binary matrix such that $c_{ij} = 1$ if courses K_i and K_j have common students, and $c_{ij} = 0$ otherwise.

3.2. *Optimization problem*

De Werra (1985) includes the following objective function in the problem TTP3

$$max \quad \sum_{i=1}^{q} \sum_{k=1}^{p} d_{ik} y_{ik}$$

where d_{ik} is the desiderability of having a lecture of course K_i at period k.

Tripathy (1992) considers a conflict matrix $C_{q \times q}$ with integer values, such that c_{ij} represents the number of students taking both courses K_i and K_j. In this way, c_{ij} represents also a measure of dissatisfaction in case a lecture of K_i and a lecture of K_j are scheduled at the same time. Tripathy tries to minimize the global dissatisfaction obtained as the sum of all the dissatisfactions of the above type.

Several authors, after (Eiselt and Laporte 1987), split the requirements into hard and soft ones. The hard requirements are included in the constraints and they define the search space, whereas the soft ones are included in the objective function (see e.g., Aubin and Ferland 1989). Soft requirements generally include event spreading constraints and room capacity constraints.

3.3. *Variants of the problem*

We now briefly discuss some of the most common variants of the basic problem proposed above.

3.3.1. *Unavailabilities and preassignments*

Preassignments and unavailability are not included in the basic model of Section 3.1. They can be included in the model exactly in the same way it was done in Section 2 for the school timetabling problem.

However, in the university case the problem is NP-complete even without them. Therefore, unavailabilities and preassignments do not add worst-case complexity as in the school case.

3.3.2. *Multiple sections and grouping subproblem*

In some universities, some courses are repeated more than once during the week. In particular, those courses involving a large number of students and belonging to several curricula are split into *multiple sections*. The creation of different sections for a course may help to reduce the number of conflicts in a timetable. For example, suppose that curriculum S_1 involves courses K_1 and K_2 and curriculum S_2 involves courses K_1 and K_3. Suppose also that a lecture of K_2 takes place at time p and a lecture of K_3 at time q. In this case, the lectures of course K_1 can take place neither at time p nor at time q.

However, if course K_1 is given in two sections, then the lecture of one section can take place at time p, and the lecture of the other section at time q.

Given a certain timetable, the problem of assigning the students taking a particular curriculum to a specific section of a course in order to minimize conflicts is called *grouping subproblem* (or *student sectioning*).

The grouping subproblem is considered in (Laporte and Desroches 1986; Aubin and Ferland 1989; Hertz 1991; Tripathy 1992). We refer to (Laporte and Desroches 1986) for a mathematical formulation of the grouping subproblem.

Laporte and Desroches (1986) take an approach analogous to the one they take in (Laporte and Desroches 1984) for the examination timetabling (see Section 4). They formulate the problem as an optimization problem splitting the requirements into hard and soft ones. The hard requirements are: (i) A student cannot be in two lectures at one time and (ii) a student cannot have two adjacent lectures in two different campuses. Soft requirements include number of lectures in a single day and changes of campus in a free hour.

The problem is then solved in two phases: In the first one the algorithm searches for an admissible solution, whereas in the second one it searches for a local optimum.

In those approaches that take into account the grouping subproblem, the overall timetabling algorithm is based on the alternate solution of the pure timetable problem and the grouping one. For example, Hertz (1991) uses an iterative procedure that at each iteration solves both problems, first the timetabling and thereafter the grouping subproblem.

3.3.3. *Periods of variable length*

So far we have considered all lectures of one-period length. Several authors consider also lectures of different length.

In all approaches the lectures are allowed to last an integer number of periods; for example, in (Ferland and Roy 1985) lectures may last one or two or three periods.

Therefore, for each lecture we have to consider both the starting time and the length. In this case, Constraints (11) and (12) are replaced by constraints that take into account the fact that two lectures l_i and l_j, starting at time p and $q > p$, are conflicting also if $q - p < d_{l_i}$, where d_{l_i} is the length of lecture l_i.

Hertz (1992) tackles the more general situation in which lectures may have a variable length. In particular, Hertz allows one to specify that a course is composed of a certain number of periods which can be given in a variable number of lectures of different length. For example, a course composed of twelve time periods may consist of lectures of two or three time periods. In this case the course can be given in four, five or six lectures of the following lengths: $(3, 3, 3, 3)$, $(2, 2, 2, 3, 3)$, $(2, 2, 2, 2, 2, 2)$.

3.3.4. *Classroom assignment subproblem*
The *classroom assignment* subproblem consists in assigning classes to rooms, given a fixed timetable. Carter and Tovey (1989) analyze in detail this problem and give various alternative formulations and variants. They also show in which cases the problem is tractable and in which it is NP-complete. In particular, they show that the problem becomes NP-complete when we impose the constraint that all lectures of a course must take place in the same room, whereas the problem is polynomial if we consider the assignment in each period independently of the other periods. We refer to Carter and Tovey for the exact formulation of this problem.

Ferland and Roy (1985) solve the classroom assignment subproblem by a reduction to a quadratic assignment problem.

3.4. *Solution techniques and approaches*

We now list the solution techniques proposed in the literature. As we did for the school timetabling problem, we try to follow the chronological order.

3.4.1. *Reduction to graph coloring*
De Werra (1985) shows how to reduce a course timetabling problem to graph coloring (Appendix A): Associate with each lecture l_i of each course K_j a vertex m_{ij}; for each course K_j introduce a clique between the vertices m_{ij} (for $i = 1, \ldots, q$). Introduce all edges between the clique for K_{j_1} and the clique for K_{j_2} whenever K_{j_1} and K_{j_2} are conflicting.

In case of unavailabilities, introduce a set of p new vertices, each one corresponding to a period. The new vertices are all connected to each other. This ensures that each one is assigned to a different color. If a course cannot have lectures at a given period, then all the vertices corresponding to the lectures of the course are connected to the vertex corresponding to the given period. Conversely, if a lecture must take place at a given time, then the vertex corresponding to that class is connected to all period vertices but the one representing the given period.

The reduction to graph coloring is applied, among the others, in (Selim 1988).

3.4.2. *Integer linear programming*
Several authors solved the timetabling problem using integer linear programming techniques (e.g., Breslaw 1986; Shin and Sullivan 1977; McClure and Wells 1984; Tripathy 1984; Ferland and Roy 1985; Tripathy 1992).

For example, Tripathy (1984) employs a *Lagrangian Relaxation* technique. In (Tripathy 1992), he extends the technique to deal also with the grouping subproblem.

Ferland and Roy (1985) formulate the problem as an assignment problem, and solve it through a reduction to a quadratic assignment problem.

3.4.3. *Network flow techniques*

Dyer and Mulvey, Mulvey, Chahal and de Werra, Dinkel et al. (1976, 1982, 1989, 1989) propose to use a network model as the core of the timetabling algorithm. The general formulation of a network model can be found in Section 2.5.3.

The network employed by Dinkel et al. contains three levels, plus a source and a sink vertex. The first level is the *Department Level* which includes a vertex for each department, such that all of these vertices are connected to the source. The second level is the *Faculty/Staff Level* which includes a vertex for each possible combination of teacher and course taught by the teacher; these vertices are connected to the vertices representing the departments to which the teachers belong. The third level is the *Room Size/Time Level*, which contains a vertex for each combination of room and time. Each vertex of this level is connected to a vertex of the second level only if the size of the room represented by the vertex in compatible with the number of students of the course represented by the other vertex. An edge between levels 2 and 3 represents a possible lecture.

The capacities and the lower bounds of edges representing the lectures are 0 and 1 respectively, and due to unimodularity, this ensures that the optimal solution to the problem will possess all integer values.

The coefficients of the objective function are assigned based on availabilities of teachers and rooms, and preferences of the teachers.

The network model can be solved in polynomial time; however it does not prevent the solution from assigning a single teacher to multiple lectures at the same time. The procedure therefore solves the problem and, if it finds a feasible solution, the process is over, otherwise a human intervention changes some of the weights manually so as to get rid of the reason of infeasibility. The procedure is executed several times until a feasible solution is obtained.

3.4.4. *Tabu search*

The tabu search technique (Appendix B) has been applied to university course timetabling in (Hertz 1991; Hertz 1992). The work in (Hertz 1991) considers the optimization problem with the grouping option, whereas (Hertz 1992) extends the approach to a more complex case which takes into account also lectures of different length.

In (Hertz 1991) the tabu search is applied to both the timetabling problem and the grouping subproblem. We describe only the application of tabu search to the pure timetabling problem. In addition, we simplify the treatment in

(Hertz 1991), and we do not consider constraints involving the location of the lecture rooms.

The application of tabu search to our specific problem requires the definition of the following entities: (i) a feasible solution; (ii) the neighbor relation and the neighbor selection procedure; (iii) the objective function to minimize; (iv) the initial solution.

In (Hertz 1991) the constraints, as usual, are split into hard and soft ones. However, a feasible solution is not defined based on the hard constraints because this does not guarantee that the corresponding search space is connected (w.r.t. the neighbor relation). For this reason, the concept of feasible solution for the search procedure is relaxed with respect to the concept of feasible timetable, and the procedure is allowed to pass also through infeasible timetables. In particular, Hertz considers a feasible solution also a timetable in which courses given at the same time share a common teacher or involve common students. All the other hard constraints are satisfied by feasible solutions.

The neighborhood $N(s)$ of a solution s consists of all timetables that can be obtained from s by assigning a lecture l to a different period t. Notice that, based on the above definition of feasible solution, this operation always results in a feasible solution. Among the neighbors of the current solution, the algorithm selects only the most *promising* ones. That is, it only considers reassignments of lectures which create at least one conflict in the current solution.

The objective function is the weighted sum of the number of teacher and student conflicts for each period.

Due to the way we defined a feasible solution, a starting solution can be easily selected at random.

3.4.5. *Rule-based approach*

A solution technique based on expert systems is given in (Meisels et al. 1991; Solotorevsky et al. 1994).

Solotorevsky et al. define a rule-based language, called RAPS, for specifying general resource allocation problems, and they use it, among the others, for a course timetabling problem. In particular, they consider lectures as activities and periods as resources to be assigned to the activities.

RAPS has five types of rules, namely *assignment rules*, *constraint rules*, *local change rules*, *context rules*, and *priority rules*.

Assignment rules assign lectures to periods, one at a time, and therefore they are the core of the system. These rules, like all the others, are supplied by the user, and thus the heuristics used is not predefined but is chosen by the user.

Constraint rules specify the constraints that the solution must satisfy. They are checked each time a new lecture is assigned to a period (i.e. a new activity is assigned to a resource). Constraint rules are split into positive and negative ones, that is, constraints that must be satisfied and constraints that must fail. Constraint rules allow to identify conflicts as soon as they arise.

Local change rules specify the action to perform when there is a lecture that the assignment rules are not able to tackle. The purpose of local change rule is to undo a previous assignment in order to create the opportunity to assign the current one (like Strategy C in Section 2.5.1).

Context rules select the *active context*. In fact, the system allows for multiple contexts: In different contexts the various lectures and periods may have different priority. The priority of the objects determines which object, among those of a given type, is processed first.

Priority rules determine the priorities of the lectures and the periods, in each context. The priorities are calculated each time a context is entered.

The system may work in two possible modes: *greedy* and *non-greedy*. In the greedy mode, when the assignment rules fail to make an assignment, the system selects a different lecture. Conversely, in the non-greedy mode, when a fail occurs the control is passed to the local change rules.

Other authors (Monfroglio 1988; Petrie et al. 1989; Dhar and Ranganathan 1990) make use of expert systems. For example, Dhar and Ranganathan (1990) propose the use of an expert system, called PROTEUS, for the allocation of teachers to courses, and compare it to integer programming techniques.

3.4.6. *Constraint logic programming approach*

A constraint logic programming (CLP) system (Jaffar and Lassez 1987) is a tool for modeling a specific search problem, which provides the ability to declare variables and their domains, and to place constraints.

In order to search for a solution, a CLP system generates values for the variables, propagating values through the constraints in order to prune parts of the solution space where inconsistencies are discovered. The basic method is therefore a backtrack search where the constraints allow the system to look ahead to the consequences of decisions and spot failure earlier.

For dealing with optimization problems the CLP systems provide a solution technique based on a form of depth first branch-and-bound search.

Azevedo and Barahona (1994) deal with the timetabling problem using a CLP language called DOMLOG. DOMLOG extends CHIP (Van Hentenryck 1991), a popular CLP language, with features such as user-defined heuristics and a flexible lookahead constraint solving.

In particular, DOMLOG allows for the possibility to specify a finite domain for the variables. In addition, the user can specify the heuristics for the selection of the value of a domain to assign first to a given variable.

Several other authors recently employed constraint logic programming for course timetabling with a good success. Frangouli et al. (1995) and Gueret et al. (1995) solve their course timetabling problem relying on the *finite domain* libraries of the logic programming language ECL^{i}PS^e (ECRC 1995) and CHIP, respectively. Henz and Würtz (1995) rely on the OZ system (Smolka 1995), which is a multi-paradigm concurrent constraint language.

3.4.7. *Others techniques and approaches*

Genetic algorithms have been used for course timetabling in (Ling 1992; Paechter et al. 1994). Since their approaches are similar to those employed for the examination timetabling we refer to Section 4.4.4 for a discussion of this technique.

A logic programming approach is taken by Fahrion and Dollansky (1992). They propose a PROLOG implementation of a heuristics based on a priority scheme on the lectures to be scheduled. Logic programming techniques are also used in (Monfroglio 1986) and (Ling, 1992).

Other approaches to the course timetabling problem are given in (Akkoyunlu 1973; Harwood and Lawless 1975; Selim 1983). For example, Harwood and Lawless employ a mixed integer goal programming.

Dowsland (1990) compares three different approaches based on *set partitioning*, graph coloring, and simulated annealing.

4. Examination timetabling

The examination timetabling problem requires the scheduling of a given number of exams (one for each course) within a given amount of time. The examination timetabling is similar to course timetabling, and it is difficult to make a clear distinction between the two problems. In fact, some specific problems can be formulated both as an examination timetabling problem and as a course timetabling one.

Nevertheless, it is possible to state some broadly-accepted differences between the two problems. Examination timetabling has the following characteristics (different from the course timetabling problem):

- There is only one exam for each subject.
- The conflict condition is generally strict. In fact, we can accept that a student is forced to skip a lecture due to overlapping, but not that a student skips an exam.
- There are different types of constraints, e.g. at most one exam per day for each student, and not too many consecutive exams for each student.
- The number p of periods may vary, in contrast to course timetabling where it is fixed.

– There can be more than one exam per room.

As in the previous sections, we start describing the basic formulation of the problem. Thereafter, we introduce the optimization problem, we discuss its variants, and we present the solution techniques and approaches.

4.1. *Basic search problem*

The basic search problem can be formulated in a similar way to university course timetabling.

There are q courses $K_1, \ldots, K_q$, and one exam for each course K_i. There are r groups of exams $S_1, \ldots, S_r$ such that in each S_l there are students that take all exams in S_l. The number of periods is p and l_k is the maximum number of exams that can be scheduled at period k (which is not necessarily the number of rooms, since more than one exam can take place in the same room).

$$\boxed{\text{TTP4}} \qquad find \quad y_{ik} \qquad (i = 1..q; k = 1..p)$$

$$s.t. \quad \sum_{k=1}^{p} y_{ik} = 1 \qquad (i = 1..q) \tag{15}$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k \qquad (k = 1..p) \tag{16}$$

$$\sum_{i \in S_l} y_{ik} \leq 1 \qquad (l = 1..r; k = 1..p) \tag{17}$$

$$y_{ik} = 0 \text{ or } 1 \qquad (i = 1..q; k = 1..p) \tag{18}$$

where $y_{ik} = 1$ if the exam of course K_i is scheduled at period k, and $y_{ik} = 0$ otherwise.

Like TTP3, the problem TTP4 can be shown to be NP-complete by a simple reduction from the graph coloring problem.

4.2. *Optimization problem*

The most common type of soft constraint considered in the literature on examination timetabling are the *second order constraints*. That is, the system should avoid that a student takes two exams in consecutive periods. To this aim, we can add the following objective function to the basic search problem.

$$\sum_{k=1}^{p-1} \sum_{l=1}^{r} \sum_{i,j \in S_l} y_{ik} y_{jk+1}$$

The above function counts the pairs of 1's corresponding to exams belonging to the same group S_l scheduled at adjacent periods. In fact, the product $y_{ik}y_{jk+1}$ gives 1 only if both y_{ik} and y_{jk+1} are 1.

Most authors (e.g., Mehta 1981) consider in the objective function also the number of students involved in each conflict. In particular, the penalty of a conflict is linearly related to this number.

Laporte and Desroches (1984) consider also higher order constraints. That is, they penalize also the fact that a student takes two exams in periods at distance three, four, or five.

Carter et al. (1994) generalize the above constraints and consider a penalty for the fact that a student is forced to take x exams in y consecutive periods. Their system considers as consecutive also the last period of a day and the first of the next day, and the two periods before and after lunch. Therefore, the dissatisfaction of a student that takes two exams in consecutive periods is not always the same. Carter et al. deal with this problem by adding dummy evening sessions.

Corne et al. (1993) consider a slightly different objective function. They penalize – in descending order of weight – a student taking (i) more than two exams in the same day, (ii) two exams in (real) consecutive periods, and (iii) two exams just before and just after lunch.

Corne et al. also consider a student taking two exams at once as part of the objective function (with the highest weight). Carter et al. instead allow the user to choose what is a hard constraint and what is a soft one.

4.3. *Variants of the problem*

Like in Sections 2 and 3, we now discuss the main variants of the basic examination timetabling problem.

4.3.1. *Unavailabilities and preassignments*

Like in the previous two problems, unavailabilities and preassignments are generally taken into account in real cases. They can be included in the model exactly in the same way as done in Section 2 for the school timetabling problem.

4.3.2. *Room assignment*

Exams must be assigned to rooms based on the number of students taking the exams and capacities of rooms.

Some authors (e.g., Carter et al. 1994) allow only one exam per room in a given period. In this case, the problem of assigning exams to actual rooms in an optimal way can be solved in polynomial time by means of a greedy algorithm.

Conversely, if we allow more exams per room (e.g., Burke et al. 1993), then the problem becomes NP-complete being a generalization of the *bin packing* problem (Garey and Johnson 1979, SR1, p. 226).

Some authors (e.g., Laporte and Desroches 1984) consider also different types of rooms, and some exams may only be held in certain types of rooms. In addition, some exams may be split into two or more rooms, in case the students do not fit in one single room.

4.3.3. *Minimize the length of the session*

Differently from course scheduling, in the examination case we may want to minimize the number of periods required to accomplish all the exams. In that case, the number of periods p becomes part of the objective function. Up to our knowledge, no author dealt specifically with this variant.

Notice that if we reduce the problem to graph coloring then the resulting problem is that of minimizing the number of colors. This is a well-studied problem (see e.g., Halldórsson 1993), and therefore the reduction to graph coloring can be profitable in this case.

4.4. *Solution techniques and approaches*

Like in the previous two sections, we now list the solution techniques proposed in the literature roughly in chronological order.

4.4.1. *Direct heuristics*

Several direct heuristics have been proposed in the literature. Some of them are described in (Carter 1986). We now present the approach followed in (Laporte and Desroches 1984), which has been modified and improved in (Carter et al. 1994).

The algorithm works in three phases: (i) find a feasible solution, (ii) improve the solution, (iii) allocate the rooms (allowing more exams per room).

Phase (i) schedules exams iteratively to periods based on the increase of the objective function caused by that allocation. When an exam cannot be scheduled, one or more exams previously scheduled are moved – at least cost – without creating infeasibilities. Each exam that cannot be moved is put back in the list of the exams yet unscheduled. In order to avoid infinite loops, Laporte and Desroches employ a mechanism resembling the tabu list in the tabu search technique (Appendix B): An exam cannot be put back by the same other exam more than a specified number of times.

Phase (ii) employs a simple descendent method. It identifies the move that would result in the maximum decrease in the objective function and performs it. The procedure stops when it reaches a local minimum.

Phase (iii) takes for each period a list of exams and a list of rooms. It iteratively assigns the exam with largest number of students to the largest room. If the exam fits perfectly in the room, they are both eliminated from the list of items to be allocated. If the room is too large, then the exam is eliminated and the *remaining part* of the room is considered as a new available room. Conversely, if the number of students does not fit in the room, then the room is eliminated and the *remaining part* of the students are considered as a new exam.

4.4.2. *Reduction to graph coloring*

The examination problem can be easily reduced to graph coloring (Appendix A) by associating each exam with a vertex and drawing an edge between each pair of conflicting exams. This approach is followed in (Mehta 1981).

Unfortunately, there is no obvious translation of the usual second order constraints into some properties on graphs. Therefore, examination timetabling problems with second order constraints cannot be easily formulated as graph coloring problems.

4.4.3. *Simulated annealing*

Eglese and Rand (1987) and Johnson (1990), among others, have used simulated annealing (Appendix C) for examination timetabling.

Eglese and Rand actually solve a problem slightly different from the examination timetabling, which is the timetabling of conference presentations (or seminars). In their formulation, a presentation can be repeated more than once if necessary. In addition the number of rooms is fixed and all rooms are used for all periods.

Each participant is required to provide the list of seminars that he/she wants to attend plus a reserve choice. The objective function is based on the number of participants that are not allowed to attend all the presentations they want. A higher cost is given to the case where the participant was not able to attend even his/her reserve choice.

The initial timetable is found using a heuristics, which schedules a presentation at a time, ordering them w.r.t. the number of participants that want to attend that presentation and the number of times it has been already scheduled.

The neighbors of a solution are obtained by cancelling a presentation in a given period and replacing it with another presentation. In this way, the number of rooms needed, for every period, never exceeds the number of rooms available (which is a hard constraint).

4.4.4. *Genetic algorithms*

Genetic algorithms (Appendix D) have been recently applied to examination timetabling by several authors (e.g., Burke et al. 1994; Corne et al. 1994b; Paechter 1994; Corne et al. 1993). They can be considered as the current main stream for tackling the examination problem.

We discuss the approach taken in (Corne et al. 1i993) and we refer to (Corne et al. 1994a) for a general discussion on the issues related to the use of genetic algorithms for the examination timetabling problem (e.g., direct vs. implicit representation).

The representation used by Corne et al. is simply a list of length q (the number of exams to be scheduled) of integers between 1 and p. The meaning of the list is that, if the i^{th} number in the list is t, then exam K_i is scheduled at period t.

The objective function counts the number of instances of the following *offenses*: (i) a student taking more than one exam at once (weight = 30); (ii) a student taking more than two exams in one day (weight = 10); (iii) a student taking two exams in consecutive periods on the same day (weight = 3); (iv) a student taking an exam just before and another one just after lunch on the same day (weight = 1).

The operator for recombination is based on a *fixed-point* uniform crossover. That is, for one child the bits in a fixed set of positions are chosen from one parent, the others are chosen from the other parent; vice versa for the other child. The number of positions is chosen to be half the solution length.

Recombination and mutation are applied with probabilities given by the two parameters p_R and p_M. In the experiments described in the paper, such parameters are set to 0.7 and 0.003, respectively.

Corne et al. also experiment with variants of the genetic algorithms, which improved the timetable produced. We refer to the cited paper for a description of these variants.

4.4.5. *Others techniques and approaches*

Balakrishnan et al. (1992) combine a network model with a Lagrangian relaxation technique. In particular, Balakrishnan et al. use Lagrangian relaxation to calculate accurate lower bounds for the network model. Their procedure iteratively calculates new solutions and new lower bound as long as they are close enough or the procedure exceeds the maximum number of iterations allowed.

5. General issues

In this section we briefly discuss some general issues that are common to all three problems. We first give some general comments on the implementations and their results, and thereafter we introduce a variant of the problem, called *continuous timetabling*, obtained by relaxing the constraints on integer values for the variables of the problem.

5.1. *Implementations and experiences*

Almost all papers in the literature describe a substantial software implementation. In addition, every paper usually presents the results of the application of the method to one or more test cases.

The success of the application is measured in two different ways, depending on the fact that the application deals with a search problem or an optimization problem. In the first case, the measure of success is the number of lectures scheduled with respect to the total. The results reported vary between 95% and 100% depending on the various cases. Such numbers are obviously influenced by how constrained the specific instance is, therefore they are not an accurate measure of the quality of the program.

In the second case, the measure of success is given by the value of the objective function for the optimal solution. Such value in some cases has a direct practical meaning; for example, it can represent the number of student dissatisfactions. Conversely, in some others it lacks a natural interpretation; for example, it can be obtained as a weighted sum of a number of features, in which case it is not directly readable.

In both cases, the results are generally compared with the manual ones. It is superfluous saying that in all papers the results produced by computer are superior to the manual ones. Unfortunately, the absence of a common definition of the various problems and of widely-accepted benchmarks prevents us from comparing the algorithms among each other.

The computational complexity of the proposed systems is generally determined only through the computing time. In almost all cases, such time is on the order of a few seconds. The hardware used varies from case to case, ranging from mainframes to PCs. The use of PCs is advisable for school timetabling because these are the type of computers generally owned by a school administrative office.

Some other papers are devoted to the discussion of implementations and experiences. For example, Sabin and Winter (1986) provide a discussion on the impact of timetabling on universities, and Junginger (1986) reports on the life cycle of the most popular school timetabling systems in Germany.

5.2. *On the role of Artificial Intelligence*

From Sections 2, 3, and 4, it emerges that the topics of timetabling were discussed in the last decades mainly in the Operational Research community. Only recently the problem has been tackled in the field of Artificial Intelligence, too.

The contribution of Artificial Intelligence can be recognized in the fact that the most promising modern heuristics, like genetic algorithms and tabu search, have their roots in Artificial Intelligence. These techniques, for large timetabling problems, seem to outperform traditional Operational Research methods.

In addition, a new approach based on constraint programming (an emerging Artificial Intelligence field) is recently becoming prominent in the research on timetabling. Programming with constraints allows for a great *flexibility* in the formulation of the problem, which is broadly recognized as a crucial issue for the success of a timetabling system. In fact, in many cases the definition of the problem's constraints and penalties is fuzzy, and it generally requires many refinements and interactions with the user. This is mainly due to the fact that is it not always easy to identify precisely which are the real constraints, and what constitutes a good timetable.

On the other hand, in many cases the problem is computationally so hard that constraint processing methods might be not sufficiently efficient. For example, constructive methods based on pure backtracking are generally not adequate. As confirmation of this fact most timetabling systems based on constraint logic programming tend to control the automatic backtracking mechanism with some local adjustment based on heuristics.

Among different timetabling problems, school timetabling seem to be the harder one, and this is confirmed by the fact that it is solved by specialized local search algorithm (see e.g., Costa 1994; Yoshikawa et al. 1996; Schaerf 1996b). Conversely, course timetabling is generally more complex to be formalized, but simpler from the combinatorial point of view. Therefore, it seems more suitable to be solved with constructive methods implemented in general purpose constraint logic programming languages (see e.g., Frangouli et al. 1995; Gueret et al. 1995; Henz and Würtz 1995).

5.3. *Continuous timetabling*

We call continuous timetabling the problem obtained by relaxing the constraint of integer values. For example, the continuous version of problem TTP1 is obtained by replacing Constraints (4) with the less restrictive ones

$$0 \leq x_{ijk} \leq 1 \quad (i = 1..m; j = 1..n; k = 1..p) \tag{19}$$

Therefore, the solution of a continuous timetabling problem can include also fractional values for the variables x_{ijk} of the problem.

The continuous timetabling is generally a polynomial problem, and therefore it is much simpler to solve than the integer problem.

The continuous version of the school timetabling problem has been investigated in (Smith and Sefton 1974; Clementson and Elphick 1982). The cited papers discuss necessary and sufficient conditions for the existence of a continuous timetable.

The solution to the continuous problem may have some utility for the real problem in case of long term scheduling, e.g. scheduling for an academic year. In such case, it can be interpreted as a set of timetables, differing from week to week. For example the value $x_{ijk} = 1/2$ could mean that teacher t_j meets class c_i at period k every second week.

6. Related problems

In this section we describe some problems related to timetabling. The solution techniques and the results obtained for such problems can be helpful for the study of the timetabling problem.

6.1. *Student scheduling*

The *student scheduling* problem consists of assigning a student to specific course sections for a given fixed timetable (see e.g., Busam 1967; Laporte and Desroches 1986; Feldman and Golumbic 1989).

In details, we assume that a given student is required to take a certain number of courses, which are given in one or more different sections. The courses are split into groups and there are conditions on the minimum and maximum number of courses to be taken for each group.

There are also other constraints on the number and the distribution of the periods in which the student is involved in a course.

A solution to the problem for a student is a set of courses and a specific section for each of them, such that there are no time conflicts and all the constraints are satisfied.

Feldman and Golumbic (1989) propose a solution technique for the student scheduling problem based on a *constraint satisfiability* algorithm.

6.2. *Other scheduling problems*

Many scheduling problems share some features with the timetabling problem. For example, the *sport leagues games scheduling* (Ferland and Fleurent 1991;

Schreuder 1992; Schaerf 1996a) and the *service timetable problem for transportation networks* (Odijk 1994) also account to the creation of a timetable. In that cases, though, the type of the constraints are different. The latter, for example, takes into account also the order of execution of the activities.

A scheduling problem that has been largely investigated in the literature is the *job shop scheduling*: A job consists of a sequence of operations, each of which must be processed on a specific machine. The operations of a job must be processed in the order specified by the sequence, and each machine can process at most one job at a time. The problem is to produce a schedule of n jobs on m machines that minimizes the time when all jobs have completed processing.

The job shop scheduling is NP-hard (Garey and Johnson 1979, SS18, p. 242). Several algorithms have been proposed for this problem (see e.g., Shmoys et al. 1991). In addition, there are several approximability results for them. The ideas for the solution of the job shop scheduling might be profitably exploited for the timetabling problem.

7. Possible research directions

So far we have described the state of the art of the research. In this section we list a number of topics that we consider possible future research directions.

7.1. *Investigate a specific technique*

The application of search and optimization techniques, developed in various fields, to the timetabling problem probably will continue to go on in the future.

Many of the techniques mentioned in this paper have to be investigated more deeply in order to produce results even better that the ones they give at present.

In addition, other techniques may be explored. For example, GSAT (Selman et al. 1992) is a recently-proposed local search algorithm for satisfiability (SAT) problems. It has been applied also to graph coloring and other problems, suitably transformed into SAT problems. It might be interesting to try to formulate the school timetabling problem directly as a SAT problem, and apply the GSAT algorithm to it.

7.2. *Standardization*

Each school or university has its specific rules and constraints for the timetable. This is especially true for universities. Therefore, a program written for creating the timetable of a university can hardly be used for a different one.

Nevertheless, it would be useful to have a standard problem that includes a superset of the constraints of a set of universities so as to write portable programs.

To this aim, the University of Nottingham has distributed a questionnaire (Weare 1995) to all British Universities about how they timetable exams and the problems they have.

7.3. *Approximability*

The quality of the solution of a timetabling algorithm has been always measured only in comparison with other solutions, either produced manually or with other techniques. No guarantee of the quality of the solution with respect to the optimal one has ever been provided in the literature.

Conversely, the theoretical investigation of other problems (e.g., graph coloring and job shop scheduling) has provided a large number of approximation (and non-approximability) results.

It would be interesting to provide some theoretical approximation results for the timetabling problem, at least for the basic optimization problems.

7.4. *Design of a powerful constraint language*

The constraints of a timetabling problem can be very different in nature. Some of them can be easily formulated in a mathematical form, whereas some others are well expressed in some logical formulation.

In addition, in some cases, the informal definition of the constraint, given in some papers, is not clear enough for the reader.

For the above reasons, we envision the definition of a constraint language, semantically well-founded, that can express all the types of constraints usually considered in the literature.

Examples of constraint languages are given in (Cooper and Kingston 1993; Azevedo and Barahona 1994; Solotorevsky et al. 1994; Yoshikawa et al. 1996).

7.5. *Compare and combine different approaches*

The comparison of different approaches to the solution of a specific problem can give an insight on the quality of a specific method. Some comparisons are given in (Corne et al. 1994a; Dowsland 1990; Colorni et al. 1992).

The comparison can also provide information about which approach works best in different situations. In addition, it can also help for the development of combined methods that hopefully exploit the good qualities of the various methods.

Appendix

A. Graph coloring algorithms

The *graph coloring* problem is one of the classical NP-complete problems on graphs (Garey and Johnson 1979, GT4, p. 191): Given an undirected graph $G = (V, E)$, the problem consists of finding a partition of V into a minimum number of *color classes* (or simply *colors*) $c_1, \ldots, c_k$, where no two vertices can be in the same color class if there is an edge between them.

The simplest graph coloring heuristics is the following one (called SEQ in Johnson et al. 1991): Vertices $v_1, \ldots, v_n$, and colors $c_1, \ldots, c_k$ are ordered. Initially, vertex v_1 is assigned to color c_1. Thereafter, vertex v_i in turn is assigned to the "smallest" color that contains no vertices adjacent to v_i.

Such method performs rather poorly in worst-case (see Johnson et al. 1991). Welsh and Powell (1967) propose a variant of the above method in which the vertices are ordered by degree (in decreasing order). That is, the vertices with highest degree are colored first. The underlying idea of this method is that the vertices with high degree are the most difficult to be colored. Other methods based on the same idea have also been proposed. For example, Leighton (1979) adds to the algorithm of Welsh and Powell the idea of recomputing the degree of the vertices at each step, eliminating the vertices already colored. A number of methods based on ordering the vertices by degree is discussed in (Carter 1986).

A slightly different idea is used in the algorithm DSATUR by Brélaz (1979): At each step DSATUR chooses the vertex to color next by picking the one that is adjacent to the largest number of distinctly colored vertices.

Hertz and de Werra (1987) propose the use of tabu search, whereas Chams et al. (1987) and Johnson et al. (1991) make use of simulated annealing. In particular, Johnson et al. (1991) propose three different simulated annealing implementations, and they compare such implementation with many other methods for a class of random graphs.

Coloring of weighted graphs may also be useful for timetabling applications. In fact, the weight of an edge may represent the degree of confliction between two lectures. The problem of coloring of weighted graphs and its application to timetabling are discussed in (Čangalović and Schreuder 1991; Kiaer and Yellen 1992).

B. Tabu search

Tabu search is a *local search* technique designed to solve optimization problems. In this appendix, we briefly describe the technique, and we refer to (Glover 1989; Glover and Laguna 1993) for a comprehensive presentation.

Local search techniques are based on the notion of *neighbor*: Given an optimization problem P, let S be the search space of P, and let f the objective function to minimize (the case of maximization problems is analogous). 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_{init}, which can be obtained with some other technique or generated at random, the algorithm enters in a loop that *navigates* the search space, stepping from one solution to one of its neighbors. The connectivity of the search space, w.r.t. the neighbor relation, is a necessary condition for the technique to work effectively.

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

In order to prevent cycling, there is a so-called *tabu list*, which is the list of solutions to which it is forbidden to move back. It is the list of the last k current solutions, where k is a parameter of the method, and it is run as a queue; that is, when a new solution is added, due to a move, the oldest one is discarded.

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

The procedure stops either when the number of iterations reaches a given value 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 and the cardinality of the set V of neighbor solutions tested at each iteration.

C. Simulated annealing

Simulated annealing is a probabilistic local search technique for finding solutions to optimization problems. It has been proposed by Kirkpatrick et al. (1983) and extensively studied by van Laarhoven and Aarts, Aarts and Korst (1987, 1989). Its name comes from the fact that it simulates the cooling of a collection of hot vibrating atoms.

The process starts by creating a random initial solution. The main procedure consists of a loop that generates *at random* at each iteration a neighbor of the current solution. Like for tabu search, the definition of neighbor depends on the specific structure of the problem.

Let's call Δ the difference in the objective function between the new solution and the current one and suppose to deal with a minimization problem. If $\Delta < 0$ the new solution is *accepted* and becomes the current one. If $\Delta \geq 0$ the new solution is accepted with probability $e^{-\Delta/T}$, where T is a parameter, called the *temperature*.

The temperature T is initially set to an appropriately high value T_0. After a fixed number of iterations, the temperature is decreased by the *cooling rate* a, such that $T_n = a \times T_{n-1}$, where $0 \leq a \leq 1$.

The procedure stops when the temperature reaches a value very closed to 0 and no solution that increases the objective function is accepted anymore, i.e. the system is *frozen*. The solution obtained when the system is frozen is obviously a local minimum.

The control knobs of the procedure are the cooling rate a, the number of iterations at each temperature, and the starting temperature T_0.

A special architecture (both hardware and software) for performing fast programs based on simulated annealing is described in (Abramson 1992).

D. Genetic algorithms

Genetic algorithms are a solution technique for optimization problems (Davis 1991; Michalewicz 1994). Differently from tabu search and simulated annealing they are not based on local search.

A genetic algorithm starts with a set of solutions randomly chosen $\{s_1^0, \ldots, s_n^0\}$, which is called the *population* at time 0.

The core procedure is a loop that creates the population $\{s_1^{t+1}, \ldots, s_n^{t+1}\}$ at time $t+1$ starting from the population at time t. To this aim, the value of the objective function is computed for each solution s_i^t. Based on a weighted randomization, n elements of the population at time t are selected. Obviously, some solution may be selected more than once. The randomization is biased by the value of the objective function so as to assign a higher probability to be

selected to the solutions that result in a better value of the objective function. In this way the best solutions get more copies, and the worse ones probably die off.

At this point each solution is selected for recombination with a given probability (p_R). The recombination is done by the *crossover* operator. That is, two selected solutions are mixed by swapping corresponding segments of their representations. One of the most common ways to do the crossover is by selecting a fixed number of positions in which the swapping takes place (*fixed-point crossover*).

For example, if two solutions are represented by the strings `abcdef` and `uvwxyz` and we choose two crossover points after the second and the fifth character, then the new solutions would be `abwxyf` and `uvcdez`.

In addition, *mutation* arbitrarily alters randomly some part of some solutions randomly selected, based on a given probability value (p_M).

The method terminates either when it generates a fixed number of populations, or when the best solution reaches a certain value of the objective function, or when the algorithm does not make any progress for a certain number of iterations.

The main control parameters of the method are the population size n, the probability of crossover p_R, and the probability of mutation p_M.

Acknowledgments

I wish to thank Luca Cabibbo, Diego Calvanese, Maurizio Lenzerini, Marco Schaerf, and the anonymous referees for their comments on an earlier draft of the paper.

Notes

[1] The URL is `http://tawny.cs.nott.ac.uk/ASAP/ttg/resources.html` and the mail addresses are `ttp-request@Cs.Nott.AC.UK` and `ttp@Cs.Nott.AC.UK`.
[2] We discuss such topics in the appendices because such techniques are used for more than one case and we want Sections 2, 3, and 4 to be independent of each other.
[3] The paper also considers conflicts of rooms. We discuss room assignment in Section 3.

References

Aarts, E. H.L. & Korst, J. (1989). *Simulated Annealing and Boltzmann Machines*. New York: John Wiley & Sons.

Abramson, D. (1991). Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms. *Management Science* **37**(1): 98–113.

Abramson, D. (1992). A Very High Speed Architecture for Simulated Annealing. *IEEE Computer* May: 27–36.

Akkoyunlu, E. A. (1973). A Linear Algorithm for Computing the Optimum University Timetable. *The Computer Journal* **16**(4): 347–350.

Alvarez-Valdes, R., Martin, G. & Tamarit, J. M. (1996). Constructing Good Solutions for the Spanish School Timetabling Problem. *Journal of the Operational Research Society*. To appear.

Aubin, J. & Ferland, J. A. (1989). A Large Scale Timetabling Problem. *Computers and Operational Research* **16**(1): 67–77.

Azevedo, F. & Barahona, P. (January 1994). Timetabling in Constraint Logic Programming. In *Proceedings of World Congress on Expert Systems '94*.

Balakrishnan, N., Lucena, A. & Wong, R. T. (1992). Scheduling Examinations to Reduce Second-Order Conflicts. *Computers and Operational Research* **19**(5): 353–361.

Boufflet, J. P., Benouaghram, R. & Boufflet, G. (1995). An Interactive Computer Aided Design. In *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling*, 324–350.

Brélaz, D. (1979). New Methods to Color Vertices of a Graph. *Communications of the ACM* **22**: 251–256.

Breslaw, J. A. (1976). A Linear Programming Solution to the Faculty Assignment Problem. *Socio-Economic Planning Science* **10**: 227–230.

Burke, E., Elliman, D. & Weare, R. (1993). Extensions to a University Exam Timetabling System. In *IJCAI-93 Workshop on Knowledge-Based Production, Planning, Scheduling and Control*, 42–48. Chambery, France.

Burke, E., Elliman, D. & Weare, R. (1994). A Genetic Algorithm Based University Timetabling System. In *2nd East-West International Conference on Computer Technologies in Education*. Crimea, Ukraine.

Busam, V. A. (1967). An Algorithm for Class Scheduling with Section Preference. *Communications of the ACM* **10**(9): 567–569.

Čangalović, M. & Schreuder, J. A. M. (1991). Exact Coloring Algorithm for Weighted Graph Applied to Timetabling Problems with Lectures of Different Length. *Journal of Operational Research* **51**(2): 248–258.

Carter, M. W. & Tovey, C. A. (1989). When is the Classroom Assignment Problem Hard? *Operations Research* **40**(1S): 28–39.

Carter, M. W., Laporte, G. & Chinneck, J. W. (1994). A General Examination Scheduling System. *Interfaces* **24**(3): 109–120.

Carter, M. W. (1986). A Survey of Practical Applications of Examination Timetabling Algorithms. *Operations Research* **34**(2): 193–202.

Chahal, N. & de Werra, D. (1989). An Interactive System for Constructing Timetables on a PC. *European Journal of Operational Research* **40**: 32–37.

Chams, M., Hertz, A. & de Werra, D. (1987). Some Experiments with Simulated Annealing for Coloring Graphs. *European Journal of Operational Research* 32: 260–266.

Clementson, A. T. & Elphick, C. H. (1982). Continuous Timetabling Problems. *Journal of the Operational Research Society* **33**: 181–183.

Colorni, A., Dorigo, M. & Maniezzo, V. (1992). A Genetic Algorithm to Solve the Timetable Problem. Technical Report 90-060 revised, Politecnico di Milano, Italy.

Cooper, T. B. & Kingston, J. H. (1993). The Solution of Real Instances of the Timetabling Problem. *The Computer Journal* **36**(7): 645–653.

Corne, D., Fang, H.-L. & Mellish, C. (1993). Solving the Modular Exam Scheduling Problem with Genetic Algorithms. Technical Report 622, Department of Artificial Intelligence, University of Edinburgh.

Corne, D., Ross, P. & Fang, H.-L. (1994). Evolutionary Timetabling: Practice, Prospects and Work in Progress. In *UK Planning and Scheduling SIG Workshop*.

Corne, D., Ross, P. & Fang, H.-L. (1994). Fast Practical Evolutionary Timetabling. In *AISB Workshop on Evolutionary Computation*, number 865 in Lecture Notes in Computer Science, 251–263.

Costa, D. (1994). A Tabu Search Algorithm for Computing an Operational Timetable. *European Journal of Operational Research* **76**: 98–110.

Davis, L. (ed.) (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold.

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

Dempster, M. A. H., Lethridge, D. G. & Ulph, A. M. (1973). *School Timetabling by Computer – a Technical History*. Technical report, Oxford University.

Dhar, V. & Ranganathan, N. (1990). Integer Programming vs. Expert Systems: An Experimental Comparison. *Communications of the ACM* **33**(3): 323–336.

Dinkel, J. J., Mote, J. & Venkataramanan, M. A. (1989). An Efficient Decision Support System for Academic Course Scheduling. *Operations Research* **37**(6): 853–864.

Dowsland, K. A. (1990). A Timetabling Problem in Which Clashes Are Inevitable. *Journal of the Operational Research Society* **41**(10): 907–918.

Dyer, J. & Mulvey, J. M. (1976). The Implementation of an Integrated Optimization/Information System for Academic Departmental Planning. *Management Science* **22**: 1332–1341.

ECRC, Germany (December 1995). *ECL^{i}PS^e User Manual (Version 3.5.2)*.

Eglese, R. W. & Rand, G. K. (1987). Conference Seminar Timetabling. *Journal of the Operational Research Society* **38**(7): 591–598.

Eiselt, H. A. & Laporte, G. (1987). Combinatorial Optimization Problems with Soft and Hard Requirements. *Journal of the Operational Research Society* **38**: 785–795.

Even, S., Itai, A. & Shamir, A. (1976). On the Complexity of Timetabling and Multicommodity Flow Problems. *SIAM Journal of Computation* **5**(4): 691–703.

Fahrion, R. & Dollansky. G. (1992). Construction of University Faculty Timetables Using Logic Programming Techniques. *Discrete Applied Mathematics* **35**(3): 221–236.

Feldman, R. & Golumbic, M. C. (1989). Constraint Satisfiability Algorithms for Interactive Student Scheduling. In *Proc. of the 11th Int. Joint Conf. on Artificial Intelligence (IJCAI-89)*, 1010–1016. Morgan Kaufmann.

Ferland, J. A. & Fleurent, C. (1991). Computer Aided Scheduling for a Sport League. *INFOR* **29**: 14–25.

Ferland, J. A. & Roy, S. (1985). Timetabling Problem for University as Assignment of Activity to Resources. *Computers and Operational Research* **12**(2): 207–218.

Ferland, J. A., Roy, S. & Loc, T. G. (1986). The Timetabling Problem. In Coelho, J. D. & Tavares, L. V. (eds.) *O.R. Models on Microcomputers*, 97–103. North-Holland.

Frangouli, H., Harmandas, V. & Stamatopoulos, P. (1995). UTSE: Construction of Optimum Timetables for University Courses – A CLP Based Approach. In *3rd International Conference on the Practical Applications of Prolog (PAP'95)*, 225–243.

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

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

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

Gotlieb, C. C. (1963). The Construction of Class-Teacher Timetables. In Popplewell, C. M. (ed.) *IFIP Congress 62*, 73–77. North-Holland.

Gueret, C., Jussien, N., Boizumault, P. & Prins, C. (1995). Building University Timetables Using Constraint Logic Programming. In *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling*, 393–408.

Halldórsson, M. M. (1993). A Still Better Performance Guarantee for Approximate Graph Coloring. *Information Processing Letters* **45**: 19–23.

Harwood, G. B. & Lawless, R. W. (1975). Optimizing Faculty Teaching Schedules. *Decision Science* **6**: 513–524.

Henz, M. & Würtz, J. (1995). Using Oz for College Time Tabling. In *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling*, 283–296.

Hertz, A. & de Werra, D. (1987). Using Tabu Search Techniques for Graph Coloring. *Computing* **39**: 345–351.

Hertz, A. (1991). Tabu Search for Large Scale Timetabling Problems. *European Journal of Operational Research* **54**: 39–47.

Hertz, A. (1992). Finding a Feasible Course Schedule Using Tabu Search. *Discrete Applied Mathematics* **35**(3): 255–270.

Hilton, A. J. W. (1981). School Timetables. *Annals of Discrete Mathematics* **11**: 177–188.

Hopcroft, J. E. & Karp, R. (1973). An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. *SIAM Journal of Computation* **2**: 225–231.

Ikeda, H., Kitagawa, F. & Nakajima (1995). School Timetabling System: SECTA. In *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling*, 33–44.

Jaffar, J. & Lassez, J.-L. (1987). Constraint Logic Programming. In *Proc. of the 14th ACM POPL Symposium*, Munich, Germany.

Johnson, D. S., Aragon, C. R., McGeoch, L. A. & Schevon, C. (1991). Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. *Operations Research* **39**(3): 378–406.

Johnson, D. (1990). Timetabling University Examinations. *Journal of the Operational Research Society* **41**(1): 39–47.

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

Kang, L. & White, G. M. (1992). A Logic Approach to the Resolution of Constraints in Timetabling. *European Journal of Operational Research* **61**: 306–317.

Kiaer, L. & Yellen, J. (1992). Weighted Graphs and University Course Timetabling. *Computers and Operational Research* **19**(1): 59–67.

Kirkpatrick, S., Gelatt Jr, C. D. & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science* **220**: 671–680.

Klein, D. (1983). The Application of Computerized Solution Techniques to the Problem of Timetabling in High Schools and Universities. Master's thesis, Concordia University.

Klingen, L. H. (1981). Stundenplan-erstellung mit dem computer. *Log In* **1**(4): 31–33.

Laporte, G. & Desroches, S. (1984). Examination Timetabling by Computer. *Computers and Operational Research* **11**(4): 351–360.

Laporte, G. & Desroches, S. (1986). The Problem of Assigning Students to Course Sections in a Large Engineering School. *Computers and Operational Research* **13**: 387–394.

Leighton, F. T. (1979). A Graph Coloring Algorithm for Large Scheduling Problems. *J. Res. Natl. Bur. Standards* **84**: 489–506.

Ling, S.-E. (1992). Integrating Genetic Algorithms with Prolog Assignment Problem as a hybrid Solution for the Politechnic Timetable Problem. In Manner, R. & Manderick, B. (eds.) *Parallel Problem Solving from Nature* **2**, 321–329.

Mathaisel, D. F. X. & Comm, C. L. (1991). Course and Classroom Scheduling – an Interactive Computer-Graphics Approach. *Journal of Systems and Software* **15**(2): 149–157.

McClure, R. H. & Wells, C. E. (1984). A Mathematical Programming Model for Faculty Course Assignment. *Decision Science* **15**: 409–420.

Mehta. M. K. (1981). The Application of a Graph Coloring Method to an Examination Scheduling Problem. *Interfaces* **11**(5): 57–64.

Meisels, A., Gudes, E. & Kuflik, T. (1991). Limited-Resource Time-Tabling by a Generalized Expert System. *Knowledge-Based Systems* **4**: 215–224.

Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag. Second, extended edition.

Minton, S., Johnston, M. D., Philips, A. B. & Laird, P. (1992). Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* **58**: 161–205.

Monfroglio, A. (1986). School Time Table Scheduling in Prolog. *SIGART Newsletter* **96**: 20–22.

Monfroglio, A. (1988). Time-Tabling Through a Deductive Database: A Case Study. *Data and Knowledge Engineering* **3**: 1–27.

Mulvey, J. M. (1982). A Classroom/Time Assignment Model. *European Journal of Operational Research* **9**: 64–70.

Neufeld, G. A. & Tartar, J. (1974). Graph Coloring Conditions for the Existence of Solutions to the Timetable Problem. *Communications of the ACM* **17**(8): 450–453.

Odijk, Michiel A. (1994). *Construction of Periodic Timetables; Part I: A Cutting Plane Algorithm*. Technical Report DUT-TWI-94-61, Delft University of Technology, Department of Technical Mathematics and Informatics, Delft, The Netherlands, 1994.

Ostermann, R. & de Werra, D. (1983). Some Experiments with a Timetabling System. *OR Spektrum* **3**: 199–204.

Paechter, B., Luchian, H., Cumming, A. & Petruic, M. (1994). Two Solutions to the General Timetable Problem Using Evolutionary Methods. In *IEEE Conference on Evolutionary Computation*.

Paechter, B. (1994). Optimising a Presentation Timetable Using Evolutionary Algorithms. In *AISB Workshop on Evolutionary Computation*, number 865 in Lecture Notes in Computer Science, 264–276.

Papadimitriou, C. H. & Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, New Jersey: Prentice-Hall.

Papoulias, D. B. (1980). The Assignment-to-Days Problem in a School Time-Table, a Heuristic Approach. *European Journal of Operational Research* **4**: 31–41.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley.

Petrie, C., Causey, R., Steiner, D. & Dhar, V. (1989). *A Planning Problem: Revisable Academic Course Scheduling*. Technical Report ACT-AI-020, Microelectronics and Computer Technology Corporation.

Sabin, G. C. W. & Winter, G. K. (1986). The Impact of Automated Timetabling on Universities – a Case Study. *Journal of the Operational Research Society* **37**: 689–693.

Schaerf, A. (1996). Scheduling Sport Tournaments Using Constraint Logic Programming. In *Proc. of the 12th European Conf. on Artificial Intelligence (ECAI-96)*, 634–639. Budapest, Hungary: John Wiley & Sons.

Schaerf, A. (1996). Tabu Search Techniques for Large High-School Timetabling Problems. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI-96)*, 363–368. Portland, USA: AAAI Press/MIT Press.

Schmidt, G. & Strohlein, T. (1979). Timetable Construction – an Annotated Bibliography. *The Computer Journal* **23**(4): 307–316.

Schreuder, J. A. M. (1992). Combinatorial Aspects of Construction of Competition Dutch Professional Football Leagues. *Discrete Applied Mathematics* **35**: 301–312.

Selim, S. M. (1983). An Algorithm for Producing Course and Lecture Timetables. *Computers & Education* **7**: 101–108.

Selim, S. M. (1988). Split Vertices in Vertex Colouring and Their Application in Developing a Solution to the Faculty Timetable Problem. *The Computer Journal* **31**(1): 76–82.

Selman, B., Levesque, H. & Mitchell, D. (1992). A New Method for Solving Hard Satisfiability Problems. In *Proc. of the 10th Nat. Conf. on Artificial Intelligence (AAAI-92)*, 440–446.

Shin, W. & Sullivan, J. A. (1977). Dynamic Course Scheduling for College Faculty Via Zero-One Programming. *Decision Science* **8**: 711–721.

Shmoys, D. B., Stein, C. & Wein, J. (1991). Improved Approximation Algorithms for Shop Scheduling Problems. In *Proc. of the Symposium on Discrete Algorithms*.

Smith, G. & Sefton, I. M. (1974). On Lion's Counter-Example for Gotlieb's Method for the Construction of School Timetables. *Communications of the ACM* **17**: 196–197.

Smolka, G. (1995). The Oz Programming Model. In van Leeuwen, J. (ed.) *Current Trend in Computer Science*, number 1000 in Lecture Notes in Computer Science, 324–343. Springer-Verlag.

Solotorevsky, G., Gudes, E. & Meisels, A. (1994). RAPS: A Rule-Based Language Specifying Resource Allocation and Time-Tabling Problems. *IEEE Transactions on Knowledge and Data Engineering* **6**(5): 681–697.

Tripathy, A. (1984). School Timetabling – a Case in Large Binary Integer Linear Programming. *Management Science* **30**(12): 1473–1489.

Tripathy, A. (1992). Computerised Decision Aid for Timetabling – a Case Analysis. *Discrete Applied Mathematics* **35**(3): 313–323.

Van Hentenryck, P. (1991). The CLP Language CHIP: Constraint Solving and Applications. In *COMPCON-91*, San Francisco, CA.

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

Vincke, P. (1984). Timetabling in Belgium: A Survey. In *TIMS XXVI*. Copenhagen.

Weare, R. (1995). Unpublished Manuscript. Nottingham University, UK.

Welsh, D. J. A. & Powell, M. B. (1967). An Upper Bound to the Chromatic Number of a Graph and Its Application to Timetabling Problems. *The Computer Journal* **10**: 85–86.

Wong, K. H. & Ng, W. Y. (1990). An Interactive Timetabling Support System. In *Int. Conf. in System Management*, 307–313, Hong Kong.

Yoshikawa, M., Kaneko, K., Yamanouchi, T. & Watanabe, M. (1996). A Constraint-Based High School Scheduling System. *IEEE Expert* **11**(1): 63–72.