Computational Optimization and Applications 9, 275–298 (1998)

Metaheuristics for High School Timetabling

ALBERTO COLORNI colorni@elet.polimi.it
Centro di Teoria dei Sistemi del CNR, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy

MARCO DORIGO mdorigo@ulb.ac.be
IRIDIA, Université Libre de Bruxelles, CP 194/6, Avenue Franklin Roosevelt 50, 1050 Bruxelles, Belgium, European Union
http://iridia.ulb.ac.be/dorigo/dorigo.html

VITTORIO MANIEZZO maniezzo@csr.unibo.it
Scienze dell'Informazione, Università di Bologna, Contrada Sacchi, 3, 47023 Cesena, Italy
http://www.csr.unibo.it/∼maniezzo

Received ; Revised ; Accepted

Abstract. In this paper we present the results of an investigation of the possibilities offered by three well-known metaheuristic algorithms to solve the timetable problem, a multi-constrained, NP-hard, combinatorial optimization problem with real-world applications. First, we present our model of the problem, including the definition of a hierarchical structure for the objective function, and of the neighborhood search operators which we apply to matrices representing timetables. Then we report about the outcomes of the utilization of the implemented systems to the specific case of the generation of a school timetable. We compare the results obtained by simulated annealing, tabu search and two versions, with and without local search, of the genetic algorithm. Our results show that GA with local search and tabu search based on temporary problem relaxations both outperform simulated annealing and handmade timetables.

Keywords: timetable problem, tabu search, simulated annealing, genetic algorithms

1. Introduction

Metaheuristic algorithms [23] constitute a class of computational paradigms useful for function optimization, often inspired by the study of natural processes. They usually update possible solutions, one or a whole set at a time, to find the optimal solution of a given problem; in this sense the naming *evolutionary algorithm* is common in the literature. Particularly efficient instantiation of evolutionary algorithms are represented by *simulated annealing* [21], in which the natural analogy is the annealing process for metals, by *genetic algorithms* (GA) [19], in which the natural analogy is population genetics, and by *tabu search* [15, 16]. Several other algorithms of this class have already been presented, but the three mentioned ones are those that have obtained most attention by researchers and which have been successfully applied to a wider variety of optimization problems.

The main goal of our work is to compare these metaheuristic on hard, real-world combinatorial optimization problem instances and thus to evaluate their relative merits.

As a test problem we have chosen the timetable problem (TTP), that is known to be NP-hard [12], but which has been intensively investigated given its great practical relevance [1–5, 9, 10, 13, 14, 17, 18, 20, 28].

We will use the construction of a timetable, or schedule of classes, for an Italian high school as a benchmark for our investigation. The ideas introduced in this paper can be applied, of course, to the solution of other, and possibly very different, instances of the timetable problem. The possibility of on-site testing has been the main reason for the choice of this particular problem example. Italian high school timetables must conform to the following directives. In a typical Italian high school, a class receives five hours of lessons, six days a week for the five years of the high school curriculum. Teachers may teach one or more subjects, usually in two or more classes. In addition to their eighteen-hour teaching a week, they have other activities, as described in the paper. Also, every teacher has the right to take one and only one day-off per week, in addition to Sundays.

The construction of the lesson timetable for an Italian high school may be decomposed in the formulation of several interrelated timetables. This permits to cut down the size of the problem instances to be solved to dimensions that can be effectively managed by the algorithms that we tested. In fact, sections are always paired, with a pair of sections sharing many teachers and resources (e.g., laboratories). Two paired sections can therefore be processed as an "atomic unit" consisting of 10 classes, not further decomposable given its high internal dependencies, and relatively isolated from other sections.

Given these premises, the problem is described by:

- a list of m teachers (20–24 in our case);
- a list of p classes involved (10 for the two paired sections);
- a list of n weekly teaching hours for each class (30);
- the *curriculum* of each class, that is the list of the frequencies of the teachers working in the class;
- some external conditions (for example, the hours during which some teachers are involved in other sections or activities).

Notice that the values in parenthesis are typical of a pair of sections and are presented only as an example of the size of the instances we shall be dealing with. No constraint on problem dimension is intended here.

A formal representation of the TTP is the following. Given the 5-tuple $\langle \mathbf{T, A, H, R,}\ f \rangle$ where $\mathbf{T}$ is a finite set $\{T_1, T_2, \ldots, T_l, \ldots, T_m\}$ of m resources (teachers), $\mathbf{A}$ is a set of jobs (teaching in the p classes and other activities) to be accomplished by the teachers, $\mathbf{H}$ is a finite set $\{H_1, H_2, \ldots, H_j, \ldots, H_n\}$ of n time-intervals (hours), $\mathbf{R}$ is a $m \times n$ matrix of $r_{ij} \in \mathbf{A}$ (a timetable) and f is a function to be minimized, $f : \mathbf{R} \Rightarrow \mathbb{R}$; we want to compute

$$\min\ f(\sigma, \Delta, \Omega, \Pi)$$

where σ is the number of *infeasibilities*, as defined in the following, Δ is the set of didactic costs (e.g., having the hours of the same subject clustered in a few days of the week), Ω is a set of organizational costs (e.g., having no teacher available for possible temporary

teaching posts), and Π is a set of personal costs (e.g., having the day-off in an undesired day of the week).

Every solution (timetable) generated by our algorithm is *feasible* if it satisfies the following constraints:

- every teacher and every class must be present in the timetable in a predefined number of hours;
- there may not be more than one teacher in the same class in the same hour;
- no teacher can be in two classes in the same hour;
- there can be no "uncovered hours" (that is, hours when no teacher has been assigned to a class).

Previous research on this problem concentrated on heuristic approaches [10]. In fact, if it were approached with standard algorithms, i.e., defining binary variables x_{ijk} (where, according to the parameters previously specified, i identifies a teacher, j identifies a time-interval and k identifies a class) the problem would be represented by 6000 variables ($i = 1, \ldots, 20$; $j = 1, \ldots, 30$; $k = 1, \ldots, 10$), hundreds of constraints and a very computationally intensive objective function, which makes it implausible to solve to optimality in limited CPU time [2, 8, 22]. We have decided to approach it by means of evolutionary algorithms, which besides generating feasible timetables, try to optimize the objective function introduced in the next section.

The paper is structured as follows. In Section 2 we introduce the encoding we used for solution representation and the local search procedure, which forms the basis of all heuristics we tested. Section 3 describes in detail the objective function we minimize. Section 4 introduces the algorithms we have implemented and Section 5 contains the computational results. Finally, we draw some conclusions in Section 6.

2. Metaheuristics and local search for the timetable problem

We now describe how we approached the problem of generating a school timetable for a pair of sections of an Italian high school, as described in Section 1, by using three different metaheuristic approaches. We first define an encoding of the solutions that allows the application of an effective local search procedure, which is a fundamental component both of tabu search and simulated annealing, and which substantially improves the performance of the genetic algorithm.

Solution encoding

The alphabet we chose is the set **A** of the jobs that teachers have to perform: its elements are the classes to be covered and other activities. We indicate:

- with the characters 1, 2, 3, . . . , 0 the ten classes where the lessons have to be taught;
- with the character D the hours at disposal for temporary teaching posts;
- with the character P the hours for the professional development;

Teacher-Subject	Mon	Tue	Wed	Thu	Fri	Sat
Literature - 1	1100.	0010.	000..	00...	000D1	-----
Literature - 2	..1D1	555D5	.1511	55...	-----	5511.
Literature - 3	6656.	66D6.	666..	-----	65D..	..D55
Literature - 4	-----	44.D4	744..	7747.	77...	7D4D7
Literature - 5	...33	..322	-----	2D2..	232D3	22D33
Literature - 6	.D889	.9D88	9D899	-----	...8D	..899
English	-----	..441	2333.	...22	.1100	40D41
German	-----	776D9	.9955	66788	.....	6958.
History and Philosophy - 1	D4242	332..	4D2..	-----	3443.	342..
History and Philosophy - 2	877..	8D.9.	-----	88D99	..979	.7978
Math and Physics - 1	..324	-----	.2D44	.4334	..342	.3322
Math and Physics - 2	99.97	..877	-----	998D7	898..	887..
Math - 1	-----	1101.	...66	1D660	1D6..	06D00
Math - 2	55S5.	DSS..	5SS..	DSSS5	DSS..	-----
Natural sciences	4261.	9273.	38.87	.2913	.8764	-----
Art	30.78	.8956	.7102	43.41	.2596	-----
Experimental Physics	OSSS.	SSSS0	1SSS0	-----	5SS55	11666
Gymnastic - 1	234D.	-----	..D23	.100.	4DD1.	.....
Gymnastic - 2	789D.	-----	..D78	.D556	96D..	.....
Religion	6	-----	.57..	3.1..	...28	9.0.4

Figure 1. Example of a matrix representing a timetable.

- with the character S the hours during which lessons are taught in classes of sections different from the two considered; these hours are fixed in the initialization phase by hand or by previous runs of our algorithm and are called *fixed hours*;
- with the character " . " the hours in which the teacher does not have to work;
- with the characters "-----" the teacher's day-off.

Our alphabet is therefore $\mathbf{A} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0, D, P, S, ., -\}$.

This alphabet allows us to represent the problem as a matrix $\mathbf{R}$ (an $m \times n$ matrix of $r_{ij} \in \mathbf{A}$) where each row corresponds to a teacher and each column to an hour. Every element r_{ij} of the matrix $\mathbf{R}$ is a variable, whose value may vary on the subset of $\mathbf{A}$ specific to the teacher corresponding to the row containing the gene.

The problem is therefore represented by matrices similar to that proposed in figure 1 (The same representation has also been used in [30]). To be a feasible timetable a matrix must satisfy the constraints discussed in Section 1.

During search timetables are manipulated, possibly resulting in infeasible instances. However, special attention has been put in constraint satisfaction. The constraints are managed as follows.

- by the *neighborhood definition*, so that the set of hours to be taught by each teacher, allocated in the initialization phase, cannot be changed by the application of the operators that identify the neighbors of a current solution;
- by the *filtering algorithm*, so that the infeasibilities caused by the application of the operators are, totally or partially, eliminated by filtering;
- by the *objective function*, so that selective pressure is used to penalize solutions with infeasibilities (infeasibilities are explicitly considered in the objective function, by means of high penalties).

It is possible to distinguish between two kinds of constraints: rows and columns. *Row constraints* (related to the schedule of each single teacher) are incorporated in the operators

in such a way that they are always satisfied; *column constraints* (infeasibilities due to superimpositions or uncovered classes) are managed by means of a combination of fitness function and filtering. Single-teacher solutions (i.e., solutions which satisfy a single teacher) constrain each other by column constraints. Filtering must convert infeasible timetables into feasible ones, modifying them as little as possible.

We decided to manage the infeasibilities by means of both filtering and fitness function penalties because in this way the algorithm has a greater degree of freedom in moving through the search space. This choice is due to the difficulty of the problem: in our application, in fact, every teacher represents a TSP-like problem, consisting of the analysis of the permutations of a predefined symbol set.

Local search

Local search is defined by means of a neighborhood exploring operator. The application of this operator moves a solution to its local optimum. It works in two stages.

In the first stage we try to eliminate infeasibilities without worsening didactic and organizational costs. This is done using a procedure taken from the filtering algorithm—see Appendix—that identifies the causes of the infeasibilities and removes them by swaps of hours.

The second stage is a 2-opt [8] that swaps hours and days until no better solutions are present in the neighborhood of the current timetable. The operator thus moves a solution to a point of the search space that is locally optimal with respect to a neighborhood defined by every possible swap of hours and days.

Simulated annealing and tabu search are designed as modifications of this second stage of local search, except for the improvements detailed in Section 5.

3. The hierarchical structure of the objective function

The objective function (*o.f.*) for our problem measures a generalized cost which represents the gap between the timetable considered and an *ideal* timetable, that is, a timetable which satisfies all the different constraints and didactic, organizational and personal requirements. Being in most cases the different objectives in partial contradiction among themselves, the cost of feasible instances is always greater than 0. The *o.f.* for a timetable matrix $\mathbf{R}$ is

$$z(\mathbf{R}) = \alpha \cdot \Im + \beta_1 \cdot s_\Delta + \beta_2 \cdot s_\Omega + \beta_3 \cdot s_\Pi$$

where

- $\Im$ is the number of infeasibilities in the matrix $\mathbf{R}$;
- s_Δ measures the rate of dissatisfaction of didactic requirements in the matrix $\mathbf{R}$; it is computed as $s_\Delta = \sum_{i=1,4} \Delta_i \cdot d_i$, where the Δ_i are weights associated to each didactic requirement, as detailed in the following, and d_i is the number of times the ith didactic requirement is unmet in the matrix $\mathbf{R}$;
- s_Ω measures the rate of dissatisfaction of organizational requirements in the matrix $\mathbf{R}$; it is computed as $s_\Omega = \sum_{i=1,3} \Omega_i \cdot o_i$, where the Ω_i are weights associated to each

didactic requirement, as detailed in the following, and o_i is the number of times the ith organizational requirement is unmet in the matrix **R**;

- s_Π measures the rate of dissatisfaction of personal requirements in the matrix **R**; it is computed as $s_\Pi = \sum_{i \in T} \gamma_i \cdot t_i$, where $t_i, i = 1, \ldots, m$, measures the dissatisfaction of teacher i in the matrix **R** because of unmet personal desires and γ_i is a weight assigned to each teacher;
- $\alpha, \beta_1, \beta_2, \beta_3$ are weights chosen by the user to bias the timetable towards different aspects of the *o.f.*

By choosing $\alpha \gg \beta_1 \approx \beta_2 \approx \beta_3$, we induce a *hierarchical structure* in the *o.f.*, so that we acknowledge the different relevance of the several groups of problem objectives. In our application, the following structure has been chosen:

level 1, feasibility conditions σ;
level 2, management conditions (Δ, Ω);
level 3, single teachers conditions (Π).

At *level 1* we handle possible superimpositions of teachers (two or more teachers during the same hour in the same class) and "uncovered hours" for the classes (hours when a class is not covered by any teacher).

At *level 2* we consider the following requirement typologies.

Δ—didactic requirements

- no more than 4 teaching hours a day for each teacher (associated weight: Δ_1);
- not the same teacher every day at the last hour (associated weight: Δ_2);
- uniform distribution of the hours of the same subject over the week (associated weight: Δ_3);
- pairs of hours for classworks, for the teachers who require them (associated weight: Δ_4);

Ω—organizational requirements

- concentration on the same day (as much as possible) for parent-teacher meeting hours (associated weight: Ω_1);
- no less than 2 teaching hours a day for each teacher (associated weight: Ω_2);
- as few holes in a teacher's schedule as possible (associated weight: Ω_3).

The different Δ_i and Ω_i are user-defined parameters, used to discriminate the relative importance of the different didactic or organizational requirements.

At *level 3* we consider the preferences expressed by each teacher for his/her specific timetable. Each teacher assesses his/her personal requirements, such as the day-off desired or not having the first or the last hours, and so on. These assessments are then normalized, so that each teacher takes part with a specific quota to the determination of the requirements of the whole teaching staff. The number of unmet personal requirements multiplied by the normalized weight associated to each requirement constitutes the terms t_i in the *o.f.*

Moreover, a teacher ranking (based on criteria such as seniority, external engagements, etc.) is induced by weights γ_i.

4. The algorithms

To solve the problem, we tested three alternatives, implementing a simulated annealing, a tabu search and a genetic algorithm. All three algorithms work on the same problem representation, use the same objective function and local search described in Sections 2 and 3. Simulated annealing and tabu search have already been presented in the timetable literature; specifically tabu search has already been reported to be very effective on timetable problems [17, 18]. As our implementation differs very slightly from previous proposals, we simply sketch these two well-known algorithms and concentrate on the genetic algorithm, which is the most innovative of the three.

Simulated annealing

Simulated annealing (SA) is an effective single-solution randomized heuristic, based on an algorithm originally presented in [24] and proposed as a combinatorial optimization tool in [21]. An example of its application to TTP is presented in Abramson [1]. SA updates a single solution at each iteration, accepting in probability also modifications that involve a worsening of the objective function. The probability of accepting a worsening solution is a function of an internal state variable, called temperature.

The algorithm uses a function that transforms the current temperature level into a lower one; this is known as annealing. The annealing schedule that we have used is $T(t+1) = \alpha T(t)$ where $\alpha (0 < \alpha < 1)$ is a user-defined parameter, called *cooling rate*. We implemented SA essentially as described in [1], except for the use of the objective function and of the solution neighborhood described in the previous section, therefore we refer the interested reader to [1] for a detailed description of the algorithm.

Tabu search

Tabu search (TS) is another evolutionary heuristic that updates a single solution. It was originally proposed in [15, 16]; specific applications to TTP are presented by Hertz and de Werra [10, 17, 18].

The idea behind TS is to start from a random solution and successively move it to one of its current neighbors. Each time a *move* is performed, the reverse one is linked at the beginning of a fixed-length list of inhibited moves, the *tabu list.* From a given solution, not all neighbors can usually be reached. A new candidate move, in fact, brings the solution to its best neighbor, but if the move is present in the tabu list, it is accepted only if it decreases the objective function value below the minimal level so far achieved (*aspiration level*).

The algorithm we implemented includes a variable-sized tabu list: a minimal and a maximal length are specified and during the search, after a constant number of iterations, the actual length is randomly changed. Again, we do not detail our state-of-the-art implementation of TS: the interested reader is referred to Glover [15, 16] or Taillard [31].

Genetic algorithm

The GA [19] is a population-based evolutionary heuristic, where every possible solution is represented by a specific encoding, often called an *individual.* Usually the GA is initialized by a set of randomly generated feasible solutions (a *population*), but problem-specific initializations are possible. Then, individuals are randomly mated allowing the recombination of part of their encoding. The resulting individuals can then be mutated with a specific mutation probability. The new population so obtained undergoes a process of selection which probabilistically removes the worse solutions and provides the basis for a new evolutionary cycle. The fitness of the individuals is made explicit by means of a function, called the *fitness function* (*f.f.*), which is related to the objective function to optimize. The *f.f.* quantifies how good a solution is for the problem faced. In GAs individuals are sometimes also called *chromosomes*, and the positions in the chromosome (i.e., the decision variables of the mathematical formulation of the problem) are called *genes*. The value a gene actually takes is called an allele (or *allelic value*). Allelic values may vary on a predefined set, that is called *allelic alphabet*.

Let P be a population of N chromosomes (*individuals* of P). Let $P(0)$ be the initial population, randomly generated, and $P(t)$ the population at time t. The GA generates a new population $P(t+1)$ from the old population $P(t)$ applying some *genetic* operators. The three basic genetic operators are:

— *reproduction*, an operator which allocates in the population $P(t+1)$ an increasing number of copies of the individuals with a *f.f.* above the average in population $P(t)$;
— *crossover*, a genetic operator activated with a probability p_c, independent of the specific individuals on which it is applied; it takes as input two randomly chosen individuals (parents) and combines them to generate two offspring;
— *mutation*, an operator that causes, with probability p_m, the change of an allelic value of a randomly chosen gene; for instance, if the alphabet were {0, 1}, an allelic value of 0 would be modified into 1 and vice versa.

Some difficulties are encountered when applying GAs to constrained combinatorial optimization problems. The most relevant of them is that crossover and mutation operators may generate infeasible solutions.

The following corrections to this drawback have been proposed in the GA literature.

- change the representation of a solution in such a way that crossover can be applied consistently;
- define new crossover and mutation operators [25] which generate only feasible solutions;
- apply the crossover and mutation operators and then make some kind of *genetic repair* that changes the infeasible solutions to feasible ones through the use of a filtering algorithm.

In the traveling salesman case the most successful approaches have been the introduction of a new crossover operator and the application of genetic repair [26, 27]. The redefinition of

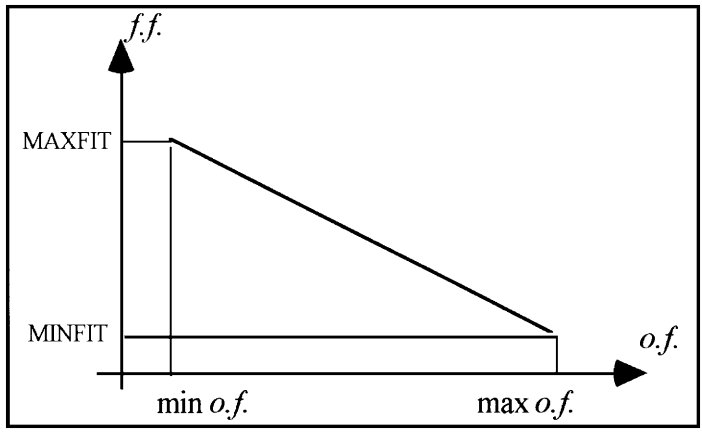

Figure 2. Mapping the *o.f.* into the *f.f.*

mutation is in this case particularly straightforward: it is sufficient to exchange the position of two cities in the string. In the TTP on the other hand, even after the redefinition of both crossover and mutation, it has been necessary to implement genetic repair (filtering).

The objective function is the basis for the computation of the *f.f.*, which provides the GA with feedback from the environment. The feedback is used to direct the population towards areas of the search space characterized by better solutions.

The need to distinguish between objective function (*o.f.*) and fitness function (*f.f.*) comes from the necessity to define the *o.f.* with reference to a cost minimization problem, while the standard GA as introduced by Holland [19] structurally solves maximization problems. The translation from *o.f.* into *f.f.* can be obtained by simply mapping the numeric values of the former into those of the latter, by means of a monotonically decreasing function. In particular, our system is based on a *linear dynamic fitness scaling* procedure. At each generation the maximum and the minimum objective function values of the individuals of the population are computed (max *o.f.* and min *o.f.* in figure 2): they define an interval on the *o.f.* axis which is linearly mapped onto an interval of the *f.f.* axis, limited by two system constants MINFIT and MAXFIT, in such a way that min *o.f.* corresponds to MAXFIT and max *o.f.* corresponds to MINFIT. This procedure attains two objectives: first, it minimizes the *o.f.* while it maximizes the *f.f.*; second, it discriminates solutions belonging to populations whose individuals have very small variations of *o.f.* values, which is often the case in the late stages of the search process when max *o.f.* and min *o.f.* are very close to each other.

The algorithm we used, in Pascal-like notation, is the following:

initialization {this routine creates a population of N random individuals, satisfying for every individual a set of constraints:
— every teacher (row) is given the right amount of hours to be taught.
— some hours are set to the “fixed hour status”, which means they cannot be moved.}

```
while (NOT_VERIFIED_END_TEST) do {the end test is on the number of iterations performed}
 begin
  apply reproduction;
  apply crossover;
  for l:= 1 to N do
   begin
    apply mutation of order k;
    apply day mutation;
    if (LOCAL_ON) then apply local search {LOCAL_ON is a Boolean control variable}
    if (num_infeasibilities > MAX_INFEASIBILITIES) then apply filter
                          {MAX_INFEASIBILITIES is a system constant}
   end;
end.
```

We examine now the operators used by our GA and their computational complexity. This complexity is a function of:

- the number N of individuals composing the population,
- the activation probabilities chosen for each genetic operator,
- the computational complexity of the *f.f.*, of the local *f.f.* (defined below), and of the genetic repair (filter) algorithm.

We call **FF** the fitness function evaluation complexity, and **GR** the genetic repair (filter) complexity (see Appendix).

We have chosen to explicitly represent the activation probabilities in the complexity formulae because they are user-defined variables which can be set to zero in specific situations, thus heavily affecting the complexity of the operator they refer to.

Reproduction. This is the classical reproduction operator that promotes individuals with an above average value of the *f.f.* It gives every individual h a reproduction probability $p_r(\mathrm{h})$ equal to its fitness divided by the total fitness of the population. New populations are generated by using these reproduction probabilities in conjunction with Monte Carlo methods.

The complexity of one application of the reproduction operator to the whole population is then $O(\mathbf{FF} \cdot N)$, where **FF** is the complexity of computing the fitness function.

Crossover. The task of this operator is that of efficiently recombining building blocks (defined below for our case), so that, given two parents, it is possible to generate two offspring with better *f.f.* values (or at least with one of them with a significantly better *f.f.* value). We call the *local fitness function* (*l.f.f.*) that part of the fitness function due only to characteristics specific to each teacher. Given two individuals (timetables) of the population, $\mathbf{R}_1$ and $\mathbf{R}_2$, the rows of $\mathbf{R}_1$ are sorted in order of decreasing *l.f.f.*, and the best k_1 rows are taken as a building block. Then, the remaining $m - k_1$ (where m is the number of teachers) rows are taken from $\mathbf{R}_2$ to generate the first son. The second son is obtained

from the non-utilized rows of $\mathbf{R}_1$ and $\mathbf{R}_2$. The value of k_1 is determined by the program on the basis of the *l.f.f.* of both parents. This operator is applied in probability to each selected pair of potential parents: the probability of its application is the system parameter p_c.

The crossover operator is implemented by means of the following algorithm:

pair randomly the individuals of the population
for each pair of individuals and with probability p_c **do** {p_c is a control parameter}
begin

compute the *l.f.f.* of the rows of the two individuals;
sort by decreasing values of the *l.f.f.* the rows of the two individuals;
create two sons merging twice the two individuals

{the first son is generated taking the best k_1 rows from the better parent and the remaining rows from the worse parent; the second son is generated using the remaining unused rows from both parents};

end;

The complexity of one application of the crossover operator is $O[(N \cdot p_c) \cdot (m \cdot \mathbf{LFF})]$, where **LFF** is the complexity of computing the *l.f.f.*

Mutation of order k. This operator takes k contiguous genes and swaps them with another k contiguous non-overlapping ones belonging to the same row. Mutation of order one is a special case of this operator. It cannot be applied when, among the genes to be mutated, there are some special characters, like A or S (in fact, these symbols correspond to hours which have been allocated once and for all during the initialization phase). This operator is applied in probability to each row of each individual. The probability of its application is the system parameter p_{mk} (p_{m1} in the case of mutation of order one).

A particular kind of mutation is *day mutation*, which takes one day and swaps it with another one belonging to the same row. The ith day, in a teacher timetable, is a substring containing genes that codify five contiguous hours, from the first to the fifth of a same day. It is a special case of mutation of order $k = 5$ and has been introduced for efficiency reasons (with special reference to day-off allocation). It is controlled by a specific application probability parameter, p_{md}.

The complexity of one application of the mutation of order k operator to the population is $O(N \cdot n \cdot m \cdot p_{mk})$.

Filter. The filter operator takes as input an infeasible solution and returns as output a feasible one. It is used to ensure global feasibility to a timetable and is based on the observation that in each column (hour) of the matrix every class must be present once and only once. If it were present twice or more times, there would be teacher superimpositions. If it were not present, the class would be uncovered. It is based on four procedures of increasing computational cost which are applied sequentially. The first two procedures identify swaps of hours of one teacher that eliminate infeasibilities. The other two procedures modify the schedules of more than one teacher.

The detailed steps comprising the filtering algorithm are reported in the Appendix.

Table 1. Genetic operators and feasibility in the TTP case.

Operator name	Global feasibility	Row feasibility
Reproduction	Maintained	Maintained
Crossover	Not maintained	Maintained
Mutation of order k	Not maintained	Maintained
Day mutation	Not maintained	Maintained
Filter	Recovered	Maintained

Table 1 gives the main properties of the considered genetic operators.

Row feasibility is maintained by all the operators presented, while this is not the case for column feasibility: the goal of filtering will, therefore, be that of recovering column—hence global—feasibility for any given timetable.

Summarizing, the operators have the following computational complexities (when applied to a population of dimension N):

Reproduction	$O(\mathbf{FF} \cdot N)$
Crossover	$O(\mathbf{LFF} \cdot N \cdot m \cdot p_c)$
Mutation of order k	$O(N \cdot m \cdot n \cdot p_{mk})$
Local search	$O(\mathbf{FF} \cdot N \cdot m \cdot n)$
Filter (Genetic Repair)	$O(N \cdot m^6 \cdot n^3)$ {see Appendix}
Fitness function **FF**	$O(m^2 \cdot n^2)$
Local fitness function **LFF**	$O(m \cdot n^2)$

5. Computational results

The program we used for our experiments was written using the C language and run on a IBM PC 486, 33 MHz, 8 Mb RAM. The model and the program were tested by defining the timetable for two high schools in Milan. In one of these schools the timetable was handmade by a group of teachers, in the other one a commercial package (PC-UNTIS 3.0) was used. To set up our application, we cooperated with the teachers who usually define the timetable: this collaboration went from the design (for requirement definition) to the validation phase.

Since we could use any predefined solution or population of solutions as our starting point, we chose the previous year's school timetables and adjusted them to meet the new needs. This gave us a proven basis from which to begin, thereby saving computing time and leading to improved solutions with respect to the handmade ones. The PC 486 environment allowed on-site runs, and the carefully designed menu-based user interface allowed the users both to directly interact with the computer in the input phase and to slightly edit the final solution: all these features, beside increasing the effectiveness of the interaction, greatly helped the acceptance of the package by the users.

When we applied our algorithms to the timetable used in the previous year in the school, we were able to better arrange many lessons, so that both some didactic requirements and

Table 2. Objective function parameters.

$\alpha = 1000$	$\Delta_1 = 2$	$\Omega_1 = 4$
$\beta_1 = 500$	$\Delta_2 = 1$	$\Omega_2 = 1$
$\beta_2 = 100$	$\Delta_3 = 1$	$\Omega_3 = 5$
$\beta_3 = 300$	$\Delta_4 = 6$	$\gamma_i \in [0.5, 2]$

several teachers' preferences were better satisfied. In one problem instance for example, for school year 1992/1993, the total cost of the hand-designed timetable was of 234, while SA has been able to lower it to 164, GA to 91 and TS to 85. Note that we computed for this problem a lower bound of 54 for the cost function, due to the presence of inconsistent single teachers requirements and fixed hours. For example, many teachers chose Saturday as desired day-off, which made it impossible to satisfy all of them with a feasible timetable. Similarly, some teachers had fixed hours that did not meet their personal requirements, which again introduced unavoidable costs.

These data, as all those reported later in this Section, were obtained with the values for the objective function parameters listed in Table 2. The parameter values presented in Tables 3–5 were obtained by fine-tuning each algorithm and the objective function on the 1992/1993 problem instance, letting the algorithms run several times with different settings and discussing with the school staff the results.

Simulated annealing

We implemented SA following the indications presented by Abramson [1], using our objective function evaluator. Given the differences between the specific problems undertaken by Abramson and by us, we had to adapt SA to work with our fitness function. In particular, we defined the neighborhood of a timetable as we did in the GA case. Also, SA was asked to minimize not just the number of infeasibilities, but also didactic, organizational and teachers' requirements.

The use of this algorithm in our example problems led to timetables consistently worse than the handmade ones: it could not even recover the initial cost level when it was let evolve from the previous year timetable. And while SA was very efficient in recovering feasible timetables from infeasible ones, it could not effectively optimize feasible timetables. This was true both in the case of high and of low infeasibility costs.

We therefore modified the basic SA in two ways. First, we tested a SA with *reinitialization* of the temperature. As soon as a timetable got frozen, that is, when the SA was unable to improve the current timetable since no proposed swap was accepted anymore, the temperature was set to its starting level. The annealing process could then start again from the previously frozen timetable. The best timetable we found with this modified algorithm, for the 1992/1993 problem instance, had a cost of 183. A graph of the 10-hours long cost evolution for the corresponding run is presented in figure 3, where the lower line plots the evolution of the best so far solution and the higher one the evolution of the current solution.

Alternatively, we tried to relax the problem when the timetable became frozen. We set the infeasibility costs to 3 for a given (parametric) number of iterations after which the normal

Table 3. SA parameters setting.

Parameter	Value
Cooling rate	0.95
Max cycles without improvement	50
Problem reinitialization	No
Problem relaxation	Yes
No. of swaps with relaxed constraints	5

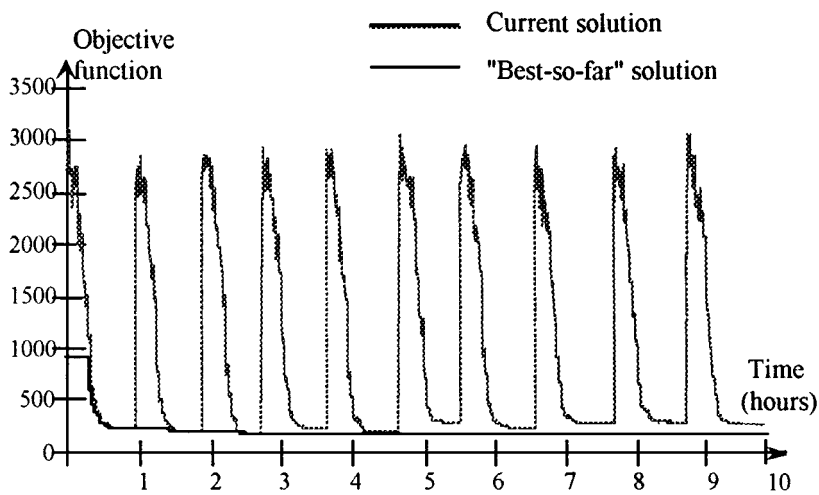

Figure 3. SA with temperature reinitialization.

cost function was applied again. This approach, coupled with a cooling rate of $\alpha = 0.9$, yielded a timetable, for the 1992/1993 problem instance, with a cost of 164. A graph of the cost evolution of the "best-so-far" and current solutions for the corresponding 10-hour long run is presented in figure 4.

The SA parameters were set to the values given in Table 3. The initial temperature level was set to a value T_0 computed as follows. Given a timetable, we generated 1000 alternative random moves and we defined T_0 to be the temperature that allowed a probability of acceptance of 0.5 over the 1000 moves. The terminating condition is based on execution time.

Tabu search

The tabu search algorithm we used is very similar to the algorithm proposed by Hertz [18]. We implemented a variable-length tabu list and used the objective function described in Section 3 to evaluate the cost of the timetables, as in the case of SA. The results achieved

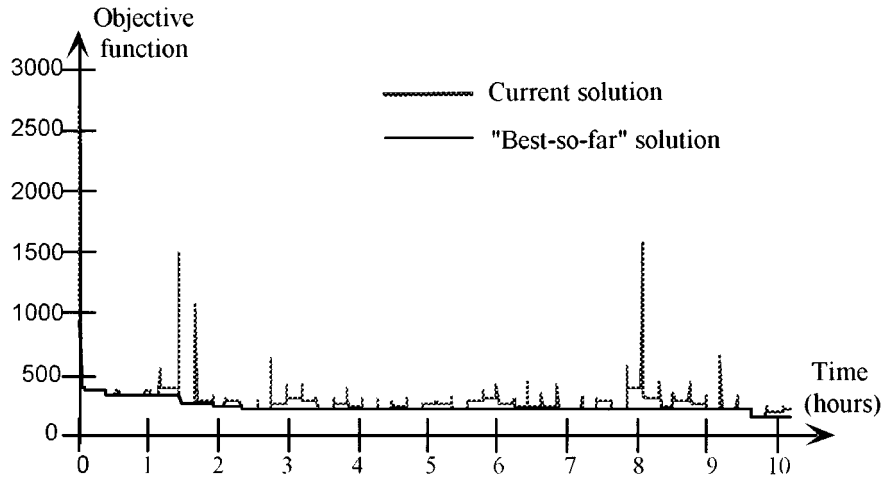

Figure 4. SA with temporary relaxations.

with this simple model however, were not very satisfying, so we implemented a relaxation procedure identical to that used with SA (which is called a 'diversification procedure' in the TS literature). The results so obtained were very good indeed, reliably producing timetables with costs lower than those obtained by both SA and GA.

The complete parameters setting used for TS is reported in Table 4. The procedure used to set them was the same as in the case of SA.

Genetic algorithm

The GA algorithm was detailed in Section 4. Here we report about the experiments carried out to define its parameter values. We conducted a number of experiments to determine an efficient parameter settings for GA. The control parameters tested were:

(i) probabilities: p_{m1}, p_{mk} and p_{md} of the different mutation operators, p_c of crossover;
(ii) local search enabled/disabled;

Table 4. TS parameters setting.

Parameter	Value
Max tabu list length	200
Min tabu list length	30
Max cycles without improvement	30
Problem reinitialization	No
Problem relaxation	Yes
No. of swaps with relaxed constraints	3

Table 5. GA parameters setting.

Parameter	Value
p_{m1}	0.30
p_{mk} $(k = 3)$	0.01
p_{md}	0.02
p_c	0.80
N	15

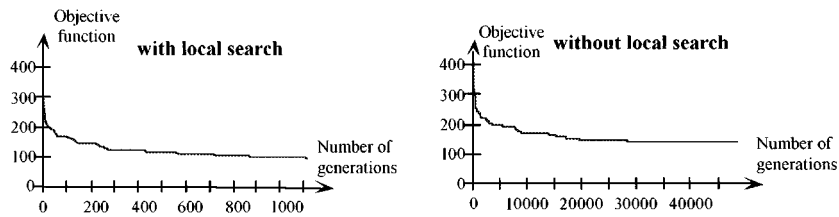

Figure 5. Best values with and without local search.

(iii) cost of infeasibilities: it has been set to a high value (1000, one order of magnitude greater than weights for management conditions) or to a low value (3, to ease explorations of infeasible regions).

Point (i) was tested considering five values for each parameter, the values were chosen at regular intervals over a predefined range. The extreme points of the predefined range was enlarged if the best performing value resulted to be one of the extremes. An initial default setting was arbitrarily defined, then five runs were carried out for each tested value combination, where each tested combination had only one parameter changed from the default setting. Each run lasted eight hours on an IBM PC 486. In this way, we could identify a good parameter setting, whose values are reported in Table 5. The algorithm was found somewhat insensitive when changes of p_{m1} and p_c were held to within ± 0.2. However, this insensitivity was not found for changes of p_{md} (day mutation probability) or p_{mk}, the values of which should be kept very small.

Point (ii) was tested running the algorithm 10 times with local search enabled and 10 times with local search disabled, always using the optimal parameter setting found in point (i). The GA with local search was found to be definitely superior to its counterpart, in accordance with the indications expressed by Mühlenbein [27]. Figure 5 presents the evolution of the best values in the two cases for the 1992/1993 problem instance. Both runs lasted eight hours. The difference in the number of generations is due to the computational cost of local search. As a further test, we applied the local search operator to the best timetables found without local search, thus carrying them to their local optima. This showed that it is definitely worthwhile to search in the space of the local optima.

Finally, tests on point (iii) suggest that a high value for the cost of infeasibilities should be used when local search is off, a low value used when local search is on. In the case

in which local search is on, swaps are applied which maximally decrease the cost of the timetable; that is, the highest-cost, unsatisfied requirements are removed first. When very low infeasibility costs are used, local search tries first to assemble a good timetable from the point of view of second level requirements, and only later it tries to make it feasible. Since we have an explicit filter operator applied after local search, we observe that, on the average, local search coupled with low infeasibility cost is much more effective than local search coupled with high infeasibility costs. In fact, if infeasibilities had a high cost, local search would first try to recover feasibility, even though this process leads to a great increase of second-level costs. Subsequent local optimization, performed on feasible or quasi-feasible timetables, cannot change these second-level costs much. A further benefit of low infeasibility costs results from the fact that in the first iterations more infeasible timetables survive in the population. This broadens the search region which can be reached from the current population, thus allowing the exploration of otherwise unreachable areas.

The case of GA without local search yields opposite results. With low infeasibility costs, filtering is always applied intensively, thus dramatically slowing down the search process. The surviving infeasible solutions, however, are usually the best-rated ones and generate many offspring.

As a final observation, note also that, while exploring promising zones of the search space, the algorithm always identified feasible solutions within very few iterations; very good solutions, however, needed many iterations to be devised.

Complete results

As mentioned in the previous sections, we tested our algorithms by devising complete timetables for two high schools in the Milan area (whose names are "Vittorio Veneto" and "G.B. Vico", but which will be referred for short by S1 and S2, respectively). Each of them had six sections (A to F), which were separately scheduled in pairs. All presented results were obtaining by running each algorithm 10 times over each instance, each time for 8 hours on a PC 486 33 MHz.

First, we present the results for the 1992/1993 problem instance of school S1. Table 6 compares on this instance the results of eight-hour long runs of the different versions of

Table 6. Results for the 1992/1993 problem instance.

Algorithm	Best	Average	Std. dev.
Handmade	234	234	//
GA	138	160	15
GA with local search	91	111	16
TS	184	196	13
TS with problem relaxation	85	97	12
SA	314	341	63
SA with reinitialization	183	262	57
SA with problem relaxation	164	174	25

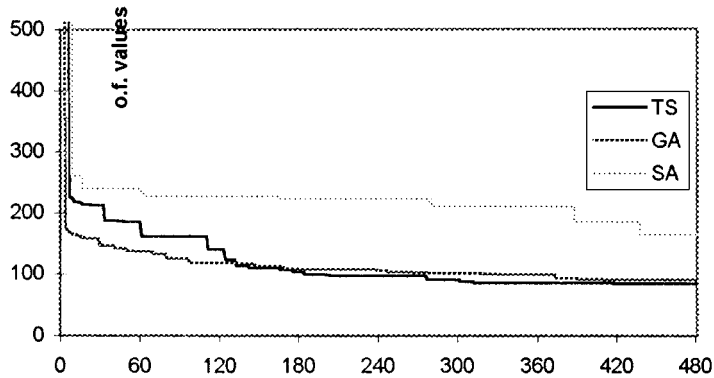

Figure 6. Best runs of the three algorithms.

SA, TS and GA previously introduced with the handmade timetable which was effectively in use in that school year.

As Table 6 shows, all algorithms, in their best version, were able to consistently improve the quality of the handmade solution. The reported results were obtained by using the handmade solution as the starting point for each heuristic, but qualitatively comparable results were obtained also by random initializations. All results in the following Tables were in fact obtained by randomly initializing the solutions.

Figure 6 compares the execution traces of the minute-by-minute best solutions provided by the best performing version of each of the three algorithms. It shows how all algorithms were very rapid in recovering feasibility (we cut out the initial iterations values, which started from over 10000 and would have made unintelligible the differences in the later iterations). Both TS and SA have distinctly stepwise improvements, generally due to the successive relaxations of the original problem. Obviously, the greatest improvements were achieved in the first iterations, but notice how search is effective even in the latest stages.

Table 6 shows that SA was the worst of the three heuristics, while TS was the best, closely followed by GA. This was true both in terms of best result provided and of reliability in providing it. The same conclusions hold true for the tests on 1994/1995 problem instances in schools S1 and S2. Tables 7 and 8 present the results on the instances of school S1. Notice that the construction of the whole school timetable has been reduced to the solution of three different problem instances (each one corresponding to a pair of sections), respectively identified by AB, CD and EF. In Table 7, as in the following ones, for each algorithm we report only the best performing version of each of the three heuristics: GA with local search (GA-LS), SA and TS with problem relaxation (SA-PR, TS-PR), respectively.

Notice that in the EF instance GA performed better than TS, while TS was the best code for the other problems. The averages of Table 7 are significantly worse than those of Table 6 because these problem instances were much more constrained on the use of some common resources (laboratories).

The fixed-hours mechanism makes the execution on one pair of sections dependent on those of previously scheduled pairs of sections. Therefore, in order to compensate the progressively tighter constraints met on subsequent instances to define the whole school

Table 7. Results for the 1994/1995 S1 school problem instances.

Problem instance		GA-LS	SA-PR	TS-PR
Sections AB	Best	195	258	179
	Avg.	226	292	183
	Std. dev.	11	21	6
Sections CD	Best	240	323	207
	Avg.	253	368	225
	Std. dev.	11	19	18
Sections EF	Best	208	304	233
	Avg.	241	367	252
	Std. dev.	15	90	26

Table 8. Results for the 1994/1995 S2 school problem instances.

Problem instance		GA-LS	TS-PR	PC-UNTIS 3.0
Sections AB	Best	144	111	438
	Avg.	171	131	//
	Std. dev.	28	24	//
Sections CD	Best	173	118	503
	Avg.	178	137	//
	Std. dev.	6	18	//
Sections EF	Best	168	111	445
	Avg.	186	129	//
	Std. dev.	12	22	//

timetable, in the 10 runs that were made for each algorithm and for each instance, we scheduled each instance 4 times first, 3 times second and 3 times third. Thus, the AB, CD and EF instances were accordingly permutated. The statistics presented in Tables 7 and 8 therefore consider also the (very limited, as testified by standard deviations) impact of different fixed hours on the same problem instances.

Since SA-PR is obviously a dominated alternative, when we went to school S2 we tested only GA-LS and TS-PR. However, the staff of that school already made a computer-supported timetable generation, so we have been able to compare our final timetables also with those provided by PC-UNTIS 3.0. Table 8 presents the results. Notice that GA-LS and TS-PR were allowed the usual eight-hours long run, while PC-UNTIS 3.0 only took about 2 minutes to define a timetable. However, there was no way to let it run longer, or to make it proceed from a previously saved timetable, and different run always produce the same result.

Again, TS-PR was consistently the best code, GA-LS the second best. Both outperformed by far PC-UNTIS 3.0.

6. Conclusions

In this paper we have presented a computational model and a class of algorithms and computing programs for the timetable problem, with special reference to a real world application (the timetable of an Italian high school). We compared a Genetic Algorithm-based approach with various versions of Simulated Annealing and Tabu Search.

In our experiments Tabu Search has been the best performing algorithm, and the genetic algorithm produced better timetables than simulated annealing.

However, an advantage for end-users acceptance of GAs over both SA and TS is that GAs give the user the flexibility of choosing within a set of different timetables. This is an important feature as the evaluation of a timetable is done by an objective function which can miss some characterizing aspects. This, in turn, can make a timetable with a slightly higher cost more desirable than one with a lower cost, as it often the case in real-world applications.

The main contributions of our work are:

- an empirical comparison of fine-tuned well-known metaheuristics on real-world data regarding high school timetabling;
- the hierarchical structuring (in our case on three levels) of the *o.f.*, in order to allow an easy and effective definition of the relevance of the different objectives used;
- the filtering algorithm, that is an algorithm capable of recovering from infeasibilities, converting an infeasible solution into a feasible one.
- the definition of genetic operators that minimize generalized cost functions (which penalize the possible infeasibilities of the generated solutions) and the distribution over these operators, fitness function and filtering of the management of the infeasibilities;

We believe that this comparison can be of interest to anyone who has to solve timetable problems, even not of the specific kind we dealt with, and that the genetic algorithm we designed is a useful generalization of the GA and can be applied to other highly constrained combinatorial optimization problems.

Appendix: The filtering algorithm

The steps composing the filtering algorithm (which actually are as many filters themselves) are the following.

Step 0: for each column h ($h = 1, \ldots, n$) of the matrix:

- compute the set of classes having superimpositions and put its elements in the list **over**(h) together with their relative matrix coordinates;
- compute the set of uncovered classes and put its elements in the list **miss**(h) together with their relative matrix coordinates.

Step 1: until there exist pairs of classes c_i and c_j such that:

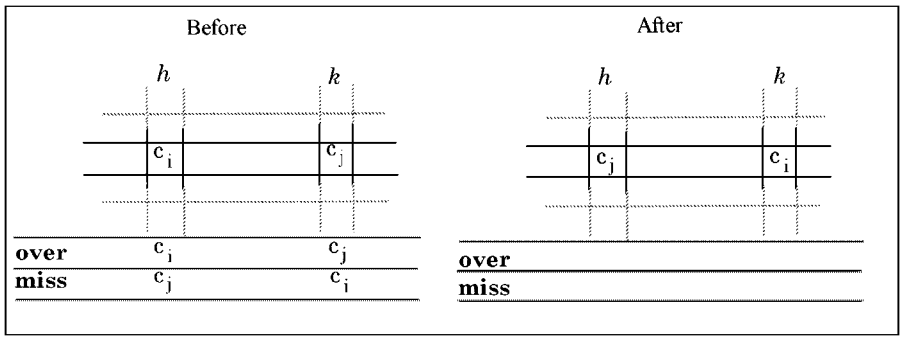

Figure 7. Swapping two classes.

- c_i and c_j are present in the same row (that is, classes c_i and c_j have a common teacher)
- in one of the rows where they are both present, they occupy columns h and k, with $c_i \in \mathbf{over}(h)$, $c_i \in \mathbf{miss}(k)$, $c_j \in \mathbf{over}(k)$, $c_j \in \mathbf{miss}(h)$

swap c_i and c_j.

The effect of the algorithm is shown in figure 7.

Step 2: until there exist pairs of elements, constituted by a class c_i and a *movable gene* e_j (that is, a gene that is either a character D or ♦), such that:

- c_i and e_j are present in the same row
- in one of the rows where they are both present, they occupy columns h and k with $c_i \in \mathbf{over}(h)$, $c_i \in \mathbf{miss}(k)$

swap c_i and e_j.

The effect of the algorithm is shown in figure 8.

Step 3: until there exist *transitive paths among classes*, that is paths connecting classes $c_i, c_j, \ldots, c_s, c_t$, such that

- the classes $c_i, c_j, \ldots, c_s, c_t$ are two by two in the same row
- $c_i \in \mathbf{over}(h)$, $c_i \in \mathbf{miss}(k)$, $c_j \in \mathbf{over}(k)$, $\ldots$, $c_s \in \mathbf{miss}(l)$, $c_t \in \mathbf{over}(l)$, $c_t \in \mathbf{miss}(h)$

swap each class with the following one belonging to the transitive path and present in the same row.

The effect of the algorithm is shown in figure 9, where only a two-step path is considered. Since the complexity of the algorithm grows exponentially with the path length, we set the maximum length path to two. However, the algorithm can be generalized for handling longer paths.

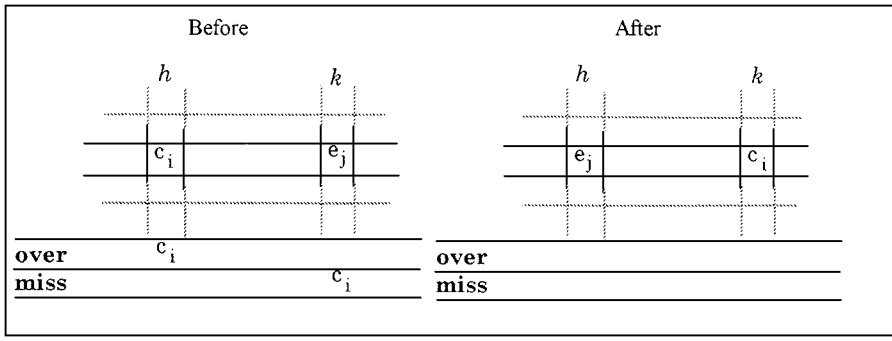

Figure 8. Swapping two elements (a class and a movable gene).

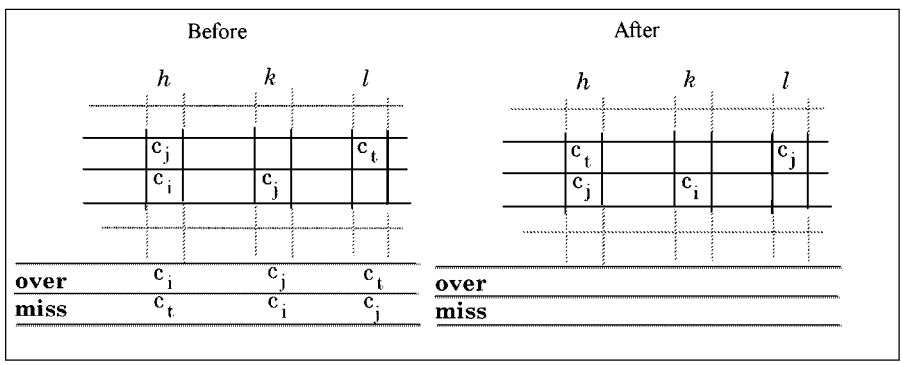

Figure 9. Swapping classes in a transitive path.

Step 4: until there exist *transitive paths among elements*, where an element can be a class or a movable gene in the sense defined in Step 2, behave as in Step 3 (allowing any class to be substituted by a movable gene).

The effect of the algorithm is shown in figure 10.
The computational complexities of the steps are the following. Step 0: $O(m \cdot n)$

Step 1: $O(m^3 \cdot n^2)$
Step 2: $O(m^3 \cdot n^2)$
Step 3: $O(m^6 \cdot n^3)$
Step 4: $O(m^6 \cdot n^3)$

Steps 3 and 4 imply a search in the set of links among superimposed and uncovered classes; it has a complexity that grows exponentially with the length of the path. For this reason we have (parametrically) limited the length of the path to two: the remaining infeasibilities (very few, as was found experimentally) are left to the penalty/reproduction

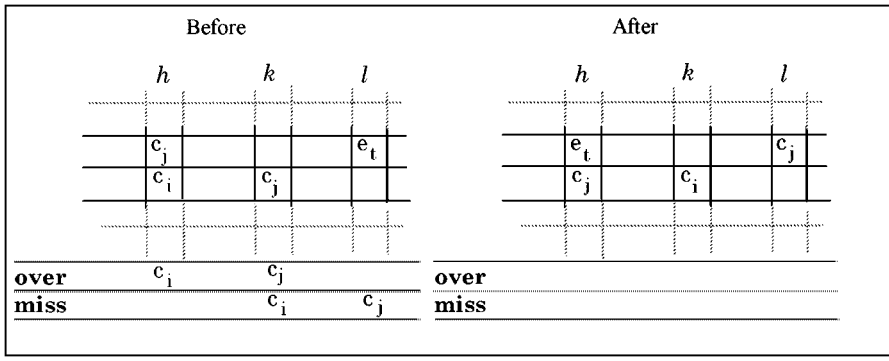

Figure 10. Swapping elements in a transitive path.

mechanism. With such a restriction the complexity of the procedure is polynomial, in our case $m^6 \cdot n^3$.

We propose, as an example, an outline of the computation of the complexity for Step 1. In this case, we have to find all the columns where an element is missing in one and superimposed in the other and vice-versa. Therefore, for each column h (complexity n), for each element $c_i \in \mathbf{over}(h)$ (complexity at most m) and for each element $c_j \in \mathbf{miss}(h)$ (complexity at most m) we have to identify another column k (complexity n) having $c_i \in \mathbf{miss}(k)$, $c_j \in \mathbf{over}(k)$ (complexity $m + m$). The complexity is thus $n \cdot m \cdot m \cdot n \cdot 2 \cdot m = 2 \cdot n^2 \cdot m^3$, that is $O(m^3 \cdot n^2)$.

Observation 1. Note that m is a large overestimate of the length of the over and miss lists, which in the real cases have usually lengths 0, 1 or 2. Therefore, the factor maximally affecting the computational speed of the filter operator is the power of n.

Observation 2. Steps 1 and 2 are special cases respectively of Steps 3 and 4. They have been introduced for efficiency reasons, and could have been obtained from Steps 3 or 4 setting the path length to one.

References

1. A. Abramson, "Constructing school timetables using simulated annealing: Sequential and parallel algorithms," Management Science, vol. 37, pp. 98–113, 1991.
2. E.A. Akkonyunlu, "A linear algorithm for computing the optimum of university timetable," Computer Journal, vol. 16, pp. 347–350, 1973.
3. E.K. Burke, J.P. Newall, and R.F. Weare, "A memetic algorithm for university exam timetabling," in Proc. of ICPTAT'95, 1st Int. Conf. on the Practice and Theory of Automated Timetabling, Napier University, Edinburgh, UK, 1995, pp. 496–503.
4. M.W. Carter, "A survey of practical applications of examination timetabling algorithms," Operations Research, vol. 34, pp. 193–202, 1986.
5. N. Chahal and D. De Werra, "An interactive system for constructing timetables on a PC," European Journal of Operational Research, vol. 40, pp. 32–37, 1989.
6. A. Colorni, M. Dorigo, and V. Maniezzo, Genetic algorithms: A New Approach to the Timetable Problem, NATO ASI Series, Springer-Verlag, 1990, vol. F82, pp. 235–239.

7. D. Costa, “A tabu search algorithm for computing an operational timetable,” European Journal of Operational Research, vol. 57, pp. 98–110, 1994.
8. G.A. Croes, “A method for solving traveling salesman problems,” Operations Research, vol. 6, pp. 791–812, 1958.
9. J. Csima and C.C. Gotleib, “Tests on a computer method for construction of school timetables,” Communications of the ACM, vol. 7, pp. 160–163, 1961.
10. D. de Werra, “An introduction to timetabling,” European Journal of Operational Research, vol. 19, pp. 151–162, 1985.
11. W. Erben and J. Keppler, “A genetic algorithm solving a weekly course-timetabling problem,” in Proc. of ICPTAT’95, 1st Int. Conf. on the Practice and Theory of Automated Timetabling, Napier University, Edinburgh, UK, 1995, pp. 21–32.
12. S. Even, A. Itai, and A. Shamir, “On the complexity of timetable and multicommodity flow problems,” SIAM Journal of Computing, vol. 5, no. 4, pp. 691–703, 1976.
13. J.A. Ferland and S. Roy, “Timetabling problem for university as assignment of activities to resources,” Computers and Operations Research, vol. 12, pp. 207–218, 1985.
14. P. Gianoglio, “Application of neural networks to timetable construction,” in Proc. of the Third Int. Workshop on Neural Networks and Their Applications.
15. F. Glover, “Tabu search—Part I,” ORSA J. on Computing, vol. 1, pp. 190–206, 1989.
16. F. Glover, “Tabu search—Part II,” ORSA J. on Computing, vol. 2, pp. 4–32, 1990.
17. A. Hertz, “Finding a feasible course schedule using tabu search,” Discrete Applied Mathematics, vol. 35, pp. 255–270, 1992.
18. A. Hertz, “Tabu search for large scale timetabling problems,” European Journal of Operational Research, vol. 54, pp. 39–47, 1992.
19. J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press: Ann Arbor, Michigan, 1975. Reprinted by MIT Press, 1992.
20. W. Junginger, “Course scheduling by genetic algorithms,” in Evolutionary Algorithms in Management Applications, J. Biethahn and V. Niessen (Eds.), Springer Verlag, 1995, pp. 357–370.
21. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, “Optimization by simulated annealing,” Science, vol. 220, pp. 671–680, 1983.
22. N.H. Lawrie, “An integer programming model for a school timetabling problem,” Computer Journal, vol. 12, pp. 307–316, 1969.
23. V. Maniezzo, A. Colorni, and M. Dorigo, “Algodesk: An experimental comparison of eight evolutionary heuristics applied to the quadratic assignment problem,” European Journal of Operational Research, vol. 81 no. 1, pp. 188–205, 1995.
24. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” J. of Chemical Physics, vol. 21, pp. 1087–1092, 1953.
25. Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag, 1992.
26. P. Moscato, “An introduction to population approaches for optimization and hierarchical objective functions: Discussion on the role of tabu search,” Annals of Operations Research, vol. 41, pp. 85–121, 1993.
27. H. Muhlenbein, “Parallel genetic algorithms, population genetics and combinatorial optimization, in Proc. of the Third Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1989, pp. 124–129.
28. J.M. Mulvey, “A classroom/time assignment model,” European Journal of Operational Research, vol. 9, pp. 64–70, 1982.
29. D.C. Rich, “A smart genetic algorithm for university timetabling,” in Proc. of ICPTAT’95, 1st Int. Conf. on the Practice and Theory of Automated Timetabling, Napier University, Edinburgh, UK, 1995, pp. 202–216.
30. A. Schaerf and M. Schaerf, “Local search techniques for high school timetabling,” in Proc. of ICPTAT’95, 1st Int. Conf. on the Practice and Theory of Automated Timetabling, Napier University, Edinburgh, UK, 1995.
31. E. Taillard, “Robust taboo search for the quadratic assignment problem,” Report ORPW 90/10, DMA, Swiss Federal Institute of Technology of Lausanne, 1990.