# Tabu Search Techniques for Examination Timetabling

Luca Di Gaspero[1] and Andrea Schaerf[2]

[1] Dipartimento di Matematica e Informatica
Università degli Studi di Udine
via delle Scienze 206, I-33100, Udine, Italy
digasper@dimi.uniud.it
[2] Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica
Università degli Studi di Udine
via delle Scienze 208, I-33100, Udine, Italy
schaerf@uniud.it

**Abstract.** The Examination Timetabling problem regards the scheduling for the exams of a set of university courses, avoiding the overlapping of exams having students in common, fairly spreading the exams for the students, and satisfying room capacity constraints.
We present a family of solution algorithms for a set of variants of the Examination Timetabling problem. The algorithms are based on tabu search, and they import several features from the research on the Graph Colouring problem.
Our algorithms are tested on both public benchmarks and random instances, and they are compared with previous results in the literature. The comparison shows that the presented algorithms performs as well as constructive methods and memetic algorithms, and only a decomposition based approach outperforms them in most cases.

## 1 Introduction

The Examination Timetabling problem is a combinatorial problem that commonly arises in universities: schedule a certain number of examinations in a given number of time slots in such a way that no student is involved in more than one exam at a time. The assignment of exams to days and to time slots within the day is also subject to constraints on availabilities, fair spreading of the students workload, and room capacity. Different variants of the timetabling problem have been proposed in the literature, which differ from each other based on the type of constraints and objectives (see [4], [17] for recent surveys). Constraints involve room capacity and teacher availability, whereas objectives mainly regard student workload.

In this paper, we present an ongoing research on the development of a family of solution algorithms for a set of variants of the Examination Timetabling problem. Our algorithms are based on tabu search [11], and they make use of several features imported from the literature on the Graph Colouring problem [9, Problem GT4, page 191].

E. Burke and W. Erben (Eds.): PATAT 2000, LNCS 2079, pp. 104–117, 2001.

The investigation of different versions of the problem allows us not only to obtain a more flexible application, but also to understand the general structure of the problem family. As a consequence, we should be able to perform a more robust parameter tuning mechanisms and to compare our results with most of previous ones on different versions of the problem.

We tested our algorithms on the popular Toronto benchmarks [6] and on the Nottingham instance, which are the only publicly available ones at present[1] for Examination Timetabling. In this way, we could compare the outcomes of our algorithms with previous results obtained in [1], [2], [6] using different techniques.

This paper is organized as follows. In Section 2 we introduce several formulations of the Examination Timetabling problem, which take into account different combinations of hard and soft constraints. Section 3 contains a presentation of the general local search framework and, in particular, of the tabu search meta-heuristic and of the tandem strategy. The family of algorithms employed in this study are described in detail in Section 4, whereas in the following section we compare their results with those obtained in previously published works. In Section 6 we draw some conclusions and consider some future lines of research that we intend to pursue.

## 2 Examination Timetabling Problem

We introduce the Examination Timetabling problem in stages. In Section 2.1 we present the basic search problem. In Section 2.2 we consider further (hard) constraints, and we describe various components of the objective function (soft constraints). Finally, in Section 2.3 we describe other versions of the problem not considered in this work.

### 2.1 Basic Search Problem

Given is a set of $n$ exams $E = \{e_1, \ldots, e_n\}$, a set of $q$ students $S = \{s_1, \ldots, s_q\}$, and a set of $p$ time slots (or periods) $P = \{1, \ldots, p\}$.

Consecutive time slots lie one unit apart; however, the time distance between periods is not homogeneous due to lunch breaks and nights. This fact is taken into account in second-order constraints as explained below.

There is a binary *enrolment* matrix $C_{n \times q}$, which tells which exams the students plan to attend: i.e. $c_{ij} = 1$ if and only if student $s_j$ wants to attend exam $e_i$.

The basic version of Examination Timetabling is the problem of assigning examinations to time slots avoiding exam overlapping.

The assignment is represented by a binary matrix $Y_{n \times p}$ such that $y_{ik} = 1$ if and only if exam $e_i$ is assigned to period $k$. The corresponding mathematical formulation is the following.

[1] At `ftp://ie.utoronto.ca/pub/carter/testprob` and `ftp://ftp.cs.nott.ac.uk/ttp/Data`, respectively.

$$\text{find} \quad y_{ik} \qquad (i = 1, \ldots, n; k = 1, \ldots, p)$$

$$\text{s.t.} \quad \sum_{k=1}^{p} y_{ik} = 1 \qquad (i = 1, \ldots, n) \tag{1}$$

$$\sum_{h=1}^{q} y_{ik} y_{jk} c_{ih} c_{jh} = 0 \qquad (k = 1, \ldots, p; i, j = 1, \ldots, n; i \neq j) \tag{2}$$

$$y_{ik} = 0 \text{ or } 1 \qquad (i = 1, \ldots, n; k = 1, \ldots, p). \tag{3}$$

Constraints (1) state that an exam must be assigned exactly to one time slot; Constraints (2) state that no student shall attend two exams scheduled at the same time slot.

It is easy to recognize that this basic version of the Examination Timetabling problem is a variant of the well-known NP-complete Graph Colouring problem. In particular, colours represent time slots, each node represents an exam, and there is an (undirected) edge between two nodes $i$ and $j$ if at least one student is enrolled in both exams $e_i$ and $e_j$.

### 2.2 Additional Constraints and Objectives

Many different types of hard and soft constraints have been considered in the literature on Examination Timetabling. The hard ones that we take into account are the following.

**Capacity:** Based on the availability of rooms, we have a *capacity* array $L = (l_1, \ldots, l_p)$, which represents the number of available seats. For each time slot $k$, the value $l_k$ is an upper bound of the total number of students that can be examined at period $k$. The capacity constraints can be expressed as follows:

$$\sum_{i=1}^{n} \sum_{h=1}^{q} c_{ih} y_{ik} \leq l_k \qquad k = 1, \ldots, p). \tag{4}$$

Notice that we do not take into account the number of rooms, but only the total number of seats available in that period. This is reasonable under the assumption that more than one exam can take place in the same room. Alternative formulations that assign one exam per room are discussed in Section 2.3.

**Preassignments and Unavailabilities:** An exam $e_i$ may have to be scheduled necessarily in a given time slot $k$, or, conversely, may not be scheduled in such a time slot. These constraints are added to the formulation by simply imposing $y_{ik}$ to be 1 or 0 respectively.

We now describe the soft constraints, which contribute, with their associated weights, to the objective function to be minimized. For the sake of brevity, we do not provide the mathematical formulation of the objective function.

**Second-Order Conflicts:** A student should not take two exams in consecutive periods. To this aim, we include in the objective function a component that counts the number of times a student has to take a pair of exams scheduled at adjacent periods. Many versions of this constraint type have been considered in the literature, according to the actual time distance between periods:

I) Penalize conflicting exams equally.
II) Penalize *overnight* (last one of the evening and the first of the morning after) adjacent periods less than all others [1].
III) Penalize less the exams just before and just after lunch, do not penalize overnight conflicts [7].

**Higher-Order Conflicts:** This constraint penalizes also the fact that a student takes two exams in periods at distance three, four, or five. Specifically, it assigns a proximity cost $pc(i)$ whenever a student has to attend two exams scheduled within $i$ time slots. The cost of each conflict is thus multiplied by the number of students involved in both examinations. As in [6], we use a cost function that decreases from 16 to 1 as follows: $pc(1) = 16$, $pc(2) = 8$, $pc(3) = 4$, $pc(4) = 2$, $pc(5) = 1$.

**Preferences:** Preferences can be given by teachers and student for scheduling exams to given periods. This is the soft version of preassignments and unavailability.

We have considered many versions of the problem, which differ from each other based on which of the above hard and soft constraints are taken into account.

### 2.3 Other Variants of the Problem

We briefly discuss variants of the problem and different constraint types not considered in our work.

**Room assignment:** Some authors (see, e.g., [5]) allow only one exam per room in a given period. In this case, then exams must be assigned not only to periods, but also to rooms. The assignment must be done based on the number of students taking the exams and capacity of rooms.

**Special rooms:** Some other authors (see, e.g., [14]) consider also different types of rooms, and exams that may only be held in certain types of rooms.

**Exams of variable length:** Exams may have length that do not fit in one single time slot. In this case consecutive time slots must be assigned to them.

**Minimize the length of the session:** We have assumed that the session has a fixed length. However, we may also 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.

**Other higher-order conflicts:** Carter et al. [5] generalize the higher-order constraints and consider a penalty for the fact that a student is forced to take $x$ exams in $y$ consecutive periods.

## 3 Local Search

Local search is a family of general-purpose techniques for search and optimization problems. Local search techniques are *non-exhaustive* in the sense that they do not guarantee to find a feasible (or optimal) solution, but they search non-systematically until a specific stop criterion is satisfied.

### 3.1 Basics

Given an instance $\pi$ of a problem $\Pi$, we associate a *search space* $S$ with it. Each element $s \in S$ corresponds to a potential solution of $\pi$, and is called a *state* of $\pi$. Local search relies on a function $N$, depending on the structure of $\Pi$, which assigns to each $s \in S$ its *neighbourhood* $N(s) \subseteq S$. Each state $s' \in N(s)$ is called a *neighbour* of $s$.

A local search algorithm starts from an initial state $s_0$ (which can be obtained with some other technique or generated randomly) and enters a loop that *navigates* the search space, stepping from one state $s_i$ to one of its neighbours $s_{i+1}$. The neighbourhood is usually composed by the states that are obtained by some local change (called *move*) from the current one.

Local search techniques differ from each other according to the strategy they use to select the move in each state and to stop the search. In all techniques, the search is driven by a *cost function* $f$ that estimates the quality of each state. For optimization problems, $f$ generally accounts for the number of violated constraints and for the objective function of the problem.

The most common local search techniques are *hill climbing*, *simulated annealing*, and *tabu search* (TS). We now describe in more detail TS, which is the technique that we use in our application. However, a full description of TS is beyond the scope of this paper (see, e.g., [11] for a detailed review): we only present the formulation of the technique which has been used in this work.

### 3.2 Tabu Search

At each state $s_i$, TS explores a subset $V$ of the current neighbourhood $N(s_i)$. Among the elements in $V$, the one that gives the minimum value of the cost function becomes the new current state $s_{i+1}$, independently of the fact that $f(s_{i+1})$ is less or greater than $f(s_i)$.

Such a choice allows the algorithm to *escape* from local minima, but creates the risk of cycling among a set of states. In order to prevent cycling, the so-called *tabu list* is used, which determines the forbidden moves. This list stores the most recently accepted moves. The *inverses* of the moves in the list are forbidden.

The simplest way to run the tabu list is as a queue of fixed size $k$. That is, when a new move is added to the list, the oldest one is discarded. We employ a more general mechanism which assigns to each move that enters the list a random number of moves, between two values $k_{\min}$ and $k_{\max}$ (where $k_{\min}$ and $k_{\max}$ are parameters of the method), that it should be kept in the tabu list.

When its tabu period is expired, a move is removed from the list. In this way the size on the list is not fixed, but varies dynamically in the interval $k_{\min}$–$k_{\max}$.

There is also a mechanism, called *aspiration*, that overrides the tabu status: If a move $m$ leads to a state whose cost function value is better than the current best, then its tabu status is dropped and the resulting state is acceptable as the new current one.

The stop criterion is based on the so-called *idle iterations*: The search terminates when it reaches a given number of iterations elapsed from the last improvement of the current best state.

### 3.3 Tandem Search

One of the attractive properties of the local search framework is that different techniques can be combined and alternated to give rise to complex algorithms.

In particular, we explore what we call the *tandem* strategy, which is a simple mechanism for combining two different local search techniques and/or two different neighbourhood relations. Given an initial state $s_0$ and two basic local search techniques $t_1$ and $t_2$, that we call *runners*, the tandem search alternates a run of each $t_i$, always starting from the best solution found by the other one.

The full process stops when it performs a round without an improvement by any of the two runners, whereas the component runners stop according to their specific criteria.

The effectiveness of tandem search has been stressed by several authors (see [11]). In particular, when one of the two runners, say $t_2$, is not used with the aim of improving the cost function, but rather for diversifying the search region, this idea falls under the name of *iterated* local search (see, e.g., [18]). In this case the run with $t_2$ is normally called the *mutation* operator or the *kick* move.

## 4 Tabu Search for Examination Timetabling

We propose TS algorithms for Examination Timetabling, along the lines of the TS algorithm for Graph Colouring proposed by Hertz and de Werra [12].

As already mentioned, Examination Timetabling is an extension of the Graph Colouring problem. In order to represent the additional constraints, we extend the graph with an edge-weight function that represents the number of students involved in two conflicting examinations and a node-weight function that indicates the number of students enrolled in each examination.

### 4.1 Search Space and Cost Function

As in [12], the search space is composed by all complete assignments including the infeasible ones. The only constraints that we impose to be satisfied in all states of the search space are the unavailabilities and preassignments. This can

be easily obtained generating initial solutions that satisfy them, and forbidding moves that lead to states that violate them.

The cost function that guides the search is a hierarchical one, in the sense that it is a linear combination of hard and soft constraints with the weight for hard constraints larger than the sum of all weights of the soft ones. For many cases, though, this simple strategy of assigning fixed weights to the hard and the soft components does not work well. Therefore, during the search the weight $w$ of each component (either hard or soft) is let to vary according to the so-called *shifting penalty* mechanism (see, e.g., [10]):

- If for $K$ consecutive iterations all constraints of that component are satisfied, then $w$ is divided by a factor $\gamma$ randomly chosen between 1.5 and 2.
- If for $H$ consecutive iterations all constraints of that component are satisfied, then the corresponding weight is multiplied by a random factor in the same range.
- Otherwise, the weight is left unchanged.

The values $H$ and $K$ are parameters of the algorithm (and their values are usually between 2 and 20).

This mechanism changes continuously the shape of the cost function in an adaptive way, thus causing TS to visit solutions that have a different structure than the previously visited ones.

### 4.2 Neighbourhood Relation

In our definition, two states are *neighbours* if they differ for the period assigned to a single course. Therefore, a move corresponds to changing the period of one course, and it is identified by a triple $(\langle course \rangle, \langle old\_period \rangle, \langle new\_period \rangle)$.

Regarding the notion of inverse of a move, we experimented with various definitions, and the one that has given the best results is the one that considers inverse of $(e, k_1, k_2)$ any move of the form $(e, \_, \_)$. That is, the period of the course cannot be changed again to any new one.

In order to identify the most promising moves at each iteration, we maintain the so-called *violations list* VL, which contains the exams that are involved in at least one violation (either hard or soft). A second (shorter) list HVL contains only the exams that are involved in violations of hard constraints. In different stages of the search (as explained in Section 4.4), exams are selected either from VL or from HVL. Exams not in the lists are never analysed.

For the selection of the move among the exams in the list (either VL or HVL), we experimented with two different strategies:

**Sampling:** Examine a sample of candidate exams selected based on a dynamic random-variate probability distribution biased on the exams with higher influence in the cost function.

**Exhaustive:** Examine systematically all exams.

In both cases, the selection of the new period for the chosen exam $u$ is exhaustive, and the new period is assigned in such a way that leads to a smallest value of the cost function, arbitrarily tie breaking.

More complex kinds of neighbourhood relations (see [16] for a review) are currently under investigation.

### 4.3 Initial Solution Selection

Many authors (see, e.g., [12,13]) suggest for Graph Colouring to start local search from an initial solution obtained with an *ad hoc* algorithm, rather than from a random state. We experimentally observe that, indeed, giving a good initial state saves significant computational time, which can be usefully exploited for a more complete exploration of the search space. Therefore, we use a greedy algorithm that builds $p$ independent sets (feasible period classes) and assigns all the remaining exams randomly.

### 4.4 Search Techniques

We implemented two main TS-based solvers. The first one is a single runner that uses an adaptive combination of VL and HVL. In detail, it selects from HVL when there are some hard violations, and resorts to VL in any iteration in which HVL is empty.

Our second solver is a tandem that alternates the above-described runner with a second one that always selects the exams from the violations list VL. Intuitively, the first runner focuses on hard constraints, favouring the moves that affect them, whereas the second one searches for any kind of improvement. The former, however, once it has found a feasible solution, automatically expands the neighbourhood to also include moves that deal with soft constraints.

Both runners use the shifting penalty mechanism. In addition, they both use the exhaustive exploration of the violation list, because it "blends" well with the shifting penalty mechanism. In fact, in the presence of a continuous change of the cost function, the use of a more accurate selection of the best move is experimentally proven to be more effective.

## 5 Experimental Results

Our code has been written in C++ using the framework EasyLocal++ [8]. We use also the LEDA libraries [15] that provide a toolbox for data structures and algorithms. Our algorithms have been coded in C++ using the GNU g++ compiler version 2.95.1. The tests reported in the following sections were conducted on an AMD Athlon 650MHz PC running Linux.

### 5.1 Benchmarks and Experimental Setting

Up to now, the real-world data sets available to the timetabling community are the 12 Toronto instances and the single Nottingham instance. We tested our algorithms with these and with random instances. Unavailabilities, preassignments, and preferences are considered only in the random instances, while all other components are present in the benchmarks, too.

Our best results have been obtained by the solver that uses one single runner. The setting of the parameters for each instance has been made after an extensive tuning session, and the best performances are obtained with a tabu tenure varying randomly between 15 and 25. The stop criterion is based on the number of iterations from the last improvement (idle iterations); it depends on the instance and it is set, varying from 2000 to 20 000, so that the duration of the runs is normally kept below 300 s.

In the following sections, we present the results on the benchmark instances and compare them with previous ones obtained using different methods. For the sake of brevity, we omit the results on random instances. Due to the difference in the computing power of the machines involved in the comparison, it seems unfair to compare running times. For this reason, we decided to report only the quality of the solution found.

### 5.2 Comparison with Carter, Laport, and Lee

Carter et al. [6] present some results about the application of a variety of constructive algorithms for all the Toronto instances. The algorithms are based on backtracking and exploit several well-known Graph Colouring heuristics. They consider the formulation of the problem with second-order conflicts (version I) and higher-order conflicts, but no capacity constraints.

The objective function is normalized based on the total number of students. In this way they obtain a measure of the number of violations "per student", which allows them to compare results for instances of different size.

Table 1 summarizes the performances of our first algorithm with respect to Carter's results. The last column shows the best and the worst result among the set of techniques experimented by Carter et al.

The table shows that our results are comparable with Carter's in many cases, though we perform better than all constructive techniques in four cases (boldface).

### 5.3 Comparison with Burke, Newall, and Weare

Burke et al. [2] consider the problem with capacity constraints and second-order conflicts (version I), and they solve it using a memetic algorithm (MA1). The algorithm interleaves evolutionary steps (mutation and cross-over) with a local search operator. This way, the genetic component is able to focus on a restricted search space made up of local optima only.

**Table 1.** Comparison with results of Carter et al. [6]

| Data set | Exams | Time slots | TS solver | | Carter et al. |
|---|---|---|---|---|---|
| | | | Best | Avg | Costs |
| CAR-F-92 | 543 | 32 | **5.2** | 5.6 | 6.2–7.6 |
| CAR-S-91 | 682 | 35 | **6.2** | 6.5 | 7.1–7.9 |
| EAR-F-83 | 189 | 24 | 45.7 | 46.7 | 36.4–46.5 |
| HEC-S-92 | 80 | 18 | 12.4 | 12.6 | 10.8–15.9 |
| KFU-S-93 | 461 | 20 | 18.0 | 19.5 | 14.0–20.8 |
| LSE-F-91 | 381 | 18 | 15.5 | 15.9 | 10.5–13.1 |
| STA-F-83 | 138 | 13 | **160.8** | 166.8 | 161.5–165.7 |
| TRE-S-92 | 261 | 23 | 10.0 | 10.5 | 9.6–11.0 |
| UTA-S-92 | 638 | 35 | 4.2 | 4.5 | 3.5–4.5 |
| UTE-S-92 | 184 | 10 | 29.0 | 31.3 | 25.8–38.3 |
| YOR-F-83 | 180 | 21 | **41.0** | 42.1 | 41.7–49.9 |

**Table 2.** Comparison with results of Burke et al. [2]

| Data set | Exams | Time slots | TS solver | | Burke et al. |
|---|---|---|---|---|---|
| | | | Best cost | Avg cost | MA1 |
| CAR-F-92 | 543 | 40 | 424 | 443 | 331 |
| CAR-S-91 | 543 | 51 | 88 | 98 | 81 |
| KFU-S-93 | 461 | 20 | **512** | 597 | 974 |
| TRE-S-92 | 261 | 35 | 4 | 5 | 3 |
| NOTT | 800 | 26 | **11** | 13 | 53 |
| NOTT | 800 | 23 | **123** | 134 | 269 |
| UTA-S-93 | 638 | 38 | **554** | 625 | 772 |

Table 2 presents the comparison of our TS solver with the best results obtained by Burke *et al.* [2]. The table shows that our results are superior in many cases.

### 5.4 Comparison with Burke and Newall

The results of Burke et al. [2] are refined by Burke and Newall [1], who consider the problem with capacity constraints and second-order conflicts, but in this case with version II. They present results about a subset of the Toronto instances and on the Nottingham one.

They propose a new version of the above memetic algorithm. This version uses a multistage procedure that decomposes the instances into smaller ones and combines the partial assignments. The decomposition is performed along the lines proposed by Carter in [3]. For comparison, they also implement a constructive method.

**Table 3.** Comparison with results of Burke and Newall [1]

| Data set | Exams | Time slots | TS solver | | Burke and Newall | | |
|---|---|---|---|---|---|---|---|
| | | | Best | Avg | MA2 | MA2+D | Con |
| CAR-F-92 | 543 | 36 | 3048 | 3377 | 12167 | 1765 | 2915 |
| KFU-S-93 | 461 | 21 | 1733 | 1845 | 3883 | 1608 | 2700 |
| NOTT | 800 | 23 | 751 | 810 | 1168 | 736 | 918 |
| PUR-S-93 | 2419 | 30 | 123 935 | 126 046 | 219 371 | 65 461 | 97 521 |

Table 3 shows the comparison of their best[2] results with our TS solver. In the table, we call MA2 the memetic algorithm which uses decomposition only at the coarse-grain level, MA2+D the one with a strong use of decomposition (into groups of 50–100 exams), and Con the constructive method.

The table shows that our solver works better than the pure memetic algorithm and the constructive one. Only the approach based on the decomposition performs better.

Decompositions are independent of the technique used, and we are currently working on exploiting such idea also in our TS algorithms. Unfortunately, preliminary experiments do not show any improvement with respect to the results presented in this paper.

### 5.5 Discussion of the Algorithms

To conclude the experiments, we want to show the relative importance of the various features of our TS algorithm. To this aim we compare our regular algorithm with some modified versions that miss one feature at a time.

In detail, we consider the following modifications of our algorithm:

1. Disable the shifting penalty mechanism. Penalties are fixed to their original values throughout the run. Hard constraints are all assigned the same value $\alpha$, which is larger than the sum of all soft ones.
2. Make the selection of the best neighbour always based on the full violation list VL. In the regular algorithm the selection is performed on the sole HVL when hard-conflicts are present.
3. Explore the whole set of examinations at each search step, instead of focusing on the conflicting examination only.
4. Set a fixed value for the tabu list, rather than letting it vary within a given range.
5. Start from a random initial state instead of using the heuristics that searches for $p$ independent sets.

We perform five runs on each version of the algorithm, recording the best, the worst and the average value, and their computing time. Table 4 shows the

[2] The best combination of heuristic and size of decomposition; the results are the average on five runs.

results of such experiments for the instance KFU-S-93 (21 periods and 1955 seats per period). We use the Burke and Newall formulation, and a parameter setting as follows: tabu list length 10–30, idle iterations 10 000.

The results show that the key features of our algorithm are the shifting penalty mechanism and the management of the conflict set. Removing these features, on the average, the quality of the solution degrades more than 60%. In fact, both these features prevent the algorithm from wasting time on large plateaux rather then making worsening moves that diversify the search toward more promising regions.

The intuition that the landscape of the cost function is made up of large plateaux is confirmed from a modified version of the algorithm which explores the whole set of examinations at each step of the search. This algorithm is not even able to find a feasible solution, and uses all the time at its disposal in exploring such regions.

Regarding the selection of the initial state, the loss of starting from a random state is relatively small on regular runs. However, the random initial state sometimes leads to extremely poor results, as shown by the maximum cost obtained. In addition, as previously observed, starting from a good state saves computation time.

The use of a fixed-length tabu list also affects the performance of the algorithm. Furthermore, additional experiments show that the fixed length makes the selection of the single value much more critical (the value of 20 moves used in the reported experiment has been chosen after a long trial-and-error session). Conversely, the variable-length case is more robust with respect to the specific values in use, and gives good results for a large variety of values.

**Table 4.** Relative influence of the features of the algorithm: time in seconds

| | Modified feature | Minimum cost | | Maximum cost | | Average cost | |
|---|---|---|---|---|---|---|---|
| | | cost | time | cost | time | cost | time |
| | None (regular algorithm) | 1793 | 127.65 | 2463 | 189.95 | 2100.4 | 179.47 |
| 1 | Fixed penalties | 3391 | 55.9 | 5827 | 80.95 | 4395.2 | 64.63 |
| 2 | Selection on full VL | 2662 | 110.22 | 4233 | 210.04 | 3507.6 | 3547 |
| 3 | Extended neighbourhood | 23943 | 172.55 | 35105 | 171.59 | 29774.8 | 174.07 |
| 4 | Fixed tabu list length | 2024 | 116.30 | 3333 | 68.95 | 2382.4 | 125.58 |
| 5 | Random initial state | 2142 | 182.62 | 8017 | 64.82 | 3377.8 | 187.24 |

## 6 Conclusions and Future Work

We have implemented different TS-based algorithms for the Examination Timetabling problem and we have compared them with the existing literature on the problem. The most effective algorithm makes use of a shifting penalty

mechanism, a variable-size tabu list, a dynamic neighbourhood selection, and a heuristic initial state. All these features are shown experimentally to be necessary for obtaining good results.

Unfortunately, the experimental analysis shows that the results of our algorithms are not satisfactory on all benchmark instances. Nevertheless, we consider these preliminary results quite encouraging, and in our opinion they provide a good basis for future improvements. To this aim we plan to extend our application in the following ways:

- Use decomposition techniques as Burke and Newall for large instances.
- Implement and possibly interleave other local search techniques, different from TS.
- Implement more complex neighbourhoods relations. In fact, many relations have been proposed inside the Graph Colouring community, which could be profitably adapted for our problem.

The long-term goal of this research is twofold: On the one hand we want to assess the effectiveness of local search techniques for Graph Colouring to Examination Timetabling. On the other hand, we aim at drawing a comprehensive picture of the structure and the hardness of the numerous variants of the Examination Timetabling problem. For this purpose, we are going to consider further versions of the problems, as briefly discussed in Section 2.3.

## References

1. Burke, E., Newall, J.: A Multi-stage Evolutionary Algorithm for the Timetable Problem. IEEE Trans. Evol. Comput. **3** (1999) 63–74
2. Burke, E., Newall, J., Weare R.: A Memetic Algorithm for University Exam Timetabling. In: Proc. 1st Int. Conf. on the Practice and Theory of Automated Timetabling (1995) 241–250
3. Carter, M.W.: A Decomposition Algorithm for Practical Timetabling Problems. Working Paper 83-06. Industrial Engineering, University of Toronto (1983)
4. Carter, M.W., Laporte, G.: Recent Developments in Practical Examination Timetabling. In: Proc. 1st Int. Conf. on the Practice and Theory of Automated Timetabling (1996) 3–21
5. Carter, M.W., Laporte, G., Chinneck, J.W.: A General Examination Scheduling System. Interfaces **24** (1994) 109–120
6. Carter, M.W., Laporte, G., Lee, S.Y.: Examination Timetabling: Algorithmic Strategies and Applications. J. Oper. Res. Soc. **74** (1996) 373–383
7. Corne, D., Fang, H.-L., Mellish, C.: Solving the Modular Exam Scheduling Problem with Genetic Algorithms. Technical Report 622. Department of Artificial Intelligence, University of Edinburgh (1993)
8. Di Gaspero, L., Schaerf, A.: EasyLocal++: An Object-Oriented Framework for Flexible Design of Local Search Algorithms. Technical Report UDMI/13/2000/RR. Dipartimento di Matematica e Informatica, Università di Udine (2000). Available at `http://www.diegm.uniud.it/~aschaerf/projects/local++`
9. Garey, M.R., Johnson, D.S.: Computers and Intractability – A Guide to NP-completeness. Freeman, San Francisco (1979)

10. Gendreau, M., Hertz, A., Laporte, G.: A Tabu Search Heuristic for the Vehicle Routing Problem. Manage. Sci. **40** (1994) 1276–1290
11. Glover, F., Laguna, M.: Tabu Search. Kluwer, Dordrecht (1997)
12. Hertz, A., de Werra, D.: Using Tabu Search Techniques for Graph Coloring. Computing **39** (1987) 345–351
13. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by Simulated Annealing: an Experimental Evaluation. Part II: Graph Coloring and Number Partitioning. Oper. Res. **39** (1991) 378–406
14. Laporte, G. Desroches, S.: Examination Timetabling by Computer. Comput. Oper. Res. **11** (1984) 351–360
15. Mehlhorn, K., Näher, S., Seel, M., Uhrig, C.: The LEDA User Manual. Max Plank Institute, Saarbrücken, Germany (1999). Version 4.0
16. Morgenstern, C., Shapiro, H.: Coloration Neighborhood Structures for General Graph Coloring. In: 1st Ann. ACM–SIAM Symp. on Discrete Algorithms (1990) 226–235
17. Schaerf, A.: A Survey of Automated Timetabling. Artif. Intell. Rev. **13** (1999) 87–127
18. Stützle, T.: Iterated Local Search for the Quadratic Assignment Problem. Technical Report AIDA-99-03. FG Intellektik, TU Darmstadt (1998)