Tabu Search Techniques for Examination Timetabling

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

[1] Dipartimento di Matematica e Informatica
Università di Udine
via delle Scienze 206, I-33100, Udine, Italy
email: digasper@dimi.uniud.it
[2] Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica
Università di Udine
via delle Scienze 208, I-33100, Udine, Italy
email: 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 compared with previous results in the literature.

1 Introduction

The Examination Timetabling problem is a combinatorial problem that commonly arises in universities: One has to 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 student workload, and room capacities.

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 involved (see [4,16] 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 a family of solution algorithms for a set of variants of the Examination Timetabling problem. Our algorithms are based on tabu search [10], and they make use of several features imported from the literature on the Graph Colouring problem [8, Prob. GT4, page 191].

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 perform preliminary experiments of the algorithms on the popular Toronto's benchmarks [6] and on the Nottigham's instance, which are the only publicly available ones at present.[1] We compared our results with previous ones obtained in [2,1,6] using different techniques on such instances. In addition, we experimented with a set of randomly generated instances.

2 The 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 homogenous due to lunch breaks and nights. This fact is taken into account in second-order constraints as explained below.

There is a binary *enrollment* matrix $C_{n \times q}$, which tells which exams the student 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.

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

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

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

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

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

Constraints 1 state that an exam must be taken in exactly one time slot. Constraints 2 state that no student takes two exams scheduled at the same time slot.

It is easy to recognise 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_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.

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

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 can have to be scheduled necessarily in a given time slot k, or, conversely, have to be not scheduled in such time slot. These constraints are added to the formulaton simply imposing y_{ik} to be 1 or 0 respectively.

We now describe the soft constraints, which contribute, with different weights, to the objective function to be minimized. For the sake on 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, based on 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 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, we assign 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], the cost function 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 based on which of the above hard and soft constraint we select.

2.3 Other Variants of the Problem

In this section, we briefly discuss variants of the problem and different constraint types not taken into account in this 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., [13]) consider also different types of rooms, and exams that may only be held in certain types of rooms. In addition, some exams may be split into two or more rooms, in case the students do not fit in one single room.

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

Minimize the length of the session: We have assumed that the session has 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, which has gain popularity in the AI community after the seminal papers by Selman *et al.* [17] and Minton *et al.* [14]. 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 Introduction

Given an instance p of a problem P, we associate a *search space* S with it. Each element $s \in S$ corresponds to a potential solution of p, and is called a *state* of p. Local search relies on a function N, depending on the structure of P, 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 both 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 the 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 details TS which is the technique that we use in our application. However, a full description of TS is out of the scope of this paper (see, e.g., [10]). 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 whether $f(s_{i+1})$ 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 [10]). 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 a TS algorithm for Examination Timetabling, along the lines of the TS algorithm for Graph Colouring proposed by Hertz and de Werra [11].

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 [11], the search space is composed by all complete colourings of the graph, including 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 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 problems, though, this simple strategy of assigning fixed weights to the hard and the soft components doesn't 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., [9]):

– If for K consecutive iterations all constraints of that component are satisfied, then w is divided by a factor γ 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 colouring of a single node. Therefore, a move corresponds to changing the colour of one node, and it is identified by a triple $(\langle node\rangle, \langle old\_colour\rangle, \langle new\_colour\rangle)$. 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 $\langle u, c_1, c_2\rangle$ any move of the form $\langle u, \_, \_\rangle$. That is, the colour of the node 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 nodes that are involved in at least one violation (either hard or soft). A second (shorter) list HVL containts only the nodes that are involved in violations of hard constraints. In different stages of the search (as explained in Section 4.4), nodes are selected either from VL or from HVL. Nodes not in the lists are never analysed.

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

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

In both cases, the selection of the new colour for the selected node u is exhaustive, and the new colour 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 [15] for a review) are currently under investigation.

4.3 Initial Solution Selection

Many authors (see, e.g., [11,12]) 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 useful exploited for a more complete exploration of the search space. Therefore, we use a greedy algorithm that builds k independent sets (feasible colour classes) and assigns all the remaining nodes randomly.

4.4 Search Techniques

We implemented two main TS-based algorithms. The first one is a simple TS that uses the sampling selection rule based on the full violations list VL. No shifting penalty mechanism is used in this case. This algorithm proved to be reasonably fast.

Our second algorithm is a tandem one that alternates two TS runners. Both runners use the shifting penalty mechanism and they make an exhaustive exploration of the neighbourhood. The first runner selects the nodes from the violations list VL, whereas the second one uses an adaptive combination of VL and HVL. In details, it selects form HVL when there are some hard violations, and resorts to VL in any iteration in which HVL is empty.

Intuitively, the first runner searches for any kind of improvement, whereas the second one focusses of hard constraints, taking into account only the moves that affect them. The latter, however, once it has found a feasible, automatically expands the neighbourhood to include also moves that deal with soft constraints.

For both runners, we use the exhaustive exploration of the violation list because it "blends" well with the shifting penalty mechanism. In fact, in 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.

The tandem algorithm is currently ten times slower than the simple TS one.

5 Experimental results

Our algorithms have been coded in C++ using the GNU g++ compiler version 2.8.1. The tests reported in the following sections were conducted on a Sun Ultra Enterprise 450 Server running Solaris 2.6.

5.1 Benchmarks and Experimental Setting

Up to now, the real-world data sets available to the timetabling community are the twelve Toronto instances and the single Nottingham instance. We experimented with most of them and with random instances. Unavailabilities, preassignments, and preferences are considered only in the random instances. All other components are present in the benchmarks, too.

For the first fast solver, the tabu list length is set to 5–10. For the tandem solver, the first runner uses a long tabu list of length 20–40, whereas the second uses a short one of length 5–10. The number of idle iterations is set so that the duration of the runs is between 10 and 200 secs for the fast solver and between 100 and 2000 secs for the latter.

For the sake of brevity, we present only the results on the benchmark instances, and we compare them with previous ones. Our results are preliminary, in the sense that a fine grain parameter tuning phase is still in progress.

Carter *et al.* [6] present some results about the application of a variety of constructive algorithms for all the Toronto instances. They consider 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. This way they obtain a measure of the number of violations "per student", which allows them to compare results for instances of different size.

Data set	**exams**	**time slots**	**TS results**				**Carter's algorithms results**	
			best cost	time	avg cost	avg time	min—max cost	min—max time
EAR-F-83	190	24	47.0	34.8 s	49.4	40.8 s	36.4—46.5	6.0—242.2 s
HEC-S-92	81	18	13.8	11.1 s	15.4	13.4 s	10.8—15.9	6.5—22.8 s
KFU-S-93	461	20	18.3	80.6 s	19.7	57.2 s	14.0—20.8	120.2—208.9 s
LSE-F-91	381	18	14.7	32.5 s	16.2	29.0 s	10.5—13.1	48.0—233.5 s
STA-F-83	139	13	158.3	8.9 s	159.8	10.1 s	161.5—165.7	5.7—6.1 s
TRE-S-92	261	23	10.2	170.1 s	10.8	137.4 s	9.6—11.0	74.1—150.6 s
UTA-S-92	638	35	4.7	929.8 s	5.0	1351.0 s	3.5—4.5	664.3—1287.8 s
UTE-S-92	184	10	28.8	10.2 s	30.6	6.9 s	25.8—38.3	4.7—9.1 s
YOR-F-83	180	21	42.8	53.0 s	46.2	32.0 s	41.7—49.9	174.5—428.0 s

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

Table 1 (next page) summarizes the performances of our first algorithm with respect to Carter's results. The table shows that our results are comparable with Carter's ones in many cases, even though we perform better than all constructive techniques only in one case. The tandem solver hasn't given better results at this stage.

5.2 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 using a memetic algorithm (MA1).

Data set	**n**	**p**	**tandem solver**		**Burke *et al.***
			best	avg.	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. Comparison with results of Burke *et al.* [2]

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

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.

Data set	exams	time slots	tandem 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	2135	2825	3883	1608	2700
NOTT	800	23	751	810	1168	736	918
PUR-S-93	2419	30	123935	126046	219371	65461	97521

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

They propose a new memetic algorithm and a constructive one (Con) for comparison. The memetic algorithm uses a multistage procedure that decomposes the instances in smaller ones and combines the partial assignments. We call MA2 the version with only a coarse grain decomposition and MA2+D the one with a strong use of decomposition (into groups of 50-100 exams). The decomposition is performed by an heuristic method proposed by Carter [3].

Table 3 (next page) shows the comparison of the results with our tandem solver. Our solver works better than the pure memetic algorithm and the constructive one. Only the approach based on the decomposition performs better.

The decomposition technique, however, are usually quite sensible to the problem instance. In addition, they are independent of the technique used, and they could be exploited in our TS algorithms as well.

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.

Our results are not satisfactory in all instances, but we plan to improve our algorithms in several ways:

– engineer the algorithms and optimize the code;
– perform an extensive parameter tuning session for all instances;
– use decomposition techniques as Burke and Newall for large instances.

Nevertheless, we consider these preliminary results quite encouraging, and in our opinion they provide a good basis for future improvements.

Furthermore, as a longer term goal, we plan to extend our application in the following ways:

– 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.

Finally, we are going to consider futher versions of the problems, as briefly discussed in Section 2.3.

References

1. E. Burke and J. Newall. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1):63–74, 1999.
2. E. Burke, J. Newall, and R. Weare. A memetic algorithm for university exam timetabling. In *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling*, pages 241–250, 1995.
3. M. W. Carter. A decomposition algorithm for pratical timetabilng problems. Working Paper 83-06, Industrial Engineering, University of Toronto, April 1983.
4. M. W. Carter and G. Laporte. Recent developments in pratical examination timetabling. In *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling*, pages 3–21, 1996.
5. M. W. Carter, G. Laporte, and J. W. Chinneck. A general examination scheduling system. *Interfaces*, 24(3):109–120, 1994.
6. M. W. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 74:373–383, 1996.
7. D. Corne, H.-L. Fang, and C. Mellish. Solving the modular exam scheduling problem with genetic algorithms. Technical Report 622, Department of Artificial Intelligence, University of Edinburgh, 1993.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability—A guide to NP-completeness.* W.H. Freeman and Company, San Francisco, 1979.
9. M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.
10. Fred Glover and Manuel Laguna. *Tabu search.* Kluwer Academic Publishers, 1997.
11. A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.
12. D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.
13. G. Laporte and S. Desroches. Examination timetabling by computer. *Computers and Operational Research*, 11(4):351–360, 1984.
14. Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proc. of the 8th Nat. Conf. on Artificial Intelligence (AAAI-90)*, pages 17–24. AAAI Press/MIT Press, 1990.
15. Craig Morgenstern and Harry Shapiro. Coloration neighborhood structures for general graph coloring. In *First Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 226–235, 1990.
16. Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.
17. Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the 10th Nat. Conf. on Artificial Intelligence (AAAI-92)*, pages 440–446, 1992.
18. Thomas Stützle. Iterated local search for the quadratic assignment problem. Technical Report AIDA-99-03, FG Intellektik, TU Darmstadt, 1998.