Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 195 (2009) 307–318
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

O.R. Applications

A decomposed metaheuristic approach for a real-world university timetabling problem

Patrick De Causmaecker [b], Peter Demeester [a,*], Greet Vanden Berghe [a]

[a] KaHo Sint-Lieven, Information Technology, Gebroeders Desmetstraat 1, 9000 Gent, Belgium
[b] Katholieke Universiteit Leuven Campus Kortrijk, Computer Science and Information Technology, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium

Received 8 December 2006; accepted 22 January 2008
Available online 7 February 2008

Abstract

In this paper we present a decomposed metaheuristic approach to solve a real-world university course timetabling problem. Essential in this problem are the overlapping time slots and the irregular weekly timetables. A first stage in the approach reduces the number of subjects through the introduction of new structures that we call 'pillars'. The next stages involve a metaheuristic search that attempts to solve the constraints one by one, instead of trying to find a solution for all the constraints at once. Test results for a real-world instance are presented.

Keywords: Metaheuristics; Timetabling; Tabu search; Decomposed heuristic

1. Introduction

Ever since the penetration of the computer in educational environments, people have tried to solve the university timetabling problem automatically (see [7,20] for a historical overview). Academic researchers know this problem very well as many of them are confronted with university timetabling daily. It remains a compelling research area even after many years of research.

The general term university timetabling typically refers to both university course and exam timetabling. As discussed in [39] both problems are different in nature. In examination timetabling one of the goals is to spread, to the best possible extent, the different exams for each individual student, while in course timetabling the students want an as compact timetable as possible. Recurring timetables (weekly, fortnightly, etc.,) are appreciated for course timetabling, while for exam timetabling this is certainly not the case [11]. We will not go into further detail on university exam timetabling, since it is not the topic of this paper.

The university course timetabling problem is the process of assigning lectures, which are covered by lecturers and attended by students, into 'room–time' slots, taking into account hard and soft constraints. Hard constraints cannot be violated, since that would make the solution infeasible. Neither students nor lecturers, for example, can physically be in two different places at the same time. This is what Lewis [37] calls the event-clash constraint. In contrast to hard constraints, violations of soft constraints are possible but if there are none, the quality of the solution is experienced as 'better'. An example of a soft constraint can be to avoid time gaps between lectures on the same day.

The main contribution of this work consists of a multi-stage approach to solve a non-weekly recurring real-world timetabling problem with overlapping time slots. In every stage of the method an extra constraint is added, and the constraints are solved in a progressive manner: in a first stage the local search algorithm tries to find a solution satisfying the first constraint. The solution found in the first

* Corresponding author. Tel.: +32 92658610.
E-mail addresses: Patrick.DeCausmaecker@kuleuven-kortrijk.be (P. De Causmaecker), Peter.Demeester@kahosl.be (P. Demeester), Greet.VandenBerghe@kahosl.be (G. Vanden Berghe).

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2008.01.043

stage is used as the initial solution for the second stage. This process is repeated until there are no constraints left to evaluate.

In Section 2 we review the literature on the university course timetabling problem. We describe the university course timetabling problem as it is perceived at the KaHo Sint-Lieven School of Engineering in Section 3. The timetabling problem is quite different from most problems encountered in literature, since it deals with overlapping time slots and subjects that are not necessarily covered on a weekly basis. We are not aware of any similar problems in literature. We discuss in Sections 4–6 a local search approach to solve the problem. In Section 7 we discuss the obtained solutions and we conclude in Section 8.

2. Related work

In the literature different university course timetabling problems are described in the sense that other (hard and soft) constraints, and various solution approaches are considered. The probably best known and studied university course timetabling problem was the subject of the International Timetable Competition (ITC) of the Metaheuristics Network [33] in 2002. The problem consists of scheduling (weekly recurring) events into 45 (non-overlapping) time slots (that equals: 5 days multiplied by 9 lecture hours per day) and in appropriate class rooms. Several timetablers accepted the challenge and submitted their solutions. The best solutions are based on local search algorithms, such as hill climbing, tabu search, simulated annealing, great deluge and combinations of these algorithms. Di Gaspero and Schaerf [27] combine hill climbing and tabu search, while Cordeau et al. [34] execute a pre-processing step to obtain feasible solutions. With the feasible solution as a starting point, a tabu search algorithm is applied to find better quality solutions (satisfying the soft constraints to the best possible extent). Burke et al. [8] applied the great deluge algorithm to the problem. The authors claim that the advantage of their approach is that only two parameters have to be fine-tuned, namely the search time and an estimate of the quality of the solution. Kostuch, the winner of the contest [36], also first applies a pre-processing step to find a feasible solution, after which the two following stages start. Both are based on a simulated annealing search algorithm. The first algorithm is applied to find better quality solutions that satisfy the two soft constraints involving students, by optimally sequencing the time slots. To improve the overall solution quality, the second simulated annealing algorithm swaps pairs of events between time slots, provided that the swaps do not lead to infeasible solutions.

Other researchers used the instances of the International Timetable Competition as a test bench for their algorithms. Abdullah et al. [2,3] compare a randomized iterative improvement algorithm and a variable neighbourhood local search with the results from the competitors. For small instances the VNS algorithm generates results that are comparable to the best results of the competition, while the randomized iterative improvement algorithm generates for seven out of the eleven datasets equal or better results. Socha et al. [45] describe two ant algorithms, of which the Max–Min Ant System leads to the best solutions for large instances.

Apart from the above described metaheuristic approaches that were used to solve the university course timetabling problem as it is defined by the International Timetable Competition, publications exist on other university timetabling problems with similar or different constraints for which other approaches have been applied, such as:

- Integer programming [24,46].
- Constraint logic programming [21,29,42].
- Reduction of the problem to graph coloring [6,9,25]. The idea is to translate the timetabling problem to a graph, in which the nodes (or vertices) correspond to the lectures and the edges between the nodes correspond to the constraints. The graph colouring problem is the problem of assigning a limited number of colours to the nodes of the graphs in such a way that no two nodes connected by an edge have the same colour. The number of colours corresponds to the number of available time slots.
- Case-based reasoning [12,14,17]. Case-based reasoning (CBR) is an AI method that tries to solve new problems by applying knowledge gained from previous experiences. In contrast to other methods that claim to be problem-independent but contain lots of problem specific information, Burke et al. [14] contend that CBR does not contain this kind of problem-specific information, which makes it suitable for solving different kinds of timetabling problems.
- Hyperheuristics [10,13,15]. A hyperheuristic consist of several low-level heuristics from which it can select one to solve an optimization problem. The advantage of using hyperheuristics is that the method can be applied to a broad range of problems. In [10] the authors apply their method to a nurse rostering and a university course timetabling problem (the same problem as the one reported above).
- Neural networks [18,19]. Carrasco et al. base their work on the findings of Hopfield and Tank [32], who showed that artificial neural networks can be used to solve combinatorial optimization problems. They proved that if the neural network satisfies specific constraints, it converges to stable states which can be shown to correspond to the local optima of a so-called neural energy function. Carrasco et al. formulate such a neural network energy function that contains a weighted sum of the violations of the problem's hard and soft constraints. Minimizing this sum provides neural network states which correspond to feasible or almost feasible solutions of the timetabling problem satisfying most of the soft constraints.

Several literature overviews on course timetabling appeared: [1,7,11,16,41,43]. Lewis [38] discusses several metaheuristic techniques for solving the university course timetabling problem in his review.

3. The KaHo Sint-Lieven course timetabling problem

In this section we describe the university course timetabling problem as it is perceived at the KaHo Sint-Lieven School of Engineering. An academic year consists of 2 semesters of 12 weeks each. A week consists of 5 lecture days. A typical lecture day starts after 8 AM in the morning and finishes before 7 PM in the evening. A day is divided into different time slots, which do not necessarily have the same duration. Lectures, for example, take 90 minutes and the duration of lab sessions ranges from 190 to 290 minutes. The time slots have a fixed begin and end time and cannot be changed during the search for a timetable. Table 1 presents the different time slots of a lecture day. Note that some time slots overlap. Suppose that, for example, a lecture day starts with a lab session (with a duration of 190 minutes) at 8:10 AM. The next lecture for one of the resources involved (student group, lecturer or room) cannot be scheduled before 11:20 AM. The overlapping time slots will introduce specific constraints to the timetabling problem. Several lectures for the same group of students are generally organized either in the morning or in the afternoon, while only one lab session is organized in the morning or afternoon. Possibly, this lab session can be preceded or followed by a theoretical lecture. No lectures or lab sessions can be scheduled between 1 and 2 PM.

The first three semesters are common for all the engineering students. At the start of the fourth semester, students have to choose between different disciplines such as electro mechanics, construction, chemistry and electronics-ICT.

Table 1
Overview of the 19 fixed time slots within a lecture day

Time slots	
Begin	End
08:10	09:10
08:10	09:20
08:10	09:40
08:10	11:20
08:10	12:30
08:10	13:00
09:20	13:00
09:50	11:20
09:50	13:00
11:30	13:00
14:00	15:30
14:00	17:10
14:00	17:40
14:00	18:20
14:00	18:40
15:40	17:10
15:40	17:40
15:40	18:40
17:10	18:40

Students are grouped according to the compulsory courses that they take. A course typically consists of several lectures and lab sessions. Lectures usually occur 12 times during a semester, but there exist lectures that deviate from this. Lab sessions have an occurrence that can range from 2 to 10 times during a semester. Each group of students is further divided into smaller lab session groups, mainly because students need more individual coaching for lab sessions. This leads to a hierarchy among the different student groups (see Fig. 1 for an example of the master's hierarchy). This hierarchy of student groups has to be taken into account during the evaluation process since we will consider student groups and not individual students and this can cause a violation of the above introduced event-clash constraint. The assignment of students to lab groups is given and is not part of the automated timetabling application.

Most lab sessions can only take place in special purpose rooms. This involves an important timetabling constraint.

In contrast to some other institutes [4], is the assignment of the modules to the lecturers not part of the timetabling problem. Lecturers know in advance which lectures and lab sessions they have to cover. In the first three semesters, most lab sessions (e.g. programming, technical drawing, physics or chemistry) are covered by two or more lecturers, depending on the number of students.

Generally speaking, when automating the process of scheduling courses, the following model is considered: L lectures, covered by E lecturers and taken by S student groups have to be scheduled into R rooms and T time slots, taking into account that:

- *C1* A lecturer E_i or a student group S_j cannot be assigned to more than one lecture in the same time slot.
- *C2* A room R_k cannot hold more than one lecture at the same time.
- *C3* Student groups have to fit into the assigned class room.
- *C4* Some lab sessions can only take place in especially equipped rooms.

The above constraints correspond well to the constraints described in general university course timetabling papers [5,22,23,40,44]. However, we are not aware of any problems similar to that of the KaHo Sint-Lieven School of Engineering. It differs from general descriptions in the literature, since it also deals with the following constraints:

- *C5* A resource (room R_k, lecturer E_i or student group S_j) cannot be assigned to more than one lecture in overlapping time slots.
- *C6* Every course needs to be organized as many times as demanded, not less, not more.

A result of constraint *C6* will be that, since not every course is organized every week of the semester, timetables will differ from week to week.

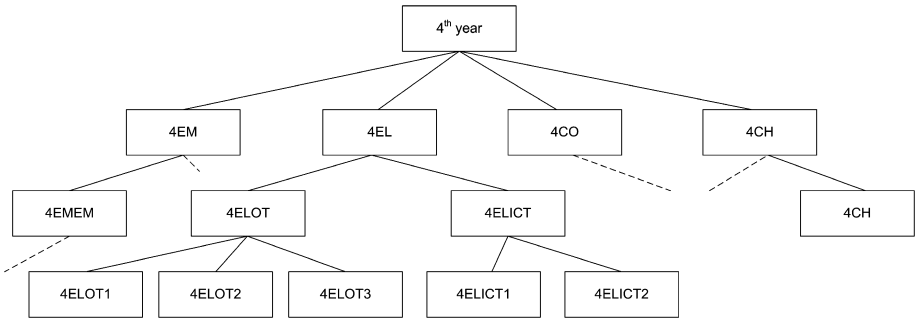

Fig. 1. Graphical representation of the student group hierarchy of the Engineering Master at KaHo Sint-Lieven.

When studying different manually constructed timetables we observed that not all of the above constraints were satisfied. A few lecturers for instance had to cover in the same time slot two different lab sessions in the same room. Upon inquiry it appeared that this was done on purpose. In the beginning of the academic year it happens that constraint *C3* is violated. Sometimes this is solved by moving chairs from another room to the assigned class room. After a few weeks this problem is automatically solved, when some of the students start skipping classes.

The problem size of a semester's timetable is on average:

- 133 lecturer groups.
- 212 student groups, each containing on average 20 students.
- 67 class rooms of different sizes.
- 19 sometimes overlapping, fixed time slots per day. These time slots are defined in such a way that there is no lecturing between 1 and 2 PM.
- 1215 lectures (with a pre-assigned duration) that need to be scheduled. Each course consists of a set of lectures. Some of these courses have a non-weekly recurring frequency.

The above discussed International Timetabling Competition (ITC) released 20 problem instances. The problem size of these instances is:

- between 200 and 350 students,
- 10 or 11 class rooms,
- 9 non-overlapping time slots per day,
- between 350 and 440 lectures that need to be scheduled.

The main difference between the two timetabling problems is that the KaHo Sint-Lieven problem does not consider individual students, but groups of students that are hierarchically structured. The number of individual students of the KaHo Sint-Lieven problem is larger than the number of student groups of the ITC problem. The number of rooms, time slots per day and courses are in the KaHo Sint-Lieven problem larger than these of the second problem. Remark that nothing is known about the lecturers in the ITC case.

In the next section we describe the local search algorithm that we apply to solve the university course timetabling problem.

4. Constructing a local search algorithm for the KaHo Sint-Lieven course timetabling problem

Since our main concern was the automated generation of real-world university course timetables and not the construction of yet another new search algorithm, we opted to employ a traditional local search algorithm. This choice is motivated by the fact that local search algorithms produced the best solutions in the International Timetable Competition of the Metaheuristics Network (see Section 2). We applied tabu search [28] which has been proven to be very successful for a variety of timetabling problems [30,31,34]. The essential ingredients are

- the representation of the solution space,
- the neighbourhoods, which describe the new solutions that can be reached from a solution by a single move,
- the cost function, which is a measure for the timetable quality.

In the next sections we describe the solution space, the neighbourhoods and the cost function in detail.

4.1. Solution space

For the university course timetabling problem described in Section 3, we represent the solution space as a collection of two dimensional arrays (matrices) with the class rooms in the rows and the time slots (over a period of a week) in the columns. Each individual matrix will only contain lectures and lab sessions of equal duration. Thus, there will be as many matrices as there are lectures and lab sessions with different durations. With this representation we will avoid that lectures or lab sessions with a particular duration would end up in a room–time slot combination of which the time slot has a different duration. A graphical representation of the search space is shown in Fig. 2. The *R*s denote the rooms and the *T*s denote the time slots. Note that every individual matrix contains all the rooms, but that each time slot is only present once in all matrices.

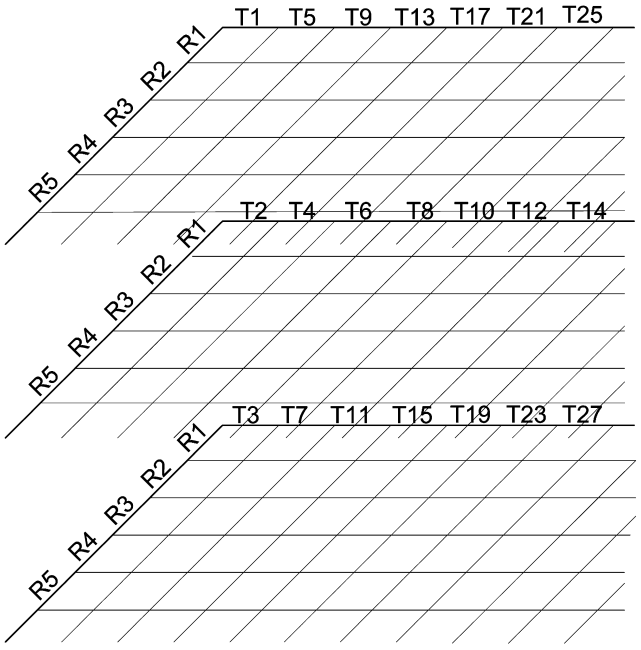

Fig. 2. Graphical representation of the solution space. Each matrix contains subjects of equal length.

Time slots are sorted lexicographically on the (start time, end time) combination. The start and end times of time slots are predefined, and do not change in the optimization process. The matrix that contains the lectures with a duration of 90 minutes contains no overlapping time slots (see also Table 1), the other matrices however can contain overlapping time slots within the same matrix. Apart from overlapping time slots in the same matrix, there are of course overlapping time slots between different matrices. This has to be taken into consideration for lecturers, student groups and rooms. We did not provide any special ordering for the rooms in the matrices.

A lecture or lab session allocated to a time slot–room combination is represented by its ID in the matrix. By choosing this representation, the constraint *C2* is automatically satisfied: no two lectures or lab sessions will ever be allocated to the same time slot–room combination. It does however not avoid that a class room will be occupied with lectures or lab sessions in overlapping time slots (constraint *C5*). A matrix element will contain 0 when no lecture or lab session is assigned to the corresponding time slot–room combination. As an initial solution, we randomly (without taking into account any of the other constraints of Section 3) assign lectures and lab sessions to the room–time slot combinations.

4.2. Neighbourhoods

To explore the solution space we implemented moves that define different neighbourhoods. The *Swap Move* is the basic move. This move swaps the contents of two slots within a two dimensional matrix (see Fig. 3). Restricting the directions for this basic Swap Move guides the exploration of the solution space. This leads to a number of neighbourhood flavours, of which the most important are:

- *Time swap neighbourhood*: In the exploration of this neighbourhood, the basic move is limited to shifting a lecture or lab session within the same row. This neighbourhood originates from swapping a specific lecture or lab session with all elements in the same row of its current matrix.
- The *room swap neighbourhood* explores the moves that shift a randomly chosen lecture or lab session into an appropriate class room in the same column. For lectures, it implies that the size of the class room should be large enough to accommodate the students taking the lecture, while for lab sessions the lecture should take place in a large enough room that is equipped for the course.
- In the *time–room swap neighbourhood* will the randomly chosen lecture or lab session be swapped with every (zero or non-zero) lecture or lab session in the row and column where this lecture or lab session is situated. This neighbourhood is a combination of the first two neighbourhoods.

All the neighbourhoods discussed above only swap lectures or lab sessions within one matrix. Swapping lectures or lab sessions between 2 of the matrices would make no sense, since the time slots have a different duration. We call these neighbourhoods 'horizontal' neighbourhoods, to emphasize the fact that we swap lectures in the same matrix and to contrast with the 'vertical' neighbourhoods we will introduce in Section 5.1.

4.3. Cost function

The cost function evaluates a solution and is determined by the weighted sum of the violations of the constraints. Since constraint *C2* is always satisfied, because the presentation of the solution space avoids any violation of the constraint, it does not need to be evaluated. The constraints we take into account are *C1*, *C3*, *C4* and *C5*. These constraints are modeled as soft constraints and can thus be violated. The *C6* constraint will be discussed in the Section 5. To limit the computation effort, we only partially reevaluate the solution, i.e. in the rows and columns that have changed.

4.4. The local search algorithm and its parameters

We experimented with different parameter settings for the tabu search algorithm: the number of unimproving moves before calling another neighbourhood and a dynamically changing (prime) tabu list length.

- Another neighbourhood is chosen every time the number of unimproving moves exceeds a certain limit. This corresponds to the token-ring search of [26].

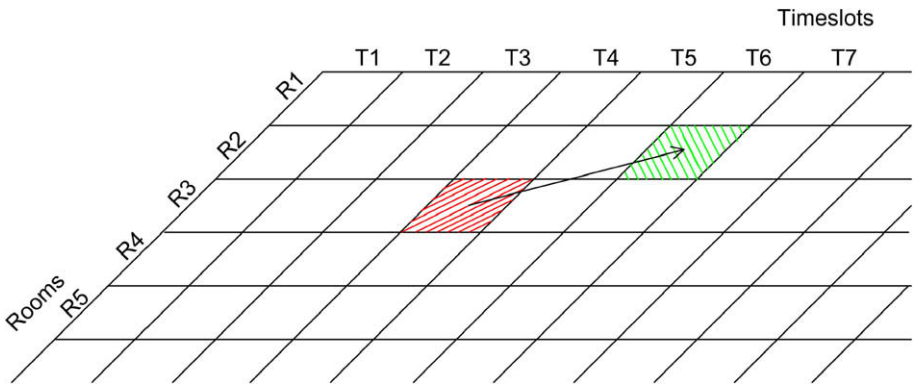

Fig. 3. Graphical representation of a swap move: if no lecture or lab session is assigned to the destination room–time slot combination, the original lecture or lab session will be moved to the empty spot, in the other case, the two sessions will be swapped.

– Before the search starts, an ordered list of primes starting at 7, with upper limit the tabu list length is constructed. The tabu search starts from a tabu list length of 7, and the next value in the prime list is selected if the best solution is not improved, if the best solution however is improved, the tabu list length becomes the preceding prime in the list.

The algorithm is schematically presented as Algorithm 1. We made abstraction of the dynamic tabu list changes in the pseudo code of Algorithm 1.

As explained in Section 4.3 the cost function is the weighted sum of the violations of the constraints. Since we consider all constraints equally important, all the weights of the cost function are assigned the value 1.

Algorithm 1. Local search algorithm

```
CN: threshold parameter to change neighbourhood
Stop condition: maximum number of iterations
reached or costfunction(solution) = 0
best solution = initial solution
while stop condition is not met do
    apply tabu search: new solution = S*
    if costfunction(S*) < costfunction(best solution) then
        best solution = S*
    else
        if number of unimproving iterations > CN then
            change neighbourhood
            set number of improving iterations = 0
        else
            increment(number of unimproving iterations)
        end if
    end if
end while
```

4.5. Real-world issues

The data needed to automate the timetabling process, is available in an administrative university database. This database contains information about the lecturers and the courses that they cover, the duration of the lectures and lab sessions, the student groups and their size, the rooms and their size. Currently no information is available about the room types. To avoid that lab sessions would end up in rooms that, even though they are large enough, are not equipped for the session, we assign rooms to lab sessions beforehand. Each lab session gets an extra constraint enforcing in which lab room it should take place. By working in this manner we probably exclude some lab rooms that are equally equipped, and thus exclude possible better solutions to the problem.

5. Decomposition approach

5.1. From tiles to pillars

The search algorithm generates a weekly recurring schedule with Algorithm 1. For a high school or a university timetabling problem with weekly recurring lectures, the method described in Section 4 will suffice. As explained in Section 3, the KaHo Sint-Lieven School of Engineering deals with subjects that are often not organized more than a few times per semester, which involves 12 lecture weeks. The number can range from every fortnight, the first 6 weeks or the last 6 weeks of a semester, or three times evenly spread during the semester, etc. Since we want to avoid scheduling numerous independent weekly timetables, which would make the timetabling problem larger and hence more complex, we need a different approach to tackle this issue.

Constructing 12 entirely independent weekly timetables is not an option since that would not be appreciated by either students or lecturers, as their timetables would differ too much from week to week. To overcome that discomfort we want the weekly timetables to be as similar as possible for everyone involved. Therefore, we carry out a pre-processing step. The method we apply, is inspired by the tiling approach that Kingston [35] successfully applied to Australian high school timetabling in which similar subjects are grouped and treated as one. The idea in the KaHo Sint-Lieven case is to sort lectures and lab sessions which are covered by the same lecturer. We start from the first

lecturer and select the most frequent covered session (whether a lecture or a lab session) with a frequency less than 12. Dependent on whether the selected session is a lecture or a lab session, the continuation of the applied approach differs.

- If it is a lecture, then the algorithm searches for another lecture from the same lecturer, of which the frequency during the semester is added to the frequency of the first lecture. The lecture that first adds up to 12 is chosen to be grouped with the original lecture. If no such lecture is found, the lecture with a frequency that adds up closest to 12 is selected.
- If, however, the selected session is a lab session, the same procedure as described for a lecture is valid, except that the lab room of the original lab session is taken into account. This means that the lab session that the algorithm searches for, has to meet two conditions: apart from being the lab session with a sufficiently large frequency that adds up to 12 or closest to 12, it should also have the lab room in common with the original lab session. In the more general case in which we would have information about the different room types (see Section 4.5), the grouping would be based on equal room types instead of on common lab rooms.

This greedy procedure is repeated until all lectures and lab sessions of the same lecturer are evaluated. The first pre-processing step ends when there are no lecturers left to evaluate. After the initial grouping of lecturers is performed, a second pre-processing step is carried out. The groups of lectures and lab sessions that were formed in the first pre-processing step and which sum of frequencies is less than 12 are evaluated. Instead of grouping the lectures and lab sessions by the same lecturer, we group all the lab sessions that take place in the same room. These lab sessions are grouped in the same greedy manner as described in the first pre-processing step.

Fig. 4 is a graphical representation of a semester consisting of twelve weeks. The 'pillar' in the middle of the figure is composed of two interleaved lectures (marked with and without a cross) that are each organized six times in the same room during a semester. Grouping the lectures in this way reduces the 1215 lectures to be scheduled to 437 pillars, which reduces the combinatorial complexity of the timetabling problem. In the case described in Fig. 4, the pillars are constructed for lectures covered in the same location and time slot. Students or lecturers are not necessarily the same in one pillar. Note that by representation constraint *C6* is always fulfilled.

Apart from the moves described above, which we call 'horizontal', we also implemented a 'vertical' move that allows swapping lectures and lab sessions within the same pillar (see Fig. 5). This move is useful when one pillar combines lectures or lab sessions that have different student groups. Problems with overlapping time slots and student groups being assigned to two places at the same time can

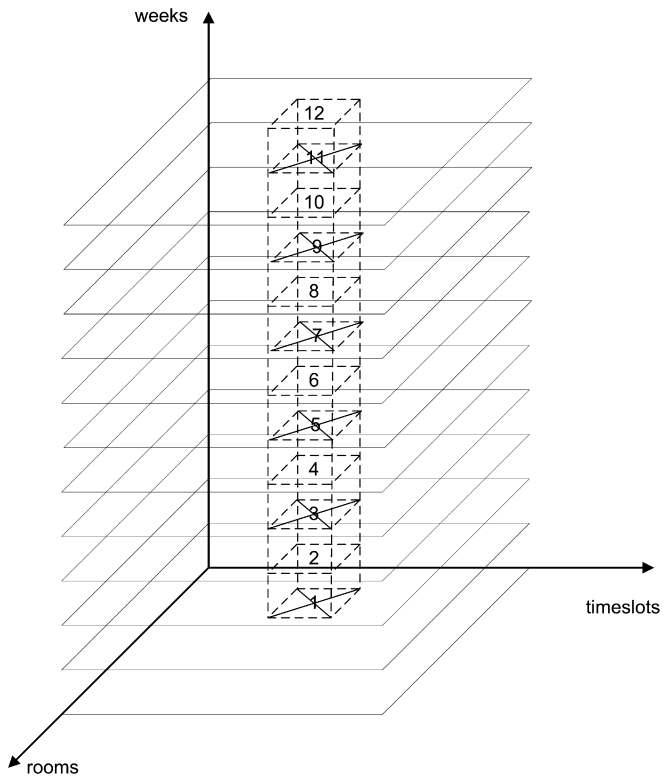

Fig. 4. Graphical representation of pillars. Lectures in the pillar are scheduled on a fortnightly basis.

be solved by shifting lectures or lab sessions in such a pillar. We call this the *Pillar Move*. We explore this 'vertical' solution space with the *Pillar Plane Neighbourhood*. In one iteration we select a random matrix, in which we select a random part of the list of pillars contained in this matrix. In each pillar of this list we try to execute a *Pillar Move* and finally the best *Pillar Move* is selected.

6. Solving the problem in different stages

Instead of trying to find a solution that satisfies all the constraints at once, we opted to work in different stages (see Algorithm 2). In the first stage, a solution that satisfies one constraint is constructed. This corresponds to assigning the value zero to all weights of the cost function (see Section 4.3) except for one of the weights, that is assigned the value 1. The stop criterion is met when either the solution does not violate that constraint anymore or when a maximum number of iterations is exceeded. Then, an extra constraint is added, or formulated differently: another weight (next to the weight that already was assigned the value 1) is assigned the value 1. By taking the best solution of the first stage as the initial solution of the second stage, the tabu search algorithm tries to find a second solution satisfying both constraints to the best possible extent. This process is repeated until no constraints remain to be evaluated, or if all weights of the cost function are assigned the value 1.

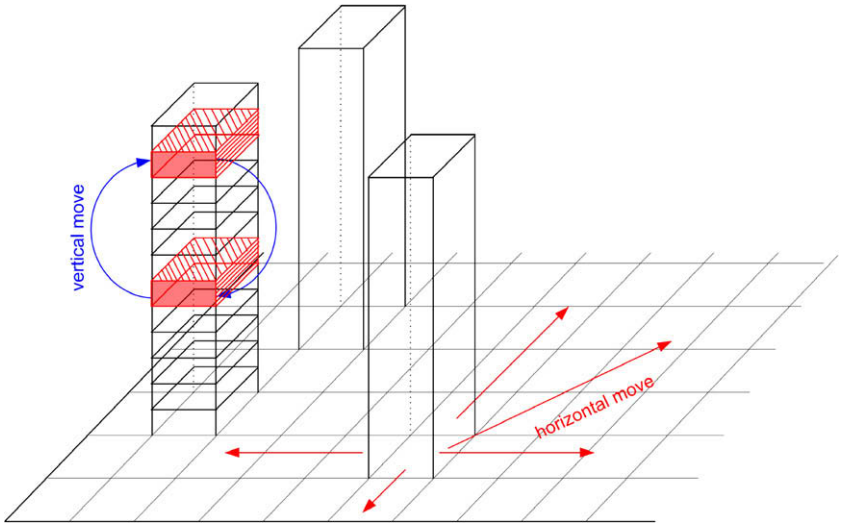

Fig. 5. Graphical representation of the 'vertical' and 'horizontal' moves in a matrix containing only lectures of the same duration.

Algorithm 2. Local search algorithm with sequential evaluation of the constraints

```
CN: threshold parameter to change neighbourhood
Stop condition: maximum number of iterations reached or
costfunction(solution) = 0
best solution = initial solution
n = number of constraints
for i = 1 to n do
    add constraint i
    while stop condition is not met do
        apply tabu search: new solution = S*
        if costfunction(S*) < costfunction(bestsolution) then
            bestsolution = S*
        else
            if number of unimproving iterations > CN then
                change neighbourhood
            else
                increment(number of unimproving iterations)
            end if
        end if
    end while
end for
```

The approach leads to solutions of similar or better quality (see Section 7) compared to the approach in which all the constraints are taken into consideration in one go (see Algorithm 1).

Solving timetabling problems in different stages is a common technique reported on in literature. Lewis [38] discusses it in his survey paper on metaheuristics for university timetabling problems. In the articles discussed, the first stage is used to find feasible solutions, while in the second stage, the feasible solution is applied as a starting point, to find feasible solutions that satisfy the soft constraints. Our approach is different in that it applies several stages to find a solution satisfying the constraints defined in Section 3.

7. Test results

As explained in Section 4.1, another neighbourhood is chosen after a certain number of unimproving moves. The number of unimproving moves is one of the parameters of the search algorithm. Another parameter is the upper bound of the dynamic tabu list length. We conducted several experiments for six sets of parameter settings and describe them in this section. Table 2 contains the six sets of parameter settings. We will discuss the first set of Table 2, the other sets are analogous. The upper limit of the tabu list length of the first parameter setting is 100 and the number of unimproving moves before changing neighbourhoods is 500. The six sets of parameter settings are chosen from a larger collection of 26 sets of parameters. Every parameter setting is run for 50,000 iterations. Box plots of the results obtained with 26 parameters settings are presented in Fig. 6. From this collection of parameter settings we make a selection of the six parameters with a 'small' dispersion.

In [26] Di Gaspero and Schaerf describe a similar token-ring search in which every new neighbourhood starts from the best solution found by the previous neighbourhood. In this experiment we examine whether it is useful to follow the approach of [26]. We executed for each of the six previous discussed parameter settings six runs of 50,000 iterations per stage each (see Fig. 7). Per run, we execute

Table 2
Overview of the six sets of parameter settings

	Tabu list length	Number of unimproving moves
Set 1	100	500
Set 2	100	1000
Set 3	200	2000
Set 4	200	800
Set 5	300	1500
Set 6	300	3000

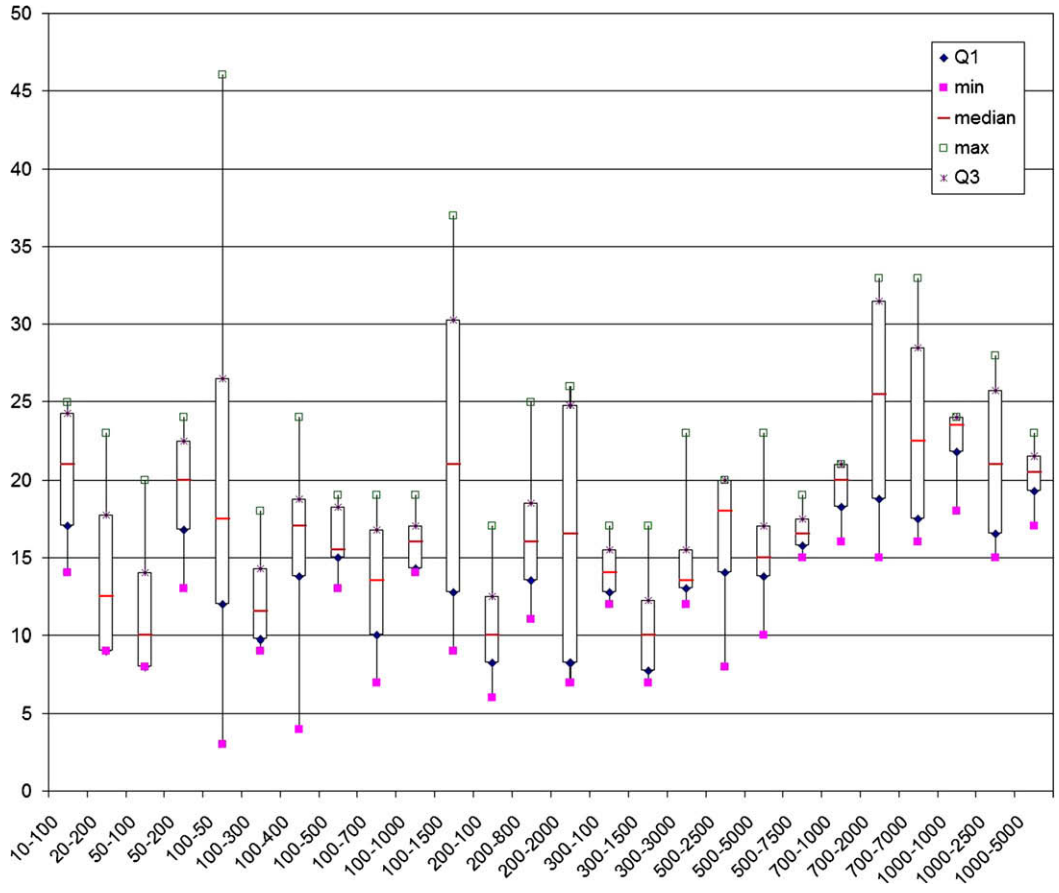

Fig. 6. Box plots of results obtained with 26 parameter settings, which are each run for 50,000 iterations. The first number of each set is the tabu list length and the second number is the number of unimproving moves before changing neighbourhoods.

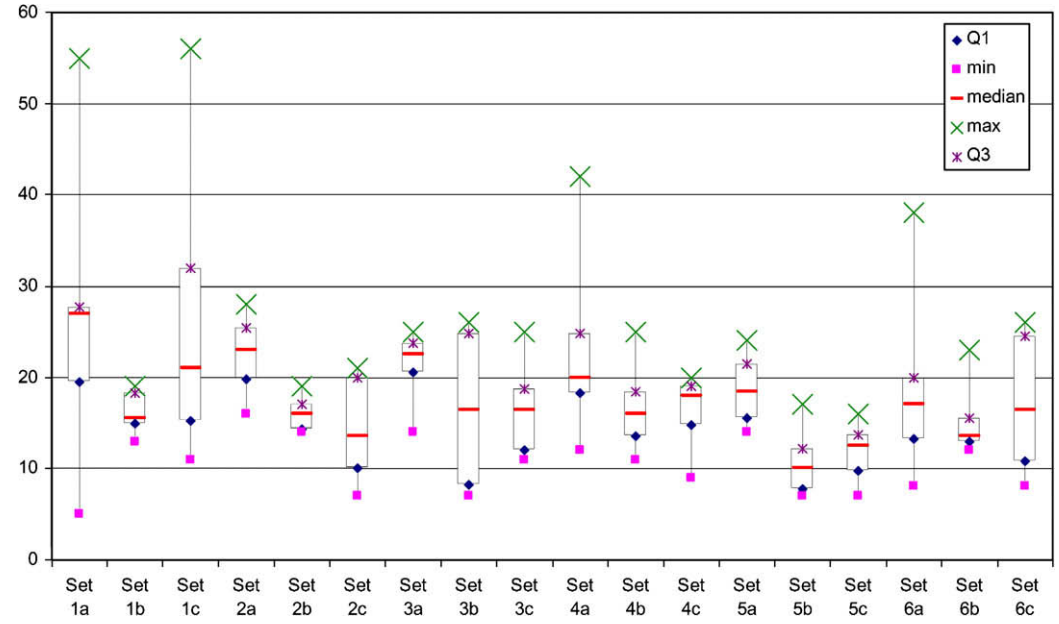

Fig. 7. Comparison between starting from the best solution found in the previous neighbourhood ('a' sets), starting from the last solution obtained in the previous neighbourhood ('b' sets) and selecting the best or the last solution obtained with the previous neighbourhood at random ('c' sets).

experiments in which different solutions (best, current, or the solution that is randomly chosen between the best or current solution) are passed on to the next neighbourhood. In Fig. 7 the result of the tests that take the best solution of

Table 3
For each of the six parameter settings, the cost of nine runs is presented, when using the multi-stages technique

S1:100-500	S2:100-1000	S3:200-2000	S4:200-800	S5:300-1500	S6:300-3000
12	12	5	5	13	8
13	12	12	19	10	15
11	12	13	6	11	18
13	12	17	9	18	6
11	10	6	8	15	7
7	7	14	10	11	15
16	6	14	7	9	6
15	9	18	5	7	11
14	27	6	12	10	18

The maximum number of iterations is 75,000 for every stage.

Table 4
Cost for six different parameter setting during nine runs, when solving all-constraints-at-once

S1:100-500	S2:100-1000	S3:200-2000	S4:200-800	S5:300-1500	S6:300-3000
40	19	5	11	20	22
20	15	13	27	16	13
11	16	19	14	35	13
17	13	21	4	14	9
11	9	11	21	6	19
19	10	11	21	11	13
12	5	17	11	19	20
16	6	13	31	14	16
26	8	10	31	21	17

Maximum number of iterations is 100,000.

the previous neighbourhood are marked Set *i*a, with *i* ranging between 1 and 6. The result of the tests that take the last solution obtained with the previous neighbourhood are indicated by Set *i*b, with *i* ranging from 1 to 6. A combination of both previous methods is presented by the Set *i*c, with *i* ranging from 1 to 6, where we make a random choice whether we begin from the best or from the last solution obtained in the previous neighbourhood. From Fig. 7 it is clear that the result of the *b* test run sets (where the last solution obtained with the previous neighbourhood is used as an initial solution for the next neighbourhood) are among the best. The results are less dispersed than for the other methods, and the median of each *b* test run is less than or equal to the median of the other test runs (except for the second set of test runs). Based on these experiences we decided to conduct the rest of the experiments with the last solution obtained with the previous neighbourhood as the initial solution for the next neighbourhood.

In the next experiment we investigate the efficiency of the multi-stages approach of Section 6 compared to solving all the constraints at once. For each of the six sets of parameters, we executed nine runs. Table 3 presents the results obtained with each of the parameter settings. For the same six sets of parameters, we conducted an analogous experiment, in which all the constraints were evaluated together. Every parameter setting is executed nine times. The results are presented in Table 4. Both approaches' results are presented in Fig. 8. Except for the second parameter set, the medians of the results of each parameter set obtained via the multi-stages approach are smaller than the corresponding median of the results obtained through the all-at-once approach. It is thus appropriate to conclude that there is a higher probability of finding a better solution with the multi-stages approach compared to the all-constraints-at-once method. In the last experiment we apply the multi-stage approach to the fifth parameter set, with

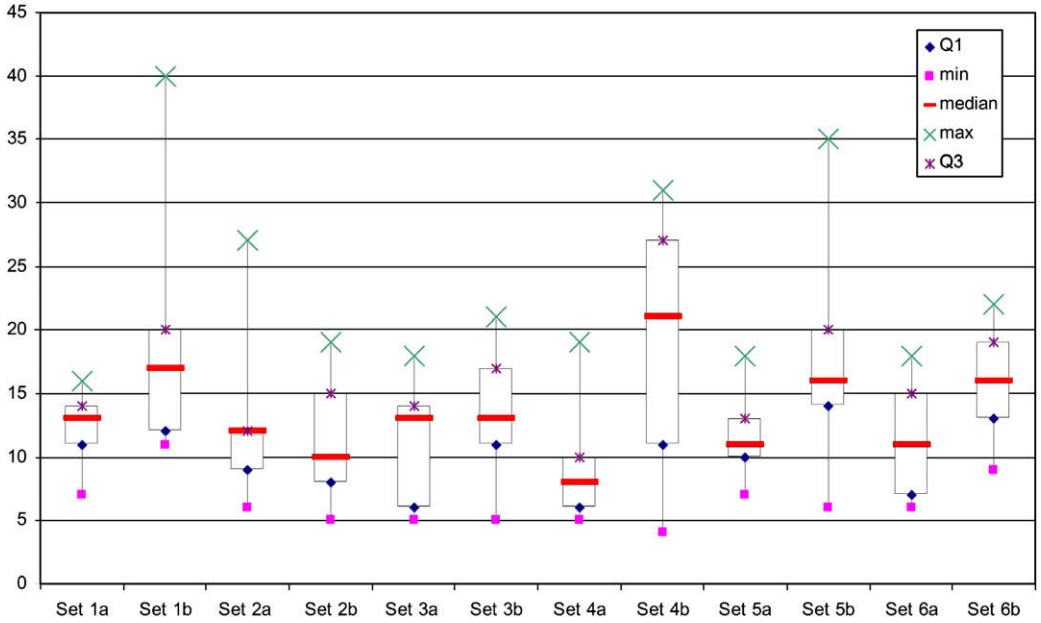

Fig. 8. Box plots of the costs when solving the constraints in different stages ('a' sets) and solving all-constraints-at-once ('b' sets).

Table 5
Cost and elapsed time in seconds for nine runs with the same parameter setting (300-1500)

Cost	Elapsed time	Number of iterations
6	104,292	180,638
6	117,956	197,273
10	90,118	143,741
4	109,627	183,879
5	94,467	167,373
3	87,920	151,041

Maximum number of iterations is 200,000 per stage.

Table 6
The total cost is split up into the contribution per constraint

Constraints				Total
C1:C3	C2:C1	C3:C5	C4:C4	C1 + C2 + C3 + C4: total
0	3	0	7	10
0	3	0	3	6
0	0	0	4	4
0	0	0	5	5
0	0	0	3	3
0	3	0	3	6

The maximum number of iterations per stage is 200,000.

a maximum number of iterations equal to 200,000 per stage. From Table 5 it can be deducted that the longer a tabu search algorithm is executed the better the results become. With these settings the algorithm takes quite a long time to come up with a solution. For practical applications, long computation times are acceptable since the timetabling process is carried out only twice a year, at the beginning of each semester. The presented results correspond to the sum of the violations of the constraints. As described in Section 4.3 the constraints that are evaluated by the cost function are *C1*, *C3*, *C4* and *C5*. In Table 6 we split the total cost of Table 5 up in separate parts, which correspond to the violations of the four constraints. From this table, one can see that the *C4* constraint is hard to satisfy. As mentioned in Section 4.5, this is due to the fact that we assigned rooms to the lab sessions based on the current manually constructed timetable. By doing so, we probably exclude other appropriate rooms from the solution.

8. Conclusion

In this paper we present a decomposed heuristic for solving a large, complex real-world timetabling problem. By grouping similar lectures into pillars first, the remaining timetabling problem becomes less complex. The search space for the reduced problem is considerably smaller than the search space for the original problem. The reduced timetabling problem is then solved by sequentially evaluating the constraints one by one, in such a way that the obtained solution of the first stage is applied as the starting solution of the second stage and so on. We experienced that for this problem this procedure leads to similar or better solutions, compared to trying to solve all the constraints at once.

In future research we will measure the effect of alternating the construction of the pillars and the improvement algorithm. Re-iterating the construction phase might lead the subsequent search away from local optima that result from arbitrary subject combinations in the pillars. We will also experiment with modifying the size of student groups during the search along the lines of the student regrouping reported upon in [30]. In future research we will also include the personal preferences of the lecturers, the specific weeks in which lectures should be organized – instead of only taking into account the total number of occurrences of each lecture – and the different room types – in contrast to our current approach in which we assign lab sessions to lab rooms beforehand due to lack of information.

References

[1] S. Abdulla, Heuristic approaches for university timetabling problems, Ph.D. Thesis, School of Computer Science and Information Technology, University of Nottingham, June 2006.
[2] S. Abdullah, E.K. Burke, B. McCollum, An investigation of variable neighbourhood search for university course timetabling, in: Proceedings of the 2nd Multi-disciplinary International Conference on Scheduling: Theory and Applications (MISTA), New York, USA, pp. 413–427, July 2005.
[3] S. Abdullah, E.K. Burke, B. McCollum, Using a randomised iterative improvement algorithm with composite neighbourhood structures for university course timetabling, in: Proceedings of the Metaheuristic International Conference 2005 (MIC'05), CD-Rom, Austria, Augustus 2005.
[4] S.M. Al-Yakoob, H.D. Sherali, Mathematical programming models and algorithms for a class faculty assignment problem, European Journal of Operational Research 173 (2) (2006) 488–507.
[5] R. Alvarez-Valdes, E. Crespo, J.M. Tamarit, Assigning students to course sections using tabu search, Annals of Operations Research 96 (2000) 1–16.
[6] A.S. Asratian, D. de Werra, A generalized class teacher model for some timetabling problems, European Journal of Operational Research 143 (3) (2002) 531–542.
[7] V.A. Bardadym, Computer-aided school and university timetabling: The new wave, Selected and Revised Papers of the First International Conference on Practice and Theory of Automated Timetabling, (PATAT 1995), Edinburgh, LNCS, vol. 1153, Springer, 1996, pp. 22–45.
[8] E.K. Burke, Y. Bykov, J. Newall, S. Petrovic, A time-predefined approach to course timetabling, Yugoslav Journal of Operations Research 13 (2) (2003) 139–151.
[9] E.K. Burke, D.G. Elliman, R.F. Weare, A university timetabling system based on graph colouring and constraint manipulation, Journal of Research on Computing in Education 27 (1) (1994) 1–18.
[10] E.K. Burke, G. Kendall, E. Soubeiga, A tabu-search hyperheuristic for timetabling and rostering, Journal of Heuristics 9 (6) (2003) 451–470.
[11] E.K. Burke, J.H. Kingston, D. De Werra, Applications to timetabling, in: Jonathan Gross, Jay Yellen (Eds.), Handbook of Graph Theory, Chapman Hall/CRC Press, 2004, pp. 445–474.
[12] E.K. Burke, B.L. MacCarthy, S. Petrovic, R. Qu, Case-based reasoning in course timetabling: an attribute graph approach, in: Proceedings of the 4th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development, Lecture Notes in Computer Science, vol. 2080, Springer, 2001, pp. 90–104.

[13] E.K. Burke, B.L. MacCarthy, S. Petrovic, R. Qu, Knowledge discovery in hyper-heuristic using case-based reasoning on course timetabling, in: Selected and Revised Papers of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT) 2002, Lecture Notes in Computer Science, vol. 2740, Springer, 2003, pp. 276–286.
[14] E.K. Burke, B.L. MacCarthy, S. Petrovic, R. Qu, Multiple-retrieval case based reasoning for course timetabling problems, Journal of the Operational Research Society 57 (2) (2006) 148–162.
[15] E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, A graph-based hyper-heuristic for educational timetabling problems, European Journal of Operational Research 179 (2007) 177–192.
[16] E.K. Burke, S. Petrovic, Recent research directions in automated timetabling, European Journal of Operational Research 140 (2) (2002) 266–280.
[17] E.K. Burke, S. Petrovic, R. Qu, Case based heuristic selection for timetabling problems, Journal of Scheduling 9 (2) (2006) 115–132.
[18] M.P. Carrasco, M.V. Pato, A Potts neural network heuristic for the class/teacher timetabling problem, Applied Optimization, Metaheuristics: Computer Decision-making Book, Kluwer Academic Publishers, 2004, pp. 173–186.
[19] M.P. Carrasco, M.V. Pato, A comparison of discrete and continuous neural network approaches to solve the class/teacher timetabling problem, European Journal of Operational Research 153 (2004) 65–79.
[20] M.W. Carter, G. Laporte, Recent development in practical course timetabling, in: Selected and Revised Papers of the Second International Conference on Practice and Theory of Automated Timetabling, (PATAT 1997), LNCS, vol. 1408, Springer, Toronto, 1998, pp. 3–19.
[21] C. Cheng, L. Kang, N. Leung, M. White, Investigations of a constraint logic programming approach to university timetabling, in: Selected Papers of the First International Conference, LNCS, vol. 1153, Edinburgh, UK, 1995, pp. 112–129.
[22] A. Colorni, M. Dorigo, V. Maniezzo, Metaheuristics for high-school timetabling, Computational Optimization and Applications 9 (3) (1998) 277–298.
[23] D. Costa, A tabu search algorithm for computing an operational timetable, European Journal of Operational Research 76 (1) (1994) 98–110.
[24] S. Daskalaki, T. Birbas, Efficient solutions for a university timetabling problem through integer programming, European Journal of Operational Research 160 (2005) 106–120.
[25] D. de Werra, An introduction to timetabling, European Journal of Operational Research 19 (2) (1985) 151–162.
[26] L. Di Gaspero, A. Schaerf, Multi-neighbourhood local search with application to course timetabling, in: Selected Revised Papers of the Fourth International Conference on Practice and Theory of Automated Timetabling (PATAT 2002), LNCS, vol. 2740, Springer, 2003, pp. 262–275.
[27] L. Di Gaspero, A. Schaerf, A multineighbourhood local search solver for the timetabling competition TTComp 2002, in: Proceedings of the Fifth International Conference, on Practice and Theory of Automated Timetabling, (PATAT 2004), Pittsburgh, USA, 2004, pp. 475–478.
[28] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Berlin, 1997.
[29] C. Guéret, N. Jussien, P. Boizumault, C. Prins, Building university timetables using constraint logic programming, in: Selected and Revised Papers of the 1st International Conference on Practice and Theory of Automated Timetabling, (PATAT 1995), LNCS, vol. 1153, Springer, Edinburgh, 1996, pp. 130–145.
[30] A. Hertz, Tabu search for large scale timetabling problems, European Journal of Operational Research 54 (1991) 39–47.
[31] A. Hertz, Finding a feasible course schedule using tabu search, Discrete Applied Mathematics 35 (1992) 255–270.
[32] J.J. Hopfield1, D.W. Tank, Neural computation of decisions in optimization problems, Biological Cybernetics 52 (3) (1985) 141–152.
[33] International Timetabling Competition: <http://www.idsia.ch/Files/ttcomp2002/>.
[34] B. Jaumard, J. Cordeau, R. Morales, Efficient timetabling solution with tabu search, Working Paper, Available from Metaheuristics Network – International Timetabling Competition (<http://www.idsia.ch/Files/ttcomp2002/jaumard.pdf>).
[35] J.H. Kingston, A tiling algorithm for high school timetabling, in: Selected and Revised Papers of the Fifth International Conference on Practice and Theory of Automated Timetabling, (PATAT 2004), LNCS, vol. 3616, Springer, Pittsburgh, 2005, pp. 208–225.
[36] P. Kostuch, The university course timetabling problem with a three-phase approach, in: Selected and Revised Papers of the Fifth International Conference on Practice and Theory of Automated Timetabling, (PATAT 2004), LNCS, vol. 3616, Springer, Pittsburgh, 2005, pp. 109–125.
[37] R. Lewis, Metaheuristics for University Course Timetabling, Ph.D. Thesis, August 2006.
[38] R. Lewis, A survey of metaheuristic-based techniques for university timetabling problems, OR Spectrum 30 (1) (2007) 167–190.
[39] B. McCollum, University timetabling: Bridging the gap between research and practice, in: Proceedings of Sixth International Conference on Practice and Theory of Automated Timetabling, (PATAT 2006), Brno, 2006, pp. 15–35.
[40] K. Nonobe, T. Ibaraki, A tabu search approach to the CSP (Constraint Satisfaction Problem) as a general problem solver, European Journal of Operational Research 106 (1998) 599–623, Special Issue on Tabu Search.
[41] S. Petrovic, E. Burke, University timetabling, in: J. Leung (Ed.), Handbook of Scheduling: Algorithms, Models, and Performance Analysis, vol. 45, CRC Press, 2004 (Chapter 45), pp. 1–23.
[42] H. Rudova, K. Murray, University course timetabling with soft constraints, in: Selected and Revised Papers of the 4th International Conference on Practice and Theory of Automated Timetabling, (PATAT 2002), LNCS, vol. 2740, Springer, Gent, 2003, pp. 311–329.
[43] A. Schaerf, A survey of automated timetabling, Artificial Intelligence Review 13 (2) (1999) 87–127.
[44] A. Schaerf, Local search techniques for large high school timetabling problems, IEEE Transactions on Systems Man and Cybernetics Part A: Systems and Human 29 (4) (1999) 368–377.
[45] K. Socha, M. Sampels, M. Manfrin, Ant algorithms for the university course timetabling problem with regard to the state-of-the-art, in: Proceedings of EvoCOP 2003, 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization, LNCS, vol. 2611, 2003, pp. 334–345.
[46] A. Tripathy, Computerised decision aid for timetabling – A case analysis, Discrete Applied Mathematics 35 (3) (1992) 313–323.