University Course Timetabling with Genetic Algorithm: a Laboratory Excercises Case Study

Z. Bratković, T. Herman, V. Omrčen, M. Čupić, D. Jakobović

University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
{zlatko.bratkovic;tomislav.herman;vjera.omrcen;
marko.cupic;domagoj.jakobovic}@fer.hr

(2009)

Abstract. This paper describes the application of a hybrid genetic algorithm to a real-world instance of the university course timetabling problem. We address the timetabling of laboratory exercises in a highly constrained environment, for which a formal definition is given. Solution representation technique appropriate to the problem is defined, along with associated genetic operators and a local search algorithm. The approach presented in the paper has been successfully used for timetabling at the authors' institution and it was capable of generating timetables for complex problem instances.

1 Introduction

The university timetabling problem and its variations are a part of the larger class of timetabling and scheduling problems. The aim in timetabling is to find an assignment of entities to a limited number of resources while satisfying all the constraints. Two forms of university timetabling problems may be recognized in today's literature: *examination timetabling* and *course timetabling* problems, where the differences between those types usually depend on the university involved. The problem can be further specialized as either *post enrollment based* or *curriculum based*. In post enrollment problems, the timetable must be constructed in such a way that all students can attend the events on which they are enrolled, whereas in curriculum problems the constraints are defined according to the university curricula and not based on enrollment data.

Due to inherent problem complexity and variability, most of the real-world university timetabling problems are NP-complete. This calls for the use of heuristic algorithms that do not guarantee an optimal solution, but are in many cases able to produce a solution that is "good enough" for practical purposes. It has been previously shown that metaheuristic-based techniques (such as evolutionary algorithms, tabu-search etc.) are especially well suited for solving these kinds of problems, and this work is an example of that approach.

The paper focuses on a *laboratory exercise timetabling problem* (LETP), which we define as a type of university course timetabling problem (UCTP). The motivation for this work emerged from a need for automated timetable generation at the authors' institution. The timetables could no longer be constructed using traditional methods due to the increased complexity caused by

teaching curriculum reforms. The work described here is a part of the research of two different metaheuristics for timetable construction: *genetic algorithm* (GA) and *ant colony optimization* (ACO). In this paper we give a formal definition of the LETP problem and apply a hybrid genetic algorithm to solve real-world instances of the problem. The main contributions of the paper are the definition of solution representation and genetic operators that are tailored to the complex set of timetable constraints, as well as a local search algorithm for additional solution refinement. The result is a GA-based system, capable of producing usable timetables, that is highly adaptive to various idiosyncratic requirements that may be imposed by a particular institution.

A concise overview of some general trends in automated timetabling can be found in [1–5]. Many university course timetabling problems in literature have been intentionally simplified, since real-world examples hold numerous features that make algorithm implementation and performance tracking complicated. A general trend that can be noticed in recent years is that the research focuses on metaheuristic algorithms, instead of application-specific heuristics [2, 5, 3].

While there is ample research based on simplified artificial problem instances [6–10], we were unable to find an approach that would encompass the requirements imposed by post-enrollment laboratory exercises timetabling, particularly when applied to a large number of students and courses. Since the complexity of the problem significantly depends on the defined constraints [11, 12], we present a variant of the problem with additional characteristics which are particularly suited for laboratory timetabling.

The remainder of this paper is organized as follows: Section 2 introduces the actual timetabling problem and in Section 3 we elaborate our approach. Section 4 presents the results while Section 5 concludes the paper and discusses future work.

2 University Course Timetabling Problem

2.1 Problem Statement

Timetable construction is an NP-complete combinatorial optimization problem [13] that consists of four finite sets: a set of meetings, a set of available resources (e.g., rooms, staff, students), a set of available time slots and a set of constraints. The problem is to assign resources and time slots to each given meeting, while maintaining constraints satisfied to the highest possible extent. University course timetabling problem (UCTP) is a timetabling problem where a set of courses and a set of attending courses for each of the students is defined, a course being a set of events that need to take place in the timetable. The main characteristic that discriminates the university course timetabling from other types of timetabling problems is the fact that students are generally allowed to choose courses they wish to enroll [14]. A set of constraints is usually divided into *hard constraints*, whose violation makes the timetable suggestion infeasible, and *soft constraints*, rules that improve the quality of timetables, but are allowed to be violated.

Since this description of UCTP usually does not cover all the requirements imposed by a particular institution, we define additional elements of the problem which allow its application to more specific timetabling instances. The presented model was used for organizing laboratory exercises at the authors' institution, but it can also be used to describe various instances of course timetabling.

2.2 Laboratory Exercise Timetabling Problem

We define the laboratory exercise timetabling problem (LETP) as a six-tuple:

$$LETP = (T, L, R, E, S, C)\,,$$

where T is a set of *time quanta* in which the scheduling is possible, L is a set of limited assets present at university, R is a set of rooms, E is a set of events that need scheduling, S is a set of attending students, and C is a set of constraints.

- A set of time quanta in which scheduling the exercises is possible is denoted T. We assume that the durations of all the exercises can be quantified as a multiple of a fixed time interval, a *time quantum*, denoted t_q. A *time slot* is defined as one or more consecutive time quanta in the timetable. When choosing the duration of the quantum, one needs to make a trade-off between finer granularity of scheduling in time and the larger size of the search space. In all of the examples below, we will assume that the quantum duration is 15 minutes and the exercises can be scheduled between 8:00 and 20:00.
- A set of all the limited assets (resources) available for laboratory exercises is denoted L. One can concieve assets that are required for many different laboratory exercises, but are available in only limited amounts. For example, a laboratory exercise may use a commercial software package, but university can be in possession of only a limited number of licenses. This limits the concurrency of laboratory exercises, as the assets are shared among various courses. For each resource $l \in L$, a quantity, denoted $quantity_l$, is defined as the number of workplaces that can use the resource concurrently.
- With each room, we associate a pair of properties: $(size_r, T_r)$, $size_r \in \mathbb{N}$, $T_r \subseteq T$, defined as follows:
  - The *workplace* is defined as an atomic room resource varying from room to room, such as seats in ordinary classrooms, computers in computer classrooms, etc. For each room $r \in R$, the number of workplaces, denoted $size_r \in \mathbb{N}$, is defined.
  - The rooms required for laboratory exercises could be used for other educational purposes such as exams or lectures. Therefore, for each room $r \in R$ a set of time quanta in which the room is not occupied, denoted $T_r \subseteq T$ is defined.
- Event $e \in E$ is defined as a single laboratory exercise of a course. A single event may be scheduled in one or more *event instances* in different time slots. For example, one instance of the event may be held at 8:00-9:00 on Monday (for one group of students) and the other at 10:00-11:00 on Tuesday (for the rest of the students). Each event has a set of properties defined as follows:

- Each event e has a duration, denoted $dur_e \in \mathbb{N}$, defined as a multiple of time quanta. For example, the event "Artificial Intelligence exercise 1" may have a duration of $dur_{AI} = 4$ quanta, or 60 minutes.
- Event timespan, denoted $span_e \in \mathbb{N}$, can be defined to ensure that all instances of the event are scheduled within a specified time interval. It is defined as a difference between the end time of the last instance and the start time of the first instance of the event. For example, let us consider an exercise whose part is a brief quiz. To ensure fairness of the test for all students, the staff may demand that all instances of the exercise have to take place within one day (or, even more restrictive, within a certain number of consecutive time quanta).
- The events may take place in different types of rooms, varying from computer classrooms to specialized electronic or electrical engineering laboratories. Hence, for each event the acceptable room set $R_e \subseteq R$ is defined.
- For each event e, a nonempty subset of suitable time quanta, denoted $T_e \subseteq T$ can be defined. For example, the staff of a course may demand events to be scheduled only on Wednesday and Friday, due to the organisation issues.
- Certain events may require the use of one or more limited assets. The set of assets used by the event is denoted $L_e \subseteq L$.
- Usually, members of teaching staff are present at events in order to help the students carry out the exercise. The number of staff present may depend on the event and the room the event is held in. The staff can be viewed as a form of limited asset of an event. The number of staff available for event e is denoted $staff_e \in \mathbb{N}$. The value $usage_{e,r}$ is defined as the number of teaching staff used for event e being scheduled at room r. The usage is undefined for $r \notin R_e$.
- A maximum number of rooms used concurrently for an event, denoted $rooms_e \in \mathbb{N}$, can be defined.
- Some pairs of events may require a partial ordering relation between them. This requirement is apparent in courses with exercises that build on top of each other. For example, event "AI exercise 1" and event "AI exercise 2" need to be scheduled in the same week, but it must be ensured that the second event is scheduled after the first. Also, the students must have at least one whole day between these exercises to prepare properly. A relation, denoted $\succ_d$ can be defined for a pair of events, $\succ_d$: $E \times E$. The events are in relation $e_2 \succ_d e_1$ iff each instance of e_2 is scheduled at least d days after the last instance of event e_1.
- For each event, the number of students per workplace, denoted $spw_e \in \mathbb{N}$ is defined. For example, the staff of some courses may prefer that students in computer classrooms do their exercise on their own, and other prefer group-work, allowing two or more students per workplace.

– Set S is the set of students that are to be scheduled. Each student $s \in S$ has a set of properties defined as follows:

- As the students are required to attend lectures and exams along with the exercises, it is not possible to assume that the student will always be available for the event. Thus, for each student a set of time quanta when the student is free $T_s \subseteq T$ is defined.
- Depending on the student's selection of enrolled courses, the student is required to attend a nonempty set of events, denoted $E_s \subseteq E$.

– The requirements of the courses are represented in a set of constraints C. The constraints are divided into hard constraints C_h, which are essential for the courses, and soft constraints C_s, which may require some manual intervention if they are not met. Hard constraints C_h are defined as follows:

  - The room can be occupied by at most one event at any time.
  - The room must be free for use at the time scheduled.
  - Event can be placed only in a room $r \in R_e$ that is suitable for that event.
  - The room r, when used for event e, can accommodate no more than $size_r \cdot spw_e$ students.
  - Event e can be held only at the time defined as suitable for that event.
  - When the event e is placed in a schedule, it occupies the consecutive dur_e quanta belonging to the same day, dur_e being the duration of the event e.
  - Event e must be scheduled within the total $span_e$ for the event.
  - When the ordering relation $\succ_d$ exists between two events, it must be satisfied in the timetable.
  - Asset $l \in L$ can be used concurrently on at most $quantity_l$ workplaces. Thus, the number of students concurrently attending an event is limited by the number of assets the students consume.
  - When the event is placed concurrently in rooms, enough teaching staff must be available to attend the event.
  - Event e can be concurrently placed in at most $rooms_e$ rooms.
  - The students must attend all the events they are enrolled in.

  The set of soft constraints C_s contains two elements:

  - Students can attend an event only at the time when she or he is free from other educational activities.
  - Students can attend only one event at a time.

  Defining these constraints as soft may seem irrational, but the reasoning behind this is as follows: 'hard' constraints are simply those that are at all times satisfied for any individual (i.e., any solution) in GA population in our implementation, whereas for the 'soft' constraints this may not be the case. 'Soft' constraints are defined as such because it was not known in advance whether there even exists a solution that satisfies all the constraints (given the complex requirements). In other words, our approach tries to find the best solution within the imposed constraints and possibly to give a feedback to the course organisers if some are still severely violated. In the remainder of the text, the term 'feasible solution' denotes the one that satisfies only the hard constraints as defined above.

3 Solving LETP with Genetic Algorithm

3.1 Solution Representation

The adequate solution representation for the LETP suggests itself from the definition of the problem: each solution *sol* (a timetable) contains all the events that have to be scheduled. Each event e is, in turn, scheduled in one or more *event instances*, where each instance is defined with the following: a time slot, a subset of feasible rooms, and a subset of students that attend event e. Events must be allocated in enough instances so that no students are left unscheduled. We consider an event to be the basic unit of heredity in a chromosome.

- *Event instance* $i_e = (ts_i, R_i, S_i)$, where:
  - ts_i denotes the time slot allocated to particular event instance i_e
  - R_i denotes the allocated room set $R_i \subseteq R_e$
  - S_i denotes allocated student subset $S_i \subseteq S_e$
- *Event instance set* $I(e) = \{i_{1e}, \cdots, i_{me}\}$
- *Solution* $sol = \{I(e_1), \cdots, I(e_n)\}$, $\forall e_i \in E$.

3.2 Creation of Initial Population

Each member of the initial population (a single solution) is created using the following procedure:

while set of events E not empty **do**
  select random event e and remove it from E;
  D_e = set of valid days for event e;
  S_e = set of students that attend event e;
  while (D_e not empty) AND (S_e not empty) **do**
    select random day d and remove it from D_e;
    $TS_{e,d}$ = set of valid timeslots for event e in day d;
    while ($TS_{e,d}$ not empty) AND (S_e not empty) **do**
      select random timeslot ts and remove it from $TS_{e,d}$;
      create event instance $i = (ts, R_i, S_i)$ with $R_i = \emptyset$ and $S_i = \emptyset$;
      for every suitable room $r \in R_e$ **do**
        if (r is available in ts) AND (r meets event requirements) **then**
          reserve room r and add it to R_i;
          assign $size_r \cdot spw_e$ students from S_e to r and add them to S_i;
          remove assigned students from S_e;
        end if
      end for
    end while
  end while
  if S_e not empty **then**
    creation unsuccessful;
  end if
end while

In the above procedure, 'valid days' and 'valid timeslots' denote suitable days and slots according to the time constraints, while 'room meets requirements' condition ensures all other requirements are met (such as the use of assets, staff availability, etc.). Thus, every member of the population satisfies the given hard constraints, whereas soft constraints may be violated.

It is obvious that the creation of a solution may not always succeed; the above algorithm only succeeds at some *success rate*, dependent of the given set of events and their requirements. However, even a small success rate allows the creation of the desired number of solutions, since the algorithm is only performed at the beginning of the evolution process. In our experiments the success rate ranged between 5% and 75%, so we were always able to build the population in this way. On the other hand, failure to do so could suggest the infeasibility of the given constraints.

3.3 Fitness Function

Fitness function evaluates the quality of a solution and it is proportional to the number of *conflicts* in a given solution. A conflict is a violation of soft constraints which occurs if a student is scheduled to more than one event in the same time slot. The number of conflicts is equal to the number of *time quanta* during which events overlap. The exact fitness measure is defined as:

$$fitness = \sum_{s \in S} \sum_{t_q \in T} \frac{N_{e,s,t_q} \cdot (N_{e,s,t_q} - 1)}{2}$$

where N_{e,s,t_q} represents the number of events the student is scheduled to in current time quantum. For instance, if a student is scheduled on two events that overlap in four time quanta, then the fitness value equals four. The best solution will have fitness value of zero, meaning that no conflicts exist in the solution.

The calculation of the fitness function may be time consuming since it is necessary to iterate through all the students and all the time slots in a solution. In our implementation, the fitness is evaluated *incrementally* after an individual has changed due to genetic operations. In other words, only the part of the solution that has undergone some changes is reevaluated, which significantly speeds up the fitness calculation.

3.4 Genetic Operators

In order to apply genetic algorithm to aforementioned problem, appropriate genetic operators need to be devised. Due to the fact that every solution needs to satisfy all of the hard constraints, the result of each genetic operation needs to be a solution with the same property. Thus, we introduce especially crafted crossover and mutation operators that are closed over the space of feasible solutions. The crossover operator is defined as follows:

- Let us define $I_{sol}(e)$ as a set of event instances assigned to event $e \in E$ in one particular solution $sol \in P$, where P is current population.

– We define the relation $\otimes$ over $\mathcal{SOL}$ and $\mathcal{I}$ where $\mathcal{I}$ denotes a power set of event instances and $\mathcal{SOL}$ denotes the LETP search space (the set of all possible solutions). The characteristic function of the relation $\chi : \mathcal{SOL} \times \mathcal{I} \rightarrow \{\mathcal{T}, \mathcal{F}\}$ is defined to be true iff the insertion of a particular set of instances $I \in \mathcal{I}$ into $sol \in \mathcal{SOL}$ does not cause violation of hard constraints.
– Now, we consider two particular solutions $a \in P$ and $b \in P$ contained in current population P as parents. Crossover operator takes the parent solutions a and b and produces a solution *child*. Having the relation $\otimes$ defined as above, we describe the crossover operator using the following pseudo code:

Crossover (soluton a, solution b)
solution $child = \emptyset$;
randomly select subset of events $E_1 \subset E$;
$E_2 = E - E_1$;
for all events $e \in E_1$ **do**
  add set of instances $I_a(e)$ to *child*;
end for
for all events e in E_2 **do**
  if $child \otimes I_b(e)$ **then**
    add set of instances $I_b(e)$ to *child*;
  else
    randomly allocate $I(e)$;
    if $child \otimes I(e)$ **then**
      add $I(e)$ to *child*;
    else
      discard *child*;
    end if
  end if
end for

In the above procedure, the creation of the child may not always succeed because the allocation of new event instances cannot always be performed without the violation of hard constraints. In that case, the procedure is repeated until it succeeds or a predefined number of repetitions is performed.

The mutation operator selects a random event and removes the associated set of instances from the solution. The removed event instances are then generated randomly with regard to hard constraints. This operation may be repeated more than once as defined by the *mutationLevel* parameter (defined in subsection 3.6). The mutation operator is applied on child solution after the crossover operation (with a certain probability) and it uses the following pseudo code:

Mutation (solution a)
n = random value between (1, *mutationLevel*);
for $i = 1$ to n **do**
  randomly select event e and remove instance set $I_a(e)$ from solution a;
  randomly allocate new instance set $I(e)$ so that $a \otimes I(e)$ holds;
  add $I(e)$ to a;
end for

3.5 Local Search Algorithm

Since genetic operators are designed to satisfy only hard constraints, the number of conflicts in a solution (which are the consequence of soft constraints violation) may increase during the evolution. To counter that, we implemented a fast local search algorithm which improves the solution significantly.

The local search algorithm operates on students with conflicted schedules within a single solution. The algorithm randomly choses a single student among those and tries to find another instance (in another time slot) of the same event that causes no conflicts for the chosen student. If such instance is not found, the algorithm selects a random instance, but in both cases the student is allocated to another instance. This operation is repeated (for randomly chosen students) until a certain number of consecutive iterations without fitness improvement occurs, where the number of repetitions is predefined. Local search is applied to every individual produced by the crossover operator or modified by mutation.

The primary goal of local search is further reduction of soft constraint violations, since genetic operators are designed to optimize room allocation in different time slots. After the rooms are allocated, the assignment of students to event instances is a subproblem contained in LETP that local search is used to optimize.

3.6 GA Parameters and Adaptation

The presented implementation has an adaptive parameter called *mutationLevel*. Mutation level defines the maximum number of events that will be rescheduled when a solution undergoes mutation. Mutation level is updated if stagnation is detected. Stagnation occurs when there is no improvement in population in particular number of generations called *stagnationThreshold*. If stagnation is detected the following parameter updates are performed:

- $stagnationThreshold = stagnationThreshold \cdot 1.5$;
- $mutationLevel = \min(mutationLevel + 1, maxMutationLevel)$;

When improvement occurs, the adaptive parameters are reset. The parameters of the genetic algorithm are shown in Table 1.

4 Results

The described hybrid GA was successfully applied to laboratory exercises scheduling at the authors' institution during the last two semesters. In this paper we present the actual results which were adopted in the students' schedule and used by the departments organizing the exercises. In each semester, there are several periods during which the exercises may take place. The number and durations of those periods are determined by the curriculum and the lectures schedule, which is made prior to the laboratory scheduling. Each scheduling period, denoted a *laboratory cycle*, is in fact a separate and independent timetabling problem with associated dataset.

Table 1. Genetic algorithm parameters

parameter	value
population size	30
selection algorithm	tournament selection
tournament size	3
individual mutation probability	55 %
initial/maximum mutation level	1 / 10
initial stagnation threshold	5
stop criteria	10000 generations or min fitness = 0

The timetabling problems at authors' institution had different durations and a greatly varying number of events and attending students (total number of students ranged from several hundred to more than two thousand). To estimate problem size and dificuilty, we introduce the *student events sum* $S_{s,e}$ as a measure of a single laboratory cycle complexity. $S_{s,e}$ is an aggregated number of events that each student must attend, defined as $S_{s,e} = \sum_{s \in S} |E_s|$.

The algorithm was used in twelve different timetabling problems as listed in the Table 2, where N_e denotes the number of events to be scheduled and *days* is the cycle duration (the statistics are derived from 10 runs on each problem). We managed to find a schedule with no conflicts for ten out of twelve problem instances. The remaining two instances were subsequently proven to be impossible to schedule without conflicts, by identifying a single event with infeasible requirements posed by course organizers, in combination with existing lectures' schedule. For instance, the best found fitness value of 24 in the first cycle resulted from six students that were double booked in four time quanta each. The actual problem datasets are available at http://morgoth.zemris.fer.hr/jagenda.

To illustrate the effectivenes of hybrid algorithm components, we also include in Table 2 *the best results* obtained with random generation of solutions, local search only (LS) and genetic algorithm without local search (GA).

5 Conclusions and Future Work

This paper describes the application of a hybrid genetic algorithm to a complex timetabling problem and shows that, with appropriate solution representation and genetic operators, it is possible to obtain solutions of a very good quality.

The problem is formulated as a laboratory exercises timetabling problem and its formal definition is given. Although highly constrained, we believe that the definition is quite general and may be applied to a wider class of problems, simply by omitting some of the requirements or further specializing the existing ones. For instance, the set L of limited assets may be removed or the time quantum duration may be changed if the problem at hand allows it. The scheduling of the

Table 2. Algorithm performance on twelve different *laboratory cycles*

				fitness - best of run				comparison		
cycle	N_e	$S_{s,e}$	days	min	avg	max	st.dev.(σ)	rnd.	LS only	GA only
1	51	7081	9	24	228.6	286	77.2	14615	1937	6647
2	9	2104	5	0	0.0	0	0	4565	399	2667
3	11	2553	5	0	16.2	32	10.1	5839	395	2522
4	16	4868	5	0	28.4	146	44.9	9755	684	4346
5	6	1586	5	0	0.0	0	0	3771	379	2668
6	8	2471	5	0	0.0	0	0	4838	610	2320
7	15	3648	5	0	7.4	12	5.5	7830	623	3424
8	19	5430	4	82	89.9	106	9.6	13486	1505	6021
9	9	3843	5	0	59.0	126	54.95	7088	506	4166
10	8	1701	5	0	3.55	8	5.81	4254	41	3210
11	11	1783	5	0	124.8	132	2.5	3646	466	1974
12	21	5934	5	0	184.3	292	48.1	12477	10540	6476

events may be forced to a single instance by defining event timespan equal to the event duration, a whole semester timetable can be generated by repeating a single cycle, etc. With all those adaptations, the solution representation and evolutionary algorithm need not be changed at all, although for a greatly simplified variant of the problem, some other algorithm may consequently obtain the results faster.

The algorithm presumes that every solution in every stage of evolution process satisfies the defined set of hard constraints. To achieve that, we define a solution initialization procedure for building the initial population. The procedure may not always succeed in creation of a valid solution, so it must be repeated until a desired number of solutions are created. While the creation success rate may be low, it still does not present a problem since the initialization of the whole population is performed only once, at the beginning of the evolution. The drawback of this approach is that it is possible to have a set of requirements that would make it impossible to create a valid solution. However, in all our experiments we have not encountered that situation.

The genetic operators preserve the mentioned property of the solutions while trying to combine 'adequately' scheduled events. The efficiency of the operators when used without local search is generally not high, and in that case the convergence is relatively slow. The inclusion of the local search operator, that concentrates on student allocation only, has proven beneficial to the optimization process as it improved the quality of the individuals and accelerated the convergence.

A possible improvement could be gained by devising and experimenting with different variants of crossover and mutation. Furthermore, a systematic experimentation is needed regarding the values of various parameters used in the

algorithm, since they can have a significant impact on the performance. Nevertheless, even the relatively simple operators used in the algorithm succeeded in producing solutions of acceptable quality.

References

1. McCollum, B.: University timetabling: Bridging the gap between research and practice. In E Burke, H.R., ed.: PATAT 2006 — Proc. 6th Int. Conf. on the Practice And Theory of Automated Timetabling, Masaryk University (2006) 15 – 35
2. Qu, R., Burke, E., McCollum, B., Merlot, L., Lee, S.: A Survey of Search Methodologies and Automated Approaches for Examination Timetabling. Technical Report NOTTCS-TR-2006-4, School of CSiT, University of Nottingham (2006)
3. Burke, E., Petrovic, S.: Recent research directions in automated timetabling. European Journal of Operational Research **127**(2) (2002) 266–280
4. Schaerf, A.: A survey of automated timetabling. In: 115. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X (1995) 33
5. Lewis, R.: A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum (2007)
6. Azimi, Z.: Hybrid heuristics for Examination Timetabling problem. Applied Mathematics and Computation **163**(2) (2005) 705–733
7. Azimi, Z.: Comparison of Methheuristic Algorithms for Examination Timetabling Problem. Applied Mathematics and Computation **16**(1) (2004) 337–354
8. Rossi-Doria, O., Sample, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., Stützle, T.: A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem. In: The Practice and Theory of Automated Timetabling IV: Revised Selected Papers from the 4th Int. conf., Gent 2002. Volume 2740 of Lecture Notes in Computer Science., Springer, Berlin, Germany (2003) 329–351
9. Socha, K., Sampels, M., Manfrin, M.: Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. In: Proc. of EvoCOP 2003 – 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization. Volume 2611 of Lecture Notes in Computer Science., Springer Verlag, Berlin, Germany (2003) 334–345
10. Socha, K., Knowles, J., Sampels, M.: A $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the University Timetabling Problem. In Dorigo, M., Di Caro, G., Sampels, M., eds.: Proceedings of ANTS 2002 – From Ant Colonies to Artificial Ants: Third International Workshop on Ant Algorithms. Volume 2463 of Lecture Notes in Computer Science., Springer Verlag, Berlin, Germany (2002) 1–13
11. E.K. Burke, K. Jackson, J.K., Weare, R.: Automated university timetabling: The state of the art. The Computer Journal **40**(9) (1997) 565–571
12. van den Broek, J., Hurkens, C., Woeginger, G.: Timetabling problems at the TU Eindhoven. In: PATAT. (2006) 123–138
13. Cooper, T.B., Kingston, J.H.: The complexity of timetable construction problems. In: Proc. of the 1st Int. Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95). (1995) 511–522
14. Gross, J.L., Yellen, J.: Handbook of Graph Theory. CRC Press (2004)